OUT-OF-DISTRIBUTION DETECTION IN LEARNING-ENABLED CYBER-PHYSICAL SYSTEMS

By

Feiyang Cai

Dissertation
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
in

Electrical Engineering

January 31, 2022

Nashville, Tennessee

Approved:

Xenonfon Koutsoukos, Ph.D.

Janos Sztipanovits, Ph.D.

Gabor Karsai, Ph.D.

Abhishek Dubey, Ph.D.

Yuankai Huo, Ph.D.

Dedicated to my Mom and Shanshan.

ACKNOWLEDGMENTS

During my Ph.D. journey, there are many people who need to thank, but first of all, I would like to express my deepest gratitude to my advisor, Dr. Xenofon Koutsoukos, who not only provided me tremendous guidance during my research, but also gave me great encouragement and support when I encountered difficulties. I have learned much from him, including his rigorous academic spirit and breakthrough research vision, which will benefit me throughout my life. It is my great privilege to be his student.

I would also like to thank Dr. Janos Sztipanovits, Dr. Gabor Karsai, Dr. Abhishek Dubey, and Dr. Yuankai Huo for serving on my Ph.D. committee and offering their feedback to this dissertation.

I greatly appreciate the research atmosphere in Institute for Software Integrated Systems, and I would like to thank every member for making this experience unforgettable. In particular, I would like to thank Dr. Zhenkai Zhang, Dr. Jiani Li, Dr. Bradley Potteiger, Dr. Ali Ozdagli, Dimitrios Boursinos, Shreyas Ramakrishna, Nicholas Potteiger, Dr. Dung Hoang Tran, and Nagabhushan Mahadevan for their help and collaboration in my research.

In addition, I would like to thank my Chinese friends at Vanderbilt, especially Weihan Wang, Jiani Li, Xuyi Luo, Huahong Zhang, Yasi Wang, Yongtai Liu, Tengyu Ma, Xiaojia Zhao, Pengfei Wang, Xinle Hong, Ruohan Wang, and Xiao Li, for making life in a foreign country enjoyable without being lonely any more.

Last but not least, no words can express my gratitude to my mother Yan Yang and my girlfriend Shanshan Hu. Their unconditional love and unwavering support are my greatest treasure. Finally, special thanks go to my brother-like friends, Chengyan Liu, He Zhang, and Yuyang Zheng, for always being there with me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

Introduction

1.1 Motivation

With the machine learning techniques, such as Deep Neural Networks (DNNs), demonstrating remarkable achievements in a wide variety of domains, it is no surprise that they have been increasingly used in Cyber-Physical Systems (CPSs) to perform different complex tasks that cannot be easily solved by conventional techniques, such as perception, planning, and control [1; 2; 3]. Even if such Learning-Enabled Components (LECs) are beneficial to increase the autonomy of the system, the safety and reliability of LECs should be analyzed and ensured before deploying them to real-world systems, especially safety-critical systems [4; 5].

Unfortunately, the characteristics and complexity of the LECs can impede the analysis. LECs encode knowledge in a form that is not transparent. DNNs, for example, capture features in a multitude of activation functions that cannot be used inspected to ensure that the LEC operates as intended. High levels of autonomy require high-capacity models that further obscure the system operation. Even if an LEC is trained and tested extensively, it is typically characterized by a nonzero error rate. More importantly, supervised and reinforcement learning, which are typical learning techniques for training LECs, are built upon an underlying assumption that the training and test data are sampled from the same distribution. Nevertheless, the training dataset is necessarily incomplete, and the Out-Of-Distribution (OOD) data are inevitably present when the LECs are deployed to the real-world system. Furthermore, it is well known that LECs, such as DNNs, are vulnerable to examples with small deliberately human-crafted perturbations in the input [6]. The OOD or adversarial examples may cause the LEC to be ineffective and predict the results with large errors, and further compromise the safety of the overall system. As such, it is significantly paramount to develop approaches for detecting OOD examples, ensuring the safe and reliable operation of CPS. Although detection of OOD examples in DNNs has received considerable attention [7; 8; 9], most of them do not take into consideration the dynamical behavior of CPS. It is not applicable to apply them to CPS in a straightforward manner since they will trigger a large number of false positives and hinder the normal operation of the system. The detection method considering the dynamical behavior of the CPS still needs to be explored.

Moreover, it is far from sufficient only to employ point-wise detection. Recent work demonstrates that CPSs are susceptible to specific cyber attacks on sensor measurements or control signals, such as sensor replay and controller integrity attacks [10; 11]. Such attacks can easily bypass the point-wise detection methods because all the data points are from the same distribution as the training dataset and will not raise any alert. However, the time-series sequence significantly deviates from the normal sequential behavior and is out of the distribution of normal sequences. As a result, detection of such attacks should consider the time-series sequences.

1.2 System Model and Research Challenges

CPSs use LECs extensively to perform various tasks in order to increase the level of autonomy. A typical simplified CPS architecture with LECs (e.g., DNNs) for perception and control is shown in Figure 1.1. A perception component observes and interprets the environment and provides information to a controller which, possibly using additional sensors (feedback from the plant), applies an action to the plant in order to achieve some task. In response to this action, the state of the physical plant changes, and the environment must be observed and interpreted again in order to continue the system operation. An end-to-end control architecture from perception to actuation can also be used.

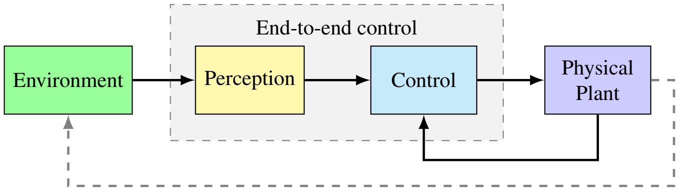


Figure 1.1: Simplified CPS control architecture.

An LEC is designed using learning methods such as supervised and reinforcement learning. We assume that the LECs are successfully trained, and the training and test errors are satisfactory. However, the training and test datasets at design time are necessarily incomplete and may under-represent safety-critical cases. OOD examples may lead LEC to predict outputs with large errors and compromise safety. Therefore, detection of OOD examples is crucial, for example, in order to enable decision-making by switching to a different control architecture or human supervision. During the system operation, the inputs arrive one by one to the LEC. After receiving each input, the objective is to compute a valid measure of the degree to which the assumption that the input example is generated from the same probability distribution as the training data is falsified. However, several challenges make this objective difficult to achieve:

- **Large number of false alarms.** Although detection of OOD examples in neural networks has received considerable attention, most of them are based on single input examples [12; 13]. Such detection techniques do not take into consideration the dynamical behavior of CPS and can exhibit a large number of false alarms, and cannot be applied to CPS in a straightforward manner. Online detection algorithms must be robust with a small number of false alarms.

- **High-dimensional input data.** For learning-enabled CPS, OOD detection must be performed in real-time, which is very challenging because inputs to perception and end-to-end control LECs are high-dimensional measurements from sensors such as cameras, LIDAR, and RADAR. The time and memory requirements must be similar to the requirements of the LECs used in the CPS architecture.

- **Long-term dependencies.** The components in CPS are generally interconnected through networks, which expose surfaces to adversaries to launch cyber attacks. Detection of some specific cyber attacks, such as replay attacks on sensor measurements, must consider the time-series sequences because point-wise detection methods will be easily bypassed by nuanced attacks. The long-term dependencies of series data, especially for the high-dimensional sequence, impose immense obstacles to detecting such attacks.

1.3 Research Contributions

Towards addressing the outlined challenge of detecting OOD examples in learning-enabled CPS in real-time, several detection approaches are proposed in this dissertation, and extensive evaluations are conducted to demonstrate their effectiveness. The primary contributions are listed below.

Chapter 3:

In this chapter, we propose an approach for the detection of OOD inputs in learning-enabled CPS. The main contributions are:

- We propose a real-time OOD detection method that leverages inductive conformal prediction and anomaly detection. In order to handle high-dimensional inputs in real-time, we propose to compute the nonconformity scores using learned models based on Variational AutoEncoders (VAEs) and deep Support Vector Data Description (deep SVDD). The VAE- and SVDD-based methods allow the efficient computation of the nonconformity score and the real-time detection of OOD high-dimensional inputs. The robustness of the detection can be improved considerably by taking into account multiple input examples and comparing them with the calibration nonconformity scores.

- We propose a method for improving OOD detection by incorporating saliency maps. The saliency map is computed to quantify how much the input features of the image contribute to the LEC output and then is used to weight the contribution of the input features to the nonconformity scores. Therefore, the detection algorithm weights the input features based on their influence on the output of the LEC. The main benefit of this method is to decrease the impact of nonconformal input features that do not contribute to the LEC prediction.

- We evaluate our approach using (1) an Advanced Emergency Braking System (AEBS), (2) a Self-Driving End-to-end Controller (SDEC), and (3) an Autonomous Vehicle Seasonal Dataset (AVSD). For all cases, the evaluation results show that the proposed approach has a very small number of false alarms. In addition, the execution time of the detection method is shorter than the execution time of the original LECs, which demonstrates that the method can be used in real-time.

Chapter 4:

In this chapter, we extend the approach in Chapter 3 and employ a VAE for classification and regression model for detecting different OOD data in learning-enabled CPS. The main contributions are:

- We first discuss the causes of the OOD examples and then categorize them into four different types. We provide formal definitions and some typical examples for these four different types of OOD data present in learning-enabled CPS.

- We propose an approach for detecting a variety of OOD data in learning-enabled CPS. In order to take into consideration outputs of the LEC, the proposed approach employs a VAE for classification and regression model, which is learned jointly by combining the VAE and classifier and regressor and conditioning the latent representation of the VAE on the target variable of the classifier and regressor.

- We demonstrate the approach using several datasets for classification and regression tasks. The evaluation results demonstrate that the proposed approach can detect different types of OOD data with a very small number of false alarms. The execution time is comparable with the sampling period of the typical CPSs, which enables real-time detection.

Chapter 5:

In this chapter, we follow the definitions for different types of OOD data in Chapter 4 and only focus on the OOD data in LECs used for classification. We develop OOD detection algorithm based on Adversarial AutoEncoder (AAE) and make the following contributions:

- We introduce a novel approach for detecting different types of OOD data in LECs that are used for classification problems. By utilizing a variant of an AAE, the joint distribution of the input and output variables on the training dataset can be represented. As such, both the input and output of the LEC can be taken into consideration for OOD detection.

- We conduct extensive experiments on several datasets to evaluate the proposed approach. The results show that the proposed method has a better detection performance than the method introduced in Chapter 4. Besides, such a method is computationally efficient and can be used for online detection.

Chapter 6:

In this chapter, we move our attention from point-wise detection to sequence-wise detection and develop an approach for detecting sensor replay and controller integrity attacks in CPSs. We make the following contributions:

- We propose a generative model used for detecting anomalies of high-dimensional time-series data. The model consists of a VAE and a Recurrent Neural Network (RNN) that is used to learn both spatial and temporal features of the normal dynamical behavior of the system. Such a model has the capacity of predicting future, and we can compare the predicted observation with the actual observation to quantify the nonconformity of actual behavior relative to normal behavior.

- We develop an approach for real-time detection of sensor replay and controller integrity attacks in CPS. We propose to recursively utilize the RNN to predict the observations for multiple time steps in the future. By comparing the expected current observations predicted from multiple steps in the past with the current actual observation, a series of nonconformity scores can be efficiently computed. The nonconformity scores are then combined with ICAD allowing detection of abnormal behavior in a long sequence.

- We provide comprehensive evaluations for the proposed approach using two case studies: (1) an AEBS and (1) an autonomous car car racing example. The evaluation validates the effectiveness of our approach for detecting sensor replay attack and controller integrity attack using high-dimensional sensor observations. The execution time of the detection method is much shorter than the sampling period of the system, which demonstrates proposed method can be used for real-time detection.

1.4 Organization

The remainder of this dissertation will be organized as follows:

- Chapter 2 reviews the respective related work in the research of detection of OOD detection in learning-enabled CPS.

- Chapter 3 proposes an approach for detecting OOD examples in learning-enabled CPS, which is the foundational framework for the subsequent chapters.

- Chapter 4 studies the problem of detecting different types of OOD data in learning-enabled CPS and introduces a detection method using VAE for classification and regression.

- Chapter 5 focuses on different types of OOD data in learning-enabled components used for classification and develops a detection approach using Adversarial AutoEncoder (AAE).

- Chapter 6 brings the attention to anomalous behavior in time-series data and proposes an approach for detecting sensor replay attack and controller integrity attack in CPS.
- Chapter 7 concludes the dissertation.

CHAPTER 2

Related Work

Learning-enabled components (LECs) in Cyber-Physical Systems (CPSs) may become ineffective because of the Out-Of-Distribution (OOD) data, which may compromise the safety of the overall system. Therefore, it is essential to detect OOD data and raise alarms to indicate that LECs may give predictions with large errors. In this chapter, we review the related work in the research of OOD detection in learning-enabled CPSs. We begin with a survey over safety assurance in learning-enabled CPSs in Section 2.1. Then, Section 2.2 discusses the security of CPS. Next, in Section 2.3, we give a state-of-the-art review on anomaly detection, with a special focus on the methods using deep neural networks. In Section 2.4, we discuss the anomaly detection specifically in the context of neural networks. We briefly review the related work about novelty detection for unknown classes and anomaly detection for reinforcement learning in Section 2.5 and Section 2.6, respectively. Finally, we discuss the related work about neural network interpretability in Section 2.7. Figure 2.1 summarizes the related work discussed in this chapter.

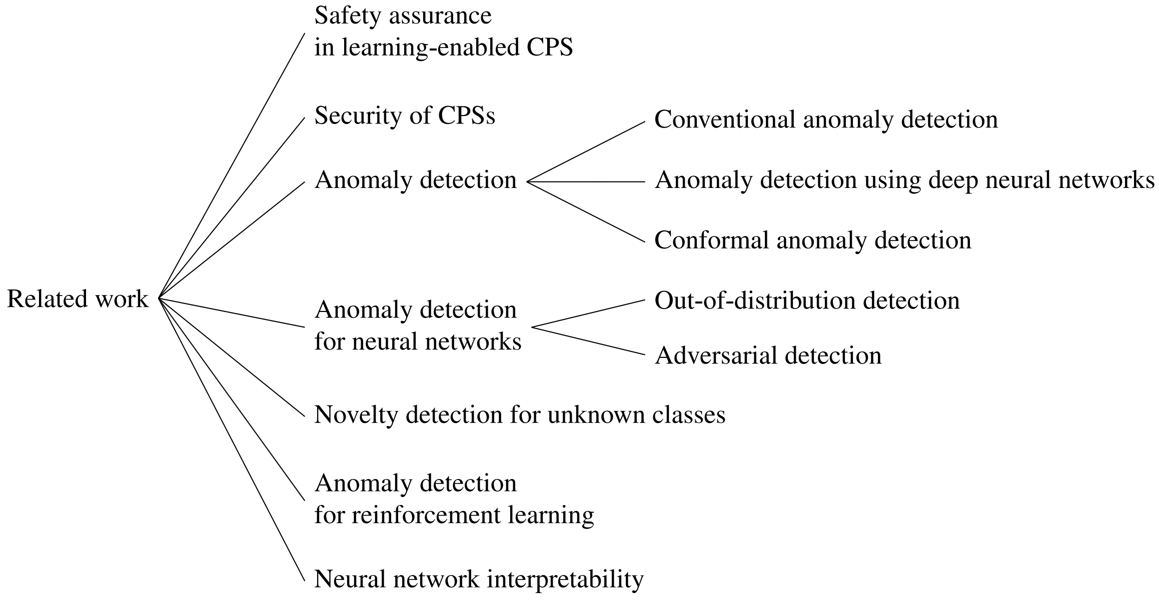

Related work
Safety assurance in learning-enabled CPS
Security of CPSs
Anomaly detection
Conventional anomaly detection
Anomaly detection using deep neural networks
Conformal anomaly detection
Anomaly detection for neural networks
Out-of-distribution detection
Adversarial detection
Novelty detection for unknown classes
Anomaly detection for reinforcement learning
Neural network interpretability

Figure 2.1: Summary of related work.

2.1 Safety Assurance in Learning-enabled Cyber-physical Systems

Verification and assurance of CPS with machine learning components is considered in [14] in a broader context of verified artificial intelligence. The challenges discussed in [14] include the integration of design-time and runtime methods to address the undecidability of verification in complex systems and environment modeling. OOD detection can be used with recovery and reconfiguration techniques to complement design-time verification. Focusing on design-time techniques, a Feature Space Partitioning Tree (FSPT) is used to split the input feature space into multiple partitions and to identify those partitions where the training samples are insufficient in [15]. The method can be used for testing whether a data is OOD which is equivalent to checking whether this input instance comes from these data-lacking feature space partitions. Compositional falsification of CPS with machine learning components is introduced in [5] and demonstrated with a simulated advanced emergency braking system. The approach is applied at design-time for identifying executions that falsify temporal logic specifications and also identifies regions of uncertainty where additional analysis and runtime monitoring is required. A related approach for simulation-based adversarial test generation for autonomous vehicles with machine learning components is presented in [4]. The technique is also used at design-time to increase the reliability of autonomous CPS and can provide additional training data for OOD detection.

2.2 Security of Cyber-physical Systems

Attacks on CPS can be summarized as attacks on the actuators, physical plant, and communication networks [16]. The attacks on the networks can be divided into two subcategories: (1) Denial of Service (DoS) attacks, which prevent the sensor readings (or control signals) from being received by the controller (or actuator); (2) deception attacks, or integrity attacks, which use malicious information to modify the original normal information in networks. Integrity attacks on the sensor and control signals are investigated in [17] and [18]. In [10], it is shown that an integrity attack on the control signal could possibly lead a car to a crash. In [11], a distributed DoS attack targets Vehicular Ad hoc NETworks (VANET) and causes severe congestion on a given road segment. To mitigate these threats, a significant amount of work has been proposed for attack prevention and detection. A robust control system against DoS attacks is proposed in [19]. Using an additional authentication control signal, a method for detecting replay attacks is presented in [20; 21]. A Chi-square detector and Fuzzy logic based classifier are used to detect and identify distributed DoS attacks in CPS [22]. Neural networks have been used before to detect attacks on CPS. In [23] and [24], neural network based detectors are for replay and integrity attacks in power systems. These detection methods are limited to low-dimensional data.

2.3 Anomaly Detection

Anomaly detection still has been an important research topic due to its widespread use in various applications. In this section, we will survey over the related work on anomaly detection. We firstly discuss the conventional anomaly detection methods that do not use deep neural networks. Then, we review the related work on deep anomaly detection methods that are the anomaly detection approaches using deep neural networks. Finally, we focus on a specific anomaly detection framework – conformal anomaly detection and discuss the related work on it.

2.3.1 Conventional Anomaly Detection

Several surveys trying to structure and categorize the existing anomaly detection algorithms have been published last few years [25; 26; 27]. In this subsection, we focus on the conventional anomaly detection algorithms which are not using deep neural networks and categorize them into following classes: statistical-based approaches, neighbor-based approaches, clustering-based approaches, and other approaches not belonging to above-mentioned classes.

Statistical-based approaches

The underlying assumption of statistical-based anomaly detection methods is *"Normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model"* [25]. The normal data points are modeled using a stochastic distribution, and therefore, the anomaly detection can be applied by testing if the unseen data point belongs to this stochastic distribution. According to whether the parameters of the distribution model is estimated, the statistical-based methods are split into two main groups: the parametric and non-parametric methods. The formal group assumes the knowledge of underlying distribution and estimates the parameters of the distribution, whereas the latter group has no assumption of prior knowledge of the distribution model [25].

Various stochastic distributions can be used for the parametric model in anomaly detection, such as Gaussian distribution [25] and Poisson distribution [28]. Gaussian mixture model (GMM) [29] is a typical parametric model, which assumes the data are generated from a given number of Gaussian distributions. The parameters can be estimated using expectation-maximization (EM) [30]. The low estimated probability density at the test point indicates the anomaly. In [31], GMM is applied to identify the cancerous masses in mammograms. The regression model is another straightforward parametric approach to anomaly detection problems. This method tries to fit the regression model to the data, and the residual of the test data is used to define the anomaly score [32].

An apparent drawback of the parametric method is that the true structure model of the probability distribution may not exist, or it is too complex to estimate. Non-parametric methods which have no assumption of prior knowledge of the distribution can overcome this drawback. Kernel density estimation (KDE) [33], also known as Parzen window estimation, is a common non-parametric method for anomaly detection. This method approximates the probability distribution function (pdf) of the unknown distribution by placing a kernel function to each data point then summing up the local contributions of the kernels. The test point that lies in the low probability area of the pdf is classified to be an anomaly.

Neighbor-based approaches

Neighbor-based approaches are based on the assumption that *"Normal data instances occur in dense neighborhoods, while anomalies occur far from their closest neighbors"* [25]. *k*-nearest neighbors method is a straightforward neighbor-based approach. The basic idea of *k*-nearest neighbors method is defining the anomaly score of the test data point as the distance to its *k*th nearest neighbor in a given data set [25]. Also, some variants are raised, and the anomaly score can be calculated as the sum of the distances to the *k* nearest neighbors [34], or as the inverse of the number of nearest neighbors whose distances are smaller than a threshold *d* [35]. Also, different distance metrics can be applied to compute the similarity between two data points. The Euclidean distance is normally used for continuous attributes [36]. Hausdorff distance [37] is a popular distance metric for trajectory data. Local outlier factor (LOF) [38] is another representative neighbor-based approach. The core of this method is the computation of the LOF score of a data instance *p* which can be computed by the ratio of average local density of the *k* nearest neighbors to the local density of *p*. Hence, the higher LOF score indicates an anomalous instance.

Clustering-based approaches

The key idea for clustering-based approach is to use the clustering techniques to group similar data points into clusters. K-means clustering algorithm [39] aims to partition all data instances into *k* clusters in which each data instance belongs to the cluster with the nearest center. The anomalies can be considered as the points that are not within or nearby any dense clusters. Isolation forest [40] recursively generate trees isolating any data from the rest of the data by randomly selecting an attribute and then selecting a split value for this attribute. The isolation score of a point is the average path length from the root of the tree to the node containing the data instance. A smaller isolation score, or short path length, denotes an anomalous instance since it is easy to separate from other nominal points.

Other approaches

One-class support vector machine (OC-SVM) [41] is an extended application of SVM algorithm to anomaly detection problem. By performing the kernel function, the input space is mapped into a high dimensional space. The approach tries to separate the normal and anomalous instances by finding a hyperplane in the high dimensional space with a maximal distance from the origin. Similar to OC-SVM, support vector data description (SVDD) [42] tries to learn a smallest hypersphere that contains all normal instances, and the anomalies lies outside the hypersphere.

2.3.2 Anomaly Detection using Deep Neural Networks

A fundamental problem in conventional anomaly detection is to expressively represent large-scale data such as high-dimensional and temporal data. Deep learning has shown the tremendous potential in dealing with such complex data, and therefore, several anomaly detection approaches based on deep learning have been proposed in recent years and demonstrate better performance in detecting the anomalies. In this subsection, we will make a review of the anomaly detection methods using deep neural networks, also known as deep anomaly detection.

The deep learning techniques are used for feature extraction only and independent from the anomaly score computation in some work [43; 44]. The feature extraction, or dimensionality reduction, aims to extract low-dimensional feature representations from high-dimensional data. Deep learning-based dimension reduction techniques have demonstrated better capability in extracting semantic-rich features and non-linear feature relations [45] compared with the methods that are popular in anomaly detection, such as principal component analysis (PCA) [46], and random projection [47]. In [43], autoencoder networks are trained to learn low-dimensional feature representations, and subsequently one-class SVMs are applied to detect the anomalies on these learned low-dimensional representations. Similar to [43], [44] utilizes the deep belief networks (DBNs) for dimension reduction for high-dimensional tabular data. In addition, some work directly use the pretrained deep learning models to extract the low-dimensional features, such as VGG [48] and ResNet [49].

Several existing neural network architectures, such as autoencoders and generative adversarial networks, are utilized in deep anomaly detection algorithms. Although such architectures are not primarily designed for anomaly detection, the learned representations can still empower the anomaly detection since key regularities of the normal instances are captured by these neural networks during training [45].

Autoencoder is a commonly-used deep learning architecture for anomaly detection. An autoencoder consists of an encoder and a decoder. The encoder maps the original input onto a low-dimensional feature representation, while the decoder tries to reconstruct the data from the low-dimensional space. In other words, an autoencoder aims to learn a low-dimensional space on which the given data instances can be well represented and reconstructed. The underlying assumption of the autoencoder-based anomaly detection methods is ”normal data instances can be better reconstructed from compressed feature space than anomalies” [45]. Therefore, the reconstruction error between the input and reconstructed output is generally used as the anomaly score.

The first attempt by using autoencoders for anomaly detection is presented in [50]. In [50], an autoencoder, also referenced by the name replicator neural network, is built upon on a multi-layer perceptron with three hidden layers. The replicator neural network, trained solely on the normal data instances, fails to reconstruct the anomalous data and generates an output with large reconstruction error. In addition, other variants of autoencoder can also be used for detecting anomalies, such as convolutional neural network autoencoder (CNN-AE) [51], variational autoencoder (VAE) [52], and long short-term memory autoencoder (LSTM-AE) [53; 54].

Generative adversarial networks (GANs) have demonstrated outstanding capacity in generating realistic image data. Therefore, GANs are introduced to anomaly detection domain trying to achieve better performance in more realistic and complex dataset. A GAN consists of two adversarial modules: a generator G and a discriminator D. The generator G learns to map from a latent space to the data distribution space, while the discriminator D maps the input to a single scalar value that can be interpreted as the probability that the given input is a real input sampled from training data or a fake input generated by G. The generator and discriminator are simultaneously optimized through a two-player minimax game whose objective is to make generator generates more realistic inputs and make discriminator identifies real and generated inputs with lower error rate. GAN-based anomaly detection methods are based on the following assumption: ”normal data instances can be better generated than anomalies from the latent feature space of the generative network in GANs” [45].

One of the early work using GAN for anomaly detection is AnoGAN [55]. AnoGAN utilizes the standard GAN architecture, and the key idea of AnoGAN is that, for a given input x, it tries to search a representation z_γ in the latent feature space so that the corresponding generated instance $G(z_\gamma)$ and input x are as similar as possible. Since the generator is trained to capture the variability of the normal data, the anomalies are expected to have less similar generated outputs than normal instances. The anomaly score is defined as

$$A(x) = (1-\lambda)\cdot R(x) + \lambda \cdot D(x)$$

where λ is a hyperparameter, $R(\cdot)$ is the residual score, and $D(\cdot)$ is the discrimination score. $R(\cdot)$ is defined as $\sum |x - G(z_\gamma)|$, which measures dissimilarity between the input x and the generated output $G(z_\gamma)$. $D(\cdot)$ is defined as $\sum |f(x) - f(G(z_\gamma))|$, where $f(\cdot)$ is the intermediate feature representation of the discriminator. The large anomaly score indicates the anomalous instances, whereas a small score means a very similar instance was already seen during the training. For every single input x, the searching process for z_γ is required, and therefore, AnoGAN has the issue of computational inefficiency.

To solve the disadvantage of AnoGAN, efficient GAN-based anomaly detection (EGBAD) and Fast AnoGAN (f-AnoGAN) are proposed by [56] and [57] respectively. Different from the AnoGAN, EGBAD and f-AnoGAN encodes the test input x to a latent representation z during testing, which avoids the optimization process searching for z_γ and reduces the computational time. The difference between the EGBAD and f-

AnoGAN is the way to train the encoder: during training, EGBAD simultaneously learns an encoder E along with a generator G and discriminator D, whereas f-AnoGAN learns a standard GAN and an encoder successively. GANomaly [58] is another GAN-based anomaly detection method extending the idea of EGBAD and f-AnoGAN. GANomaly changes the generator network in standard GAN to an encoder-decoder-encoder network which can be represented as: $x \xrightarrow{G_E} z \xrightarrow{G_D} \hat{x} \xrightarrow{E} \hat{z}$ [45]. Besides, GANomaly defines a new anomaly score as the l_1 distance between the encoded features of original input x and the encoded features of generated image $\hat{x}$ expressed as

$$A(x) = ||G_E(x) - E(G(x))||_1,$$

where G is a composition of the encoder G_E and the decoder G_D. [58] reports GANomaly achieving better performance compared with VAE, AnoGAN and EGBAD.

Targeting at the temporal data set, such as image frames in a video sequence, predictability modeling-based methods, which predict the current data instances using the information from historical instances, can be applied for anomaly detection [45]. The data instances in a sequence are time-correlated, and the normal instances are normally adherent to such dependence and can be well predicted, whereas the anomalous instances are hard to be predicted. An anomaly detection method in videos based on this idea is proposed by [59]. Given a sequence of video frames $x_1, x_2, \ldots, x_t$, a deep neural network is trained to predict a future frame $\hat{x}_{t+1}$ as close to the ground truth frame x_{t+1} as possible using all these previous frames. At evaluation stage, the prediction error between x_{t+1} and $\hat{x}_{t+1}$ is used as the anomaly score.

Aforementioned deep anomaly detection algorithms learn the representations of the data set by optimizing loss functions that are not designed for anomaly detection. There are some deep anomaly detection methods whose architectures and optimization objects are specifically designed for anomaly detection problem.

Deep one-class classification-based methods are typical deep anomaly detection algorithms that are specifically designed for anomaly detection. As discussed in Section 2.3.1, one-class support vector machine (OC-SVM) and support vector data description (SVDD) are two quintessential one-classification-based anomaly detection methods which learn a boundary in a high-dimensional space separating the normal instances from the rest of the high-dimensional space, and thus, out-of-boundary instances are clustered as anomalies. However, these conventional methods based on SVM struggle with high-dimensional data. One class neural network (OC-NN) and deep support vector data description (Deep SVDD) are the extensions combining deep neural network with OC-SVM and SVDD, respectively. OC-NN can be regarded as designing a neural network architecture using an OC-SVM equivalent loss function that tries to optimize a hyperplane to separate the normal instances from the origin in the representation space. Similar to OC-NN, the idea of deep SVDD is to train a neural network to map the input data into a hypersphere of minimum volume characterized by center $\boldsymbol{c}$ and radius R. Benefiting by the neural network, OC-NN and deep SVDD outperform conventional methods on complex data set.

In addition, reinforcement learning has also attracted significant interest due to its superior performance. The first attempt trying to apply reinforcement learning to anomaly detection is raised in [60]. The proposed approach adapts the reinforcement learning method by casting the problem of time series anomaly detection as a Markov decision process (MDP) and redefining the optimization object of the reinforcement learning as maximizing the performance of the detector. However, the labeled anomalous instances are required in this method. Also, inverse reinforcement learning (IRL) is employed for sequential anomaly detection in a recent study [61]. The proposed method takes a set of normal trajectories as input, and the normal behavior can be understood by the reward function inferred by IRL. A low reward value assigned by the reward function indicates an anomalous observation.

2.3.3 Conformal Anomaly Detection

Most anomaly detection algorithms define the anomaly threshold in terms of the distance, density, or data likelihood, which are typically not normalized, and it is hard to tune to regulate the balance between the sensitivity and the rate of false alarms of the detector. *Conformal anomaly detection* (CAD) is proposed in [37] based on *conformal prediction* (CP) [62]. The main idea of these methods is to test if a new input example conforms to the training data set by utilizing a *nonconformity measure* which assigns a numerical score indicating how different the input example is from the training data set. The next step is to define a p-value as the fraction of observations that have nonconformity scores greater than or equal to the nonconformity scores of the training examples. The p-value is then used for estimating the confidence of the prediction for the test input. In order to use the approach online, *inductive conformal anomaly detection* (ICAD) is introduced in [63], where the original training set is split into a proper training set and a calibration set, and the p-values are computed relative to calibration examples. If a p-value is smaller than a predefined anomaly threshold ε, the test example can be classified as an anomaly. An important property of the approach is that *the rate of detected conformal anomalies is well calibrated*, that is with very high probability it is less or approximately equal to a predefined threshold $\varepsilon \in (0,1)$ [63].

Several anomaly detection approaches based on the CAD and ICAD are raised, but the nonconformity measures are defined differently. Kernel density estimation (KDE) [64] and k-nearest neighbor [65] nonconformity measures are used for anomaly detection of the single point. For sequential anomaly detection of time trajectories, the sum of Hausdorff distances to k nearest neighbors, the average of Mahalanobis distances to the k nearest neighbors, and the sub-sequence local outlier factor are employed as the nonconformity measure in [37; 66], [67], and [63], separately. However, existing nonconformity measures cannot scale to the high-dimensional inputs.

Later, conformal prediction and conformal anomaly detection are extended for testing exchangeability of data [68; 69] and detecting change-point [70; 64]. Exchangeability testing and change-point detection can be performed by testing the hypothesis that p-values are independent and uniformly distributed. Such hypothesis can be tested by using martingales that are constructed using the p-values [69]: a martingale will grow only if the p-values are not independent and uniformly distributed. Several different martingales are raised and used in the literature, such as power martingale [68], simple mixture martingale [68] and plug-in martingale [69].

2.4 Anomaly Detection for Neural Networks

Although neural networks generalize well if the training and testing data are sampled from the same distribution, OOD data may still lead to incorrect outputs or outputs with large errors. In addition, recent studies show that the well trained neural networks are vulnerable to examples with small specially human-crafted perturbations in the input that cause large-error predictions [6; 71]. Therefore, anomaly detection becomes more and more important to make the neural network more robust and reliable. Although some of the anomaly detection approaches discussed in 2.3 can be adopted for anomaly detection for neural networks, they are not specially designed in the context of deep neural networks. As the categories in [72], we will survey the anomaly detection for neural networks over two paradigms: OOD examples (unintentional) detection and adversarial examples (intentional) detection.

2.4.1 Out-of-distribution Detection for Neural Networks

In some of the existing work, detectors which only take the inputs into consideration and do not use any internals of the monitored neural network is designed to detect OOD examples. Several anomaly detection approaches discussed in Section 2.3 which are not specifically in context of neural networks can be adopted for OOD detection for neural networks, such as VAE-based [13] and GAN-based [73] methods. Several new approaches for detecting OOD examples are raised in the context of neural networks. Based on autoencoder-based method, several work are proposed to improve the performance of detecting anomalies in deep learning systems. An approach using k-NN distance in the latent space of the autoencoder is proposed in [74] to improve detection performance over using only the reconstruction error. However, it is required to iterate over the training set during testing, which is not computational efficiency and not practical for real-world applications. In [74], the Mahalanobis distance is employed in the latent space between the test sample and the mean vector of the training set with the reconstruction loss of the test sample to construct an anomaly score.

In [75], a multi-class model is trained to discriminate between geometric transformations applied to the input image, and such transformations encourage the classifier to learn some geometric features that are useful for anomaly detection. The classifier can be trained over a self-labeled data set created by applying various different geometric transformations to the original training data. At inference time, the authors apply each transformation on the test image, and each of the transformed images is fed to the classifier. The anomaly score is defined as the combined log-likelihood of a transformed image conditioned on each of the applied transformations [75] and can be used to detect OOD instance. The information from the monitored neural network, such as predictions and features in the hidden layers, can be used for detecting the OOD examples. A baseline method to detect OOD examples in neural network for classification is proposed in [7]. The method is based on the idea that a well-trained neural network tends to assign higher softmax scores to in-distribution examples than OOD examples. Thus, an example with a lower predicted softmax score than a threshold is classified as an OOD instance. However, the softmax score gap between in- and out-of-distribution examples are not large enough and this technique fails to classify some of the in- and OOD examples. Hence, the authors in [8] go further and propose ODIN (OOD detector for neural networks) by using temperature scaling in the softmax function and adding small perturbations to inputs, which enlarge the softmax score gap between in- and OOD examples, thus lead a better detection performance. Despite the fact that the ODIN shows significant improvements on detection performance compared with the baseline, it needs OOD data to fine-tune the parameters for temperature scaling and input preprocessing. Recently, two strategies are proposed in [9] for freeing ODIN from the needs of tuning with OOD data.

A method of learning confidence estimates for neural networks is proposed in [76], which adds an additional branch to yield a confidence logit. Also, the loss function should be adjusted to fit the new architecture training two branches simultaneously. The OOD detection can be performed by evaluating the learned confidence estimates: the input is marked as an OOD instance if the confidence estimate is less than a threshold.

A unified framework for detecting OOD and adversarial examples is presented in [77]. The basic idea is to measure the probability density of the test example on low-level feature spaces of neural networks. Specifically, a class-conditional Gaussian distribution is fitted to the pre-trained low-level features. Next, for a test example, a confidence score can be defined as the negative of Mahalanobis distance to the closest class-conditional Gaussian distribution. Besides, weighted averaging the confidence scores from different layers of the neural network can further improve the detection performance.

2.4.2 Adversarial Examples Detection in Neural Networks

It is well known that an adversary can craft an example that looks similar to a normal input but can mislead the neural network predict an erroneous output. Consequently, detecting such adversarial examples has received considerable critical attention.

A straightforward detection method is training a binary adversarial example classifier. In [78], the detector is trained on the intermediate feature representations of a pre-trained classifier (monitored neural network) over the original data set and adversarial examples. The results show that the method can detect the adversarial examples with high accuracy. In addition, while the detector is trained against a specific adversary, it generalizes well to similar and weaker adversaries.

PixelDefend is proposed in [79] to detect adversarial inputs. A generative model PixelCNN [80] is trained using the training dataset firstly, which defines the joint distribution over all pixels. Then, for a test image, the probability density is computed using the PixelCNN and ranked among the density values of all training examples. Such rank can be used as an indicator telling whether the input image is normal or under attack. Besides, PixelDefend is also able to purify the input image to correctly classify the image despite such adversarial modifications.

In [81], two features, density estimates and bayesian uncertainty estimates, are combined to detect adversarial examples. The former uses estimates from kernel density estimation of the training set in the feature space of the last hidden layer to detect adversarial examples. This method cannot detect the data that lie far from the data manifold. Thus, the latter method estimates the uncertainty of the prediction using a dropout neural network, and the uncertainty can be used to detect the adversarial examples. The paper reports that the combined detector can achieve $85-93\%$ detection accuracy on a number of standard classification tasks.

The authors in [82] expect the deep features in the neural network are more robust than the final classification results to adversarial examples and tries to analyze the activations of neurons in hidden layers to detect adversarial examples. The approach performs a k-NN similarity search among the deep features obtained from the training images to the given test image. Then, the score assigned by the k-NN classifier to the class predicted by the neural network can be used as the confidence of the classification. The results show that the method is able to filter out many adversarial examples. However, this method is not computational efficient for large real-world data set since it is required to store all the deep features of training images and iterate over all of them during testing.

I-defender is proposed in [83], which utilizes the intrinsic properties of the pre-trained neural network to detect adversarial examples. The basic idea of this method is that when a neural network is under attack and is misled to predict an erroneous output, the distribution of the hidden states is different from the distribution obtained by the normal data of the same class. Gaussian mixture model (GMM) is utilized to approximate the hidden state distribution of each class. For a test example, the likelihood of its hidden states can be computed and compared with the corresponding class threshold to tell whether the test example is adversarial or not.

In [84], the authors identify two common channels that are exploited by the adversarial examples: the provenance channel and the activation value distribution channel. The former implies that the slight changes in the activation values of neurons in a layer may lead to substantial changes of the activation status of neurons in the next layer, and eventually lead to misclassification. The latter means that while the provenance changes slightly, the activation values of the neurons may be significantly different from those in the presence of benign inputs. Thus, detecting adversarial examples can be performed by checking provenance and value invariant violations during neural network computation. This can be achieved by training a set of one-class classification models for individual layers only using activation values and status from benign examples. At test time, a test input is fed into neural network, and the one-class classification models tell if the test input induces the states that violate the invariant distribution during neural network computation.

2.5 Novelty Detection for Unknown Classes

Since the new class unseen during training appears in the test phase, the traditional classifier will wrongly recognize this unknown sample as one of the known classes. Open set recognition was proposed to overcome this limitation. The open set classifier should perform two tasks: novelty detection for unknown classes and classification for known classes. According to [85], the methods for open set recognition can be categorized into traditional methods and deep learning-based methods. In this section, we focus and provide a state-of-art review of the deep learning-based methods. SoftMax layer is used in deep learning-based classifier converting the outputs of penultimate layer into final probabilities of known classes. The SoftMax layer is a significant obstacle to adapt deep neural networks for open set recognition due to its closed-set nature [86]. Therefore, an OpenMax layer is proposed in [86] to extend the SoftMax layer to enable the model to predict the unknown class. The first step of this method is to use the scores from the penultimate layer to estimate how far the input is from the known training data. Then, such information is fed into OpenMax layer to redistribute probabilities from SoftMax and get the class probability of an unknown sample. Extending the OpenMax method, generative adversarial networks (GANs) are employed to synthesize images from unknown classes to train the neural network [87]. Besides, based on the similar idea of OpenMax, a method called deep open classification is presented in [88], which replaces the SoftMax layer with a 1-vs-rest layer of Sigmoids. A class conditioned autoencoder (C2AE) is trained for open-set recognition in [85]. The training process is divided to two sub-tasks: (1) the encoder firstly learns to perform classification for known classes, and (2) the decoder learns to reconstructed the input using the encoded representations conditioned by the class identity. The reconstruction error is used to identify the unknown class, and the decision boundary is computed by extreme value theory (EVT). Recently, conditional Gaussian distribution learning (CGDL) method is presented in [85]. The proposed method applied a probabilistic ladder network trying to learn conditional Gaussian distributions by forcing different latent features to approximate different Gaussian models. The reconstruction error and the probability of the test sample locating in the latent space are combined to detect the unknown class.

2.6 Out-of-distribution Detection for Reinforcement Learning

Although out-of-distribution detection in neural networks has received considerable attention, the out-of-distribution detection method for reinforcement learning is still an open problem. It was only recently that an uncertainty-based out-of-distribution detection approach for reinforcement learning is proposed in [89]. The approach focuses only on the Q-learning and proposes to use the uncertainty estimated by approximate Bayesian inference methods or ensembling techniques to detect out-of-distribution examples.

2.7 Neural Network Interpretability

Understanding how the neural networks make decisions is helpful to anomaly detection, and thus, we discuss the related work on neural network interpretability. Saliency maps are a visualization tool for identifying which parts of the input that contribute most to the LEC predictions and are used in [90; 91] to help to understand if the models focus on reasonable cues in an input image. The whole features in input images are treated equally in existing OOD detection methods . However, the saliency map indicates that not all the pixels contribute equally to the final prediction of the neural network. Therefore, OOD detection may be oversensitive and raise alarms when parts of the input that do not affect the LEC output are changed slightly.

Saliency maps can be computed using gradient-based methods [92] and deconvolution [93]. The former approach uses backpropagation to compute the partial derivatives with respect to the pixels in the input image and the latter uses deconvolution to map the feature activities in intermediate layers to the input pixel space. Guided-backpropagation [94] extends the deconvolution method by changing the backpropagation rule for input-gradients to produce cleaner saliency maps. Layer-wise relevance propagation (LRP) is introduced in [95] as a method to compute partial prediction contributions for input representations by propagating the prediction back until the input layer using the network weights and the neural activations created by the forward pass. A technique for generating class activation maps (CAM) using global average pooling (GAP) in CNNs is proposed in [96]. The class activation map for a particular category indicates the discriminative image regions used by the CNN to predict that category. Grad-CAM [97] is an extension of the CAM and uses the global-average-pooled gradients to weight the feature maps. VisualBackProp [98] is originally developed as a debugging tool and computes the saliency maps by backpropagating the information of the feature maps from deeper layers while simultaneously increasing the resolution.

CHAPTER 3

Real-time Out-of-distribution Detection [1]

3.1 Introduction

Learning-Enabled Components (LECs) such as neural networks are used in many classes of Cyber-Physical Systems (CPSs). Semi-autonomous and autonomous vehicles, in particular, are CPS where LECs can play a significant role in perception, planning, and control if they are complemented with methods for analyzing and ensuring safety [4; 5]. However, there are characteristics of LECs that can complicate safety analysis. LECs encode knowledge in a form that is not transparent. Deep Neural Networks (DNNs), for example, capture features in a multitude of activation functions that cannot be inspected to ensure that the LEC operates as intended. High levels of autonomy require high-capacity models that further obscure the system operation. Even if an LEC is trained and tested extensively, it is typically characterized by a nonzero error rate. More importantly, the error estimated at design time may be different than the true error because of Out-Of-Distribution (OOD) data.

Since training datasets are necessarily incomplete, safety assessment at design time is also incomplete. Design-time verification and analysis methods must be combined with runtime monitoring techniques that can be used for safety assurance. In real-world CPS, the uncertainty and variability of the environment may result in data that are not similar to the data used for training. Although models such as DNNs generalize well if the training and test data are sampled from the same distribution, OOD data may lead to large errors. An LEC is trained and tested using data available at design time but must be deployed in a real system and operate under possibly different conditions. Testing ensures that the error is satisfactory for a large number of examples. However, during the system operation, the LEC may still encounter OOD inputs. OOD detection must quantify how different are the new test data are from the training data and raise an alarm to indicate that the LEC may give a prediction with a large error. Detection methods must be robust and limit the number of false alarms while being computational efficient for real-time monitoring.

The proposed approach is based on Conformal Prediction (CP) [62; 99] and Conformal Anomaly Detection (CAD) [66]. The main idea of these methods is to test if a new input example conforms to the training dataset by utilizing a *NonConformity Measure (NCM)* which assigns a numerical score indicating how different the input example is from the training dataset. The next step is to define a p-value as the fraction of observations that have nonconformity scores greater than or equal to the nonconformity scores of the training

[1] This chapter is adapted with permission from [F. Cai and X. Koutsoukos. “Real-time out-of-distribution detection in learning-enabled cyber-physical systems,” in *11th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, April 2020.]

examples, which is then used for estimating the confidence of the prediction for the test input. In order to use the approach online, Inductive Conformal Anomaly Detection (ICAD) is introduced in [63], where the original training set is split into the proper training set and the calibration set, and the *p*-values are computed relative to calibration examples. If a *p*-value is smaller than a predefined anomaly threshold ε, the test example can be classified as an anomaly. The approach is used for sequential anomaly detection of time trajectories in [66] by using the nearest neighbors and Hausdorff distance measuring the nonconformity between trajectories. Combined with exchangeability martingales, ICAD is used for change-point detection in [70].

However, there are still challenges in applying such methods for real-time detection of high-dimensional inputs in CPS. Existing methods rely on NCMs computed using *k*-Nearest Neighbors (*k*-NN) and Kernel Density Estimation (KDE) and cannot scale to LECs with high-dimensional inputs. OOD detection using a single example is typically not robust and may result in a large number of false alarms that inhibit the CPS operation. Methods based on martingales that incorporate multiple examples are not applicable directly to CPS because the input sequence is time-correlated and not exchangeable.

The main contribution of the chapter is an approach for real-time detection of OOD inputs. The approach leverages inductive conformal prediction and anomaly detection. In order to handle high-dimensional inputs in real-time, Variational AutoEncoders (VAEs) [100] and deep Support Vector Data Description (deep SVDD) [101] are utilized for efficient computation of the nonconformity score, which enables the real-time detection of OOD high-dimensional inputs. In order to use multiple examples for detection, we apply different heuristic techniques for VAE- and SVDD-based methods. VAE is a generative model that allows generating multiple examples in real-time similar to the input and computing multiple *p*-values that increase the robustness of detection. Deep SVDD is a model trained to perform anomaly detection, and in our method, it is combined with a test based on a sliding window. It should be noted that the VAE and deep SVDD neural networks may exhibit an intrinsic error of computing nonconformity scores. However, the robustness of the detection is improved considerably by taking into account multiple input examples and comparing them with the calibration nonconformity scores.

Another contribution of the chapter is a method for improving the OOD detection by incorporating saliency maps. Saliency maps aim to identify parts of the input that contribute most to the LEC predictions [92; 93]. In our approach, a saliency map is computed to quantify how much the input features contribute to the LEC output and then is used to weight the contribution of the input features to the nonconformity scores. Therefore, the detection algorithm weights the input features based on their influence on the output of the LEC. The main benefit of this method is to decrease the impact of nonconformal input features that do not contribute to the LEC prediction. For high-dimensional inputs such as images, for example, it is possible that parts of the image do not affect the LEC output. As an illustrative example, the VAE may have difficulty generating fine-granularity details of the original input image, however, such fine-granularity details may not affect the LEC output. We integrate two algorithms for computing saliency maps into the approach: (1) Intergrated-Gradients Optimized Saliency (I-GOS) [102] and (2) VisualBackProp (VBP) [98]. The algorithms are very efficient and can be used in real time.

The final contribution is the empirical evaluation using (1) an Advanced Emergency Braking System (AEBS), (2) a Self-Driving End-to-end Controller (SDEC), and (3) an Autonomous Vehicle Seasonal Dataset (AVSD). The first two case studies are implemented in CARLA [103], an open-source simulator for self-driving cars. AVSD evaluates the approach using the Ford autonomous vehicle seasonal dataset [104]. The AEBS uses a perception LEC to detect the nearest front obstacle on the road and estimate the distance from the host vehicle based on camera images. The distance together with the velocity of the host car are used as inputs to a reinforcement learning controller whose objective is to comfortably stop the vehicle. OOD inputs are generated by varying a precipitation parameter provided by CARLA, which introduces visual effects that may cause large error in the distance estimation resulting in a collision. The simulation results demonstrate a very small number of false positives and a detection delay less than 1 s. For the SDEC that comes with CARLA [103], the empirical evaluation shows that the proposed method can be used to detect a class of physically realizable attacks in end-to-end autonomous driving presented in [105]. The attacks are realized by painted lines on the road to cause the self-driving car to follow a target path. The objective of the AVSD is to estimate the heading changes of a host vehicle from images captured by a camera. A neural network is trained in specific weather conditions and traffic scenarios but may encounter different ones. The evaluation results show that the proposed approach is effective using a real-world dataset. For all cases, the execution time of the detection method is comparable to the execution time of the original LECs, which demonstrates that the method can be used in real time.

3.2 Problem Formulation

CPSs are greatly benefited by using LECs that can handle the uncertainty and variability of the real world. A typical simplified CPS architecture with LECs (e.g., DNNs) for perception and control is shown in Figure 1.1. A perception component observes and interprets the environment and provides information to a controller, which, possibly using additional sensors (feedback from the plant), applies an action to the plant in order to achieve some task. In response to this action, the state of the physical plant changes and the environment must be observed and interpreted again to continue the system operation. An end-to-end control architecture from perception to actuation can also be used.

An LEC is designed using learning methods such as supervised and reinforcement learning. We assume that the LECs are successfully trained, and the training and test errors are satisfactory. However, the training and test datasets at design time are necessarily incomplete and may under-represent safety-critical cases. OOD inputs, in particular, which have not been used for training or testing, may lead to large errors and compromise safety.

The problem considered in this chapter is robustly detecting OOD inputs in real time. OOD detection is crucial, for example, in order to enable decision making by switching to a different control architecture or human supervision. During the system operation, the inputs arrive one by one to the perception LEC. After receiving each input, the objective is to compute a valid measure of the degree to which the assumption that the input example is generated from the same probability distribution as the training data is falsified.

Online detection algorithms must be robust with a small number of false alarms. The detection algorithms using a single example are typically not robust and may result in a large number of false alarms. The inputs in CPS are time-correlated, and therefore, they are not independent, which imposes a significant challenge to use multiple examples in detection.

Further, for learning-enabled CPS, OOD detection must be performed in real time, which is very challenging because inputs to perception and end-to-end control LECs are high-dimensional measurements from sensors such as cameras, LIDAR, and RADAR. The time and memory requirements must be similar to the requirements of the LECs used in the CPS architecture. Typical NCMs such as the k-Nearest Neighbor (k-NN) NCM [63] and the Kernel Density Estimation (KDE) NCM [64] cannot scale to high-dimensional inputs because they require either storing the training dataset or estimating the density in a high-dimensional space.

3.3 VAE-based Detection

Variational AutoEncoder (VAE) is a generative model that learns parameters of a probability distribution to represent the data [100]. A VAE consists of an encoder, a decoder, and a loss function. The objective is to model the relationship between the observation x and the low-dimensional latent variable z using the loss function $\mathcal{L}(\theta,\phi;x) = \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}[q_\phi(z|x)||p(z)]$, where θ and ϕ are neural network parameters. The first term in the loss function is the model fit, and the second is the KL divergence between the approximate posterior and the prior of z. A popular choice for the prior is the Gaussian distribution. VAE-based methods can utilize the reconstruction error or reconstruction accuracy for anomaly detection [52]. In this section, we introduce a VAE-based detection method based on Inductive Conformal Anomaly Detection (ICAD). The VAE allows generating multiple examples that are similar to the input. If these examples are not conformal to the training data, many of the corresponding nonconformity scores will be very large indicating the test example is out of the distribution of the training dataset.

3.3.1 Offline Training and Nonconformity Measure

Let us consider an LEC $y = f(x)$ defining a mapping from input x to output y. It should be noted that in this work only the inputs are taken into consideration for OOD detection. The set of input examples used for training is denoted by $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$. During the system operation, a sequence of inputs denoted by $(x'_1, \ldots, x'_t, \ldots)$ is processed one by one. The task of the OOD detection is to quantify how different the input sequence is from the training dataset. If the difference is large, the algorithm raises an alarm indicating that the LEC may generate an output y with a large error compared to the test error obtained at design time.

Since the observations of the environment are highly time-correlated, the collected training data are not exchangeable. As suggested in [62], reshuffling – a random permutation for the training dataset – is performed before training the VAE. After reshuffling, the training dataset is split into a proper training set $\mathcal{D}_{\text{proper}} = \{x_i\}_{i=1}^{m}$ and a calibration set $\mathcal{D}_{\text{calibration}} = \{x_i\}_{i=m+1}^{l}$. For each example in the calibration set, a function A is used to compute the NCM that assigns a numerical score indicating how different a test example is from the training dataset. The nonconformity scores of the calibration examples are sorted and stored in order to be used at runtime.

Given a new input x'_t, the nonconformity score α'_t is computed using the nonconformity function A relative to the proper training set, $\alpha'_t = A\left(\{x_i\}_{i=1}^{m}, x'_t\right)$. The computation requires evaluating the nonconformity (strangeness) of x'_t relative to $\{x_i\}_{i=1}^{m}$. The choice of the nonconformity function A must ensure computing informative nonconformity scores in real time. Using k-NN, for example, requires storing the training dataset, which is infeasible for high-dimensional inputs. Instead, we learn an appropriate neural network architecture that is trained offline using the proper training set and encodes the required information in its parameters. This neural network *monitors* the inputs to the perception or end-to-end control LEC and is used to compute the nonconformity scores in real time.

For a VAE, an in-distribution input x should be reconstructed with a relatively small reconstruction error. Conversely, an OOD input will likely have a larger error. The reconstruction error is a good indication of the strangeness of the input relative to the training set, and it is used as the NCM. We use the squared error between the input example x and generated output example $\hat{x}$ as the NCM defined as

$$\alpha = A_{\text{VAE}}(x, \hat{x}) = ||x - \hat{x}||^2. \tag{3.1}$$

During the offline phase, for each example $x_j : j \in \{m+1, \ldots, l\}$ in the calibration dataset, we sample a single reconstructed input $\hat{x}_j$ from the trained VAE and compute the nonconformity score α_j using Equation (3.1). The precomputed nonconformity scores of the calibration data are sorted and stored in order to be used at runtime. The steps that are performed offline are summarized in Algorithm 1.

Algorithm 1 VAE-based out-of-distribution detection

```
Input: a training set $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$; a test sequence $(x'_1, \ldots, x'_t, \ldots)$; number of calibration examples $l-m$;
    number of examples $N$ generated by the VAE; threshold $\tau$ and parameter $\omega$ of CUSUM detector
Output: boolean variable $Anom_t$
Offline:
Split the training set $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$ into the proper training set $\mathcal{D}_{\text{proper}} = \{x_i\}_{i=1}^{m}$ and calibration set
    $\mathcal{D}_{\text{calibration}} = \{x_i\}_{i=m+1}^{l}$
Train a VAE using the proper training set
for $j = m+1$ to $l$ do
    Generate $\hat{x}_j$ using the trained VAE
    $\alpha_j = A_{\text{VAE}}(x_j, \hat{x}_j)$
end for
Online:
$S_0 = 0$
for $t = 1, 2, \ldots$ do
    for $k = 1$ to $N$ do
        Sample $\hat{x}'_{t,k}$ using the trained VAE
        $\alpha'_{t,k} = A_{\text{VAE}}(x'_t, \hat{x}'_{t,k})$
        $p_{t,k} = \frac{|\{i=m+1,\ldots,l\} \mid \alpha_i \geq \alpha'_{t,k}|}{l-m}$
    end for
    $M_t = \int_0^1 \prod_{k=1}^{N} \varepsilon p_{t,k}^{\varepsilon-1} d\varepsilon$
    $S_t = \max(0, S_{t-1} + M_t - \omega)$
    $Anom_t \leftarrow S_t > \tau$
end for
```

3.3.2 Online Detection

Given a test input x'_t, the p-value p_t is computed as the fraction of calibration examples that have nonconformity scores greater than or equal to α'_t

$$p_t = \frac{|\{i = m+1, \ldots, l\} \mid \alpha_i \geq \alpha_t|}{l}. \tag{3.2}$$

It should be noted that the computation of the p-value can be performed efficiently online since it requires storing only the calibration nonconformity scores. Besides, since the nonconformity scores for the calibration data are sorted in the offline phase, the calculation of p-values can be accelerated by using a binary search. If $p_t < \varepsilon$, the example x'_t can be classified as an anomaly. Using a single p-value for detecting OOD examples can lead to an oversensitive detector with a large number of false alarms that inhibit the operation of the CPS. Our objective is to incorporate multiple examples into the detection algorithm and compute a set of p-values to test if there are many small p-values indicating an OOD input.

Given an input example x'_t at time t, the encoder portion of a VAE is used to approximate the posterior distribution of the latent space and sample N points $\{z'_{t,k}\}_{k=1}^{N}$ from the posterior that are used as input to the decoder portion in order to generate multiple new examples $\{\hat{x}'_{t,k}\}_{k=1}^{N}$. Typically, the posterior of the latent space is approximated by a Gaussian distribution. Sampling from the posterior generates encodings $\{z'_{t,k}\}_{k=1}^{N}$ so that the decoder is exposed to a range of variations of the input example and outputs $\{\hat{x}'_{t,k}\}_{k=1}^{N}$.

For each generated example $\hat{x}'_{t,k}$, $k \in \{1,\ldots,N\}$, the nonconformity score $\alpha'_{t,k}$ is computed as the reconstruction error between the test input x'_t and the generated example using Equation (3.1). Then, the p-value $p_{t,k}$ is computed as the fraction of calibration examples that have nonconformity scores greater than or equal to $\alpha'_{t,k}$ using Equation (3.2). It should be noted that, although the generated examples $\{\hat{x}'_{t,k}\}_{k=1}^{N}$ satisfy the exchangeability assumption, they are sampled from a Gaussian distribution conditioned by the input and are not sampled from the same distribution as the calibration dataset. Therefore, the N p-values $\{p_{t,k}\}_{k=1}^{N}$ are not independent and uniformly distributed in $[0,1]$. However, if the input x'_t is sampled from the same distribution as the training dataset, the p-values will be large, and they can be used for OOD detection.

At runtime, for every new input example x'_t received by the perception or end-to-end control LEC at time t, a *power martingale* [69] can be computed based on the sequence of p-values for some ε as

$$M_t^{\varepsilon} = \prod_{k=1}^{N} \varepsilon p_{t,k}^{\varepsilon-1},$$

and the *simple mixture martingale* [69] can be defined as

$$M_t = \int_0^1 M_t^{\varepsilon} d\varepsilon = \int_0^1 \prod_{k=1}^{N} \varepsilon p_{t,k}^{\varepsilon-1} d\varepsilon. \tag{3.3}$$

Both martingales will grow only if there are many small p-values in $\{p_{t,k}\}_{k=1}^{N}$, which will indicate an OOD input. If most of the p-values are relatively greater than 0, the martingales are not expected to grow. We use simple mixture martingale in our approach to avoid parameter tuning required for the power martingale.

In order to robustly detect when M_t becomes consistently large, we use the cumulative sum (CUSUM) procedure [106]. CUSUM is a nonparametric stateful test and can be used to generate alarms for OOD inputs by keeping track of the historical information of the martingale values. The detector is defined as $S_0 = 0$ and $S_t = \max(0, S_{t-1} + M_t - \omega)$, where ω prevents S_t from increasing consistently when the inputs are in the same distribution as the training data. An alarm is raised whenever S_t is greater than a threshold $S_t > \tau$, which can be optimized using empirical data [106]. Typically, after an alarm, the test is reset with $S_t = 0$.

Algorithm 1 describes the VAE-based real-time OOD detection. The NCM can be computed very efficiently by executing the learned VAE neural network and generating N new examples. The complexity is comparable to the complexity of the perception or end-to-end LEC that is executed in real time.

3.4 SVDD-based Detection

VAEs and other autoencoder architectures are trained to perform a task other than anomaly detection, assuming that the reconstruction accuracy will be better for in-distribution examples. Deep Support Vector Data Description (deep SVDD) is an architecture trained to perform anomaly detection [101]. The idea is to train a DNN to map the input data into a hypersphere of minimum volume characterized by center $\boldsymbol{c}$ and radius R. The input space $\mathcal{X}$ is transformed to a compressed output space $\mathcal{Z}$ while minimizing the volume of the hypersphere that encloses most of the input representations. Given a training dataset $\{x_i\}_{i=1}^{n}$, the *one-class deep SVDD* [101] is based on the loss

$$\min_{\mathcal{W}} \quad \frac{1}{n}\sum_{i=1}^{n}||\phi(x_i;\mathcal{W})-\boldsymbol{c}||^2+\frac{\lambda}{2}\sum_{\ell=1}^{L}||\boldsymbol{W}^{\ell}||_F^2,$$

where $\phi(\cdot;\mathcal{W}) : \mathcal{X} \to \mathcal{Z}$ denotes the neural network with L hidden layers and sets of weights $\mathcal{W} = \{\boldsymbol{W}^{\ell}\}_{\ell=1}^{L}$ is the center of the hypersphere, and the last term is a weight regularizer with hyperparameter $\lambda > 0$, where $||\cdot||_F$ is the Frobenius norm. One-class deep SVDD learns to map the data as close to center $\boldsymbol{c}$ as possible by penalizing the distance from representations to the center. The deep SVDD neural network must not have bias terms or bounded activation functions, and the center $\boldsymbol{c}$ can be selected as the mean of the representations from the initial inference on some training data to avoid trivial solutions that map the input space to a single point [101]. Given a new test example x, the distance of the representation $\phi(x;\mathcal{W}^*)$ to the center $\boldsymbol{c}$ of the hypersphere reflects how different the test example is from the training dataset and can be used as a NCM. In contrast to VAEs, deep SVDD is not a generative model and cannot be used to sample multiple examples. In order to use effectively the SVDD architecture in our approach, we use a sliding window containing a sequence of inputs as explained below.

3.4.1 Offline Training and Nonconformity Measure

The offline phase of the SVDD-based detection algorithm is similar to the VAE-based algorithm, but the only difference is the NCM. After reshuffling the training dataset and splitting it into proper training dataset and calibration dataset, a deep SVDD model is trained using the proper training dataset. The center of the hypersphere $\boldsymbol{c}$ is fixed as the mean of the representations from the initial pass on the proper training data. After training, the neural network function $\phi(x,\mathcal{W}^*)$ maps an input example x to a representation close to the center $\boldsymbol{c}$. In-distribution inputs are likely concentrated in a relatively small area in the output space, while the OOD inputs will be far away from the center. The distance of the representation to the center $\boldsymbol{c}$ of the hypersphere can be used to evaluate the strangeness of the test example relative to the proper training set and is defined as the NCM

$$\alpha = A_{\text{SVDD}}(x) = ||\phi(x;\mathcal{W}^*) - \boldsymbol{c}||^2. \tag{3.4}$$

The SVDD-based method also uses a learned model to calculate the nonconformity score, and it can be used to compute the nonconformity score in real time. During the offline phase, the nonconformity scores for the calibration data are precomputed using Equation (3.4) and sorted in order to be used at runtime. The offline phase for the SVDD-based detector is shown in Algorithm 2.

Algorithm 2 SVDD-based out-of-distribution detection

```
Input: a training set $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$; a test sequence $(x'_1, \ldots, x'_t, \ldots)$; number of calibration examples $l-m$;
    sliding window size $N$; detector threshold $\tau$
Output: boolean variable $Anom_t$
Offline:
  Split the training set $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$ into the proper training set $\mathcal{D}_{\text{proper}} = \{x_i\}_{i=1}^{m}$ and calibration set
    $\mathcal{D}_{\text{calibration}} = \{x_i\}_{i=m+1}^{l}$
  Train a SVDD model using the proper training set
  for $j = m+1$ to $l$ do
      $\alpha_j = A_{\text{SVDD}}(x_j)$
  end for
Online:
  for $t = 1, 2, \ldots$ do
      $\alpha'_t = A_{\text{SVDD}}(x'_t)$
      $p_t = \frac{|\{i=m+1,\ldots,l\} \mid \alpha_i \geq \alpha'_t|}{l-m}$
      $M_t = \int_0^1 \prod_{i=t-N+1}^{t} \varepsilon p_i^{\varepsilon-1} d\varepsilon$
      $Anom_t \leftarrow M_t > \tau$
  end for
```

3.4.2 Online Detection

In order to improve the robustness of OOD detection, it is desirable to use a sequence of inputs. However, in contrast to the VAE, SVDD is not a generative model and cannot be used to generate multiple examples similar to the input. In the SVDD-based method, we are using a sliding window of inputs $\{x'_i\}_{i=t-N+1}^{t}$. In CPS, the inputs arrive at the perception or end-to-end LEC one by one and they are time-correlated, and therefore, the inputs within the sliding window are not independent. In order to apply this method to CPS, the size of the sliding window N should be carefully chosen based on the auto-correlation analysis on the nonconformity scores of the input sequence $\{\alpha'_i\}_{i=t-N+1}^{t}$. Within this sliding window, the main factor that differentiates consecutive observations are random disturbances and noise. Although the p-values are not guaranteed to be independent and uniformly distributed, OOD inputs will still result in small p-values, and the martingale test can be used to identify sequences with many small values.

In this case, the simple mixture martingale at time t can be defined as

$$M_t = \int_0^1 M_t^{\varepsilon} d\varepsilon = \int_0^1 \prod_{i=t-N+1}^{t} \varepsilon p_i^{\varepsilon-1} d\varepsilon. \tag{3.5}$$

Such martingale will grow only if there are many small p-values in this sliding window indicating OOD inputs are present in the sequence. It should be noted that the martingale M_t does not depend on the order of the input examples $\{x_i'\}_{i=t-N+1}^{t}$. Also, M_t must be initialized for the first steps using, for example, random independent and uniformly distributed p-values. Since we already use a sliding window to compute M_t, we employ a stateless detector based on the value M_t and a predefined threshold τ expressed as $M_t > \tau$.

Algorithm 2 describes the SVDD-based real-time OOD detection. Compared with the VAE, the SVDD based method is more efficient since it does not require generating multiple examples at each step. The martingale M_t can be computed recursively by incorporating the p-value for the new input and omitting the last one in the sliding window.

3.5 Saliency Maps

The proposed OOD detection approach aims to identify inputs that are nonconformal to the training dataset. Although the LEC predictions for such inputs may not have large errors, they are generated from a different probability distribution. However, it is possible that some nonconformal features of the input do not contribute to the LEC prediction. For high-dimensional inputs such as camera images, for example, it is possible that parts of the image do not affect the LEC output. As an illustrative example, if the VAE has difficulty generating fine-granularity details of the original input image, the algorithm presented in Section 3.3 will result in large nonconformity scores. However, such fine-granularity details may not affect the LEC output. This section extends the proposed approach by incorporating *saliency maps* that quantify the spatial support of the LEC prediction of a given input image into the OOD detection.

The purpose of saliency maps is to identify parts of the input image that contribute most to the LEC predictions [92; 93]. In our approach, a saliency map is computed to quantify how much the input features of the image contribute to the LEC output. Then, the saliency map is used to weight the contribution of the input features to the nonconformity scores. Therefore, the detection algorithm weights the input features based on their influence on the output of the perception or end-to-end LEC in the CPS architecture.

We utilize two algorithms for computing the saliency maps, Integrated-Gradients Optimized Saliency (I-GOS) and VisualBackProp (VBP). I-GOS is introduced in [102] that optimizes for the saliency map so that the classification scores on the masked image would maximally decrease, which reflects the input features that have greatest influence on the prediction. VBP is an algorithm that aims to highlight important regions that contribute towards the predictions made by convolutional neural networks [98]. The intuition of VBP is that the feature maps contain less irrelevant information to the prediction when moving deeper into the network. Since the resolution of the feature maps becomes lower for deeper layers, VBP computes the saliency map by back-propagating the information of feature maps from deeper layers while simultaneously increasing the resolution.

Consider an LEC $y = f(x)$ defining a mapping from the input x to output y used to perform regression or classification. The input image x is of size $H \times W \times C$ where H, W, and C denote the height, width, and channel, respectively. We denote each input element as $x^{h,w,c}$ where $h \in \{0, \ldots, H-1\}, w \in \{0, \ldots, W-1\}$, and $c \in \{0, \ldots, C-1\}$. I-GOS is initially designed for the classification problem. As for regression, the algorithm can be modified by optimizing a saliency map s' so that the regression result on the saliency-masked image would maximally change. In order to use the saliency map in the detection algorithm, we convert s' to a grayscale image s of size $H \times W$. VBP uses the feature maps outputted by the convolutional layers and can be used for both regression and classification tasks. In VBP, the generated saliency map s is already grayscaled of size $H \times W$. Moreover, in order to deal with the problem that different images will have different total contributions, the saliency map should be normalized by the sum of contributions for all pixels in the image. In summary, we can define a function $s = G(x_0; f)$ that generates a grayscale saliency map s for the LEC f given input image x_0 using either I-GOS or VBP.

3.5.1 VAE-based Detection with Saliency Maps

We define the NCM by weighting the squared error between the input example x and a generated example $\hat{x}$ from the VAE using the saliency map s

$$\alpha = A_{\text{VAE-S}}(x, \hat{x}, s) = \frac{1}{H \times W \times C} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \sum_{c=0}^{C-1} s^{h,w} (x^{h,w,c} - \hat{x}^{h,w,c})^2. \tag{3.6}$$

Therefore, the input features that influence the LEC predictions contribute more to the NCM.

During the offline phase of the algorithm, similar to the VAE-based detection method introduced in Sec. 3.3, the training dataset is reshuffled and split into a proper training set and a calibration set. The proper training dataset is used to train both the LEC and the VAE network. For each example in the calibration dataset, we compute the saliency map, and the nonconformity score is computed using Equation (3.6). The nonconformity scores of the calibration data are sorted and stored for monitoring at runtime.

At runtime, given an input example x'_t, the saliency map $s'_t = G(x'_t; f)$ is used to compute the nonconformity score. As described in Sec. 3.3, for each input example x'_t, N examples $\{\hat{x}'_{t,k}\}_{k=1}^{N}$ are generated from the VAE. For each generated example $\hat{x}'_{t,k}$, the nonconformity score $\alpha'_{t,k}$ is computed by $\alpha'_{t,k} = A_{\text{VAE-S}}(x'_t, \hat{x}'_{t,k}, s'_t)$

Algorithm 3 VAE-based out-of-distribution detection using saliency maps

```
Input: a training set $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$; a test sequence $(x'_1, \dots, x'_t, \dots)$; number of calibration examples $l-m$; monitored LEC $f(\cdot)$; number of examples $N$ generated by the VAE; threshold $\tau$ and parameter $\omega$ of CUSUM detector
Output: boolean variable $Anom_t$
Offline:
Split the training set $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$ into the proper training set $\mathcal{D}_{\text{proper}} = \{x_i\}_{i=1}^{m}$ and calibration set $\mathcal{D}_{\text{calibration}} = \{x_i\}_{i=m+1}^{l}$
Train a VAE using the proper training set
for $j = m+1$ to $l$ do
    Generate $\hat{x}_j$ using the trained VAE
    Compute the saliency map $s_j = G(x_j; f)$
    $\alpha_j = A_{\text{VAE-S}}(x_j, \hat{x}_j, s_j)$
end for
Online:
for $t = 1, 2, \dots$ do
    Compute the saliency map $s'_t = G(x'_t; f)$
    for $k = 1$ to $N$ do
        Generate $\hat{x}'_{t,k}$ using the trained VAE
        $\alpha'_{t,k} = A_{\text{VAE-S}}(x'_t, \hat{x}'_{t,k}, s'_t)$
        $p_{t,k} = \frac{|\{i=m+1,\dots,l\} \mid \alpha_i \geq \alpha'_{t,k}|}{l-m}$
    end for
    $M_t = \int_0^1 \prod_{k=1}^{N} \varepsilon p_{t,k}^{\varepsilon-1} d\varepsilon$
    if $t = 1$ then
        $S_t = 0$
    else
        $S_t = \max(0, S_{t-1} + M_{t-1} - \delta)$
    end if
    $Anom_t \leftarrow S_t > \tau$
end for
```

(Equation (3.6)). The corresponding p-value $p_{t,k}$ is calculated as Equation (3.2). Most of the p-values are expected to be much greater than 0, and the martingale M_t computed by Equation (3.3) is used to test if x'_t is an OOD input. Finally, a CUSUM detector is used to generate alarms for OOD inputs. Algorithm 3 summarizes the VAE-based OOD detection using saliency maps.

3.5.2 SVDD-based Detection with Saliency Maps

In order to incorporate the saliency maps into the SVDD-based detection method, the input x is masked by the saliency map s and is used as the new input to the SVDD network. Each element of the masked image denoted by $\tilde{x}$ is computed by $\tilde{x}^{h,w,c} = x^{h,w,c} \cdot s^{h,w}$.

During the offline phase, we compute the saliency map for each example in the training dataset and use it to mask the example to create a new training dataset $\{\tilde{x}_i\}_{i=1}^{l}$. Then, the new training dataset $\{\tilde{x}_i\}_{i=1}^{l}$ is reshuffled and split into the proper training set $\{\tilde{x}_i\}_{i=1}^{m}$ and calibration set$\{\tilde{x}_i\}_{i=m+1}^{l}$. The proper training set is used to train a new SVDD network which maps the inputs to a representation in a lower-dimensional hypersphere suitable for anomaly detection. The NCM A_{SVDD} is defined as the distance of the representation to the center of the hypersphere (Equation (3.4)). For each example in the new calibration dataset, the nonconformity score is computed using A_{SVDD} and sorted for runtime monitoring.

Algorithm 4 SVDD-based out-of-distribution detection using saliency maps

```
Input: a training set $\mathcal{D}_{\text{train}} = \{x_i\}_{i=1}^{l}$; a test sequence $(x'_1, \ldots, x'_t, \ldots)$; number of calibration examples $l-m$;
    monitored LEC $f(\cdot)$; sliding window size $N$; detector threshold $\tau$
Output: boolean variable $Anom_t$
Offline:
for $i = 1$ to $l$ do
    Compute the saliency map $s_i = G(x_i; f)$
    Mask the input $x_i$ using the saliency map $s_i$ and get $\tilde{x}_i$
end for
Split the training set $\{\tilde{x}_i\}_{i=1}^{l}$ into the proper training set $\{\tilde{x}_i\}_{i=1}^{m}$ and calibration set $\{\tilde{x}_i\}_{i=m+1}^{l}$
Train a SVDD model using the proper training set
for $j = m+1$ to $l$ do
    $\alpha_j = A_{\text{SVDD}}(\tilde{x}_j)$
end for
Online:
for $t = 1, 2, \ldots$ do
    Compute the saliency map $s'_t = G(x'_t; f)$
    Mask the input $x'_t$ using the saliency map $s'_t$ and get input $\tilde{x}'_t$
    $\alpha'_t = A_{\text{SVDD}}(\tilde{x}'_t)$
    $p_t = \frac{|\{i=m+1,\ldots,l\} \mid \alpha_i \geq \alpha'_t|}{l-m}$
    $M_t = \int_0^1 \prod_{i=t-N+1}^{t} \varepsilon p_i^{\varepsilon-1} d\varepsilon$
    $Anom_t \leftarrow M_t > \tau$
end for
```

During runtime monitoring, given the test input x'_t, the algorithm computes the saliency map $s'_t = G(x'_t; f)$ and uses saliency-masked input $\tilde{x}'_t$ as the input to the SVDD network in order to compute the nonconformity score α'_t as well as the p-value for the test example. Similar to Sec. 3.4, a sliding window is used to adapt the martingale test, and a stateless detector is applied to generate alarms for OOD examples. The SVDD-based detection algorithm using saliency maps is shown in Algortithm 4.

3.6 Evaluation

We evaluate the proposed approach using (1) an Advanced Emergency Braking System (AEBS), (2) a Self-Driving End-to-end Controller (SDEC), and (3) an Autonomous Vehicle Seasonal Dataset (AVSD). The AEBS and SDEC are implemented using CARLA [103], an open-source simulator for self-driving cars. AVSD evaluates the approach using the Ford autonomous vehicle seasonal dataset [104]. All the experiments presented in this chapter are conducted on a 16-core i9 desktop with 32 GB RAM and a single RTX 2080 GPU with 8 GB video memory.

3.6.1 Advanced Emergency Braking System

3.6.1.1 Experimental Setup

The architecture of the AEBS is shown in Figure 3.1. A perception LEC is used to detect the nearest front obstacle on the road and estimate the distance. The distance together with the velocity of the host car are used as inputs to a reinforcement learning controller whose objective is to generate the appropriate braking force in order to safely and comfortably stop the vehicle.

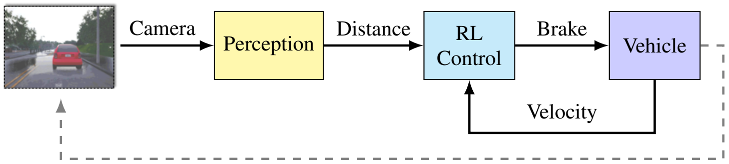


Figure 3.1: Architecture of AEBS.

The desirable behavior is illustrated in Figure 3.2. The AEBS detects a stopped lead car and applies the brake to decelerate and avoid the potential collision. The initial velocity of the host vehicle is v_0, and the initial distance between the host car and the obstacle is d_0. The goal of the controller is to stop the car between $L_{\min}$ and $L_{\max}$. The sampling period used in the simulation is $\Delta t = 1/20\,\mathrm{s}$. In order to simulate realistic scenarios, we introduce uncertainty into the system. The initial velocity v_0 is uniformly sampled between $90\,\mathrm{km/h}$ and $100\,\mathrm{km/h}$, and the initial distance d_0 is approximately $100\,\mathrm{m}$. CARLA allows controlling the precipitation in the simulation using a parameter, which takes values in $[0, 100]$. For training the perception LEC, and also the VAE and deep SVDD used for OOD detection, the precipitation parameter is randomly sampled from the interval $[0, 20]$. The uncertainty introduced affects the error of the perception LEC. It should be noted that this is just a visual effect, and it does not affect the physical behavior of the car.

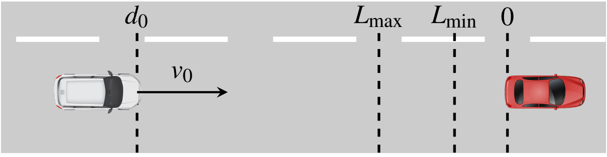


Figure 3.2: Illustration of AEBS.

The perception LEC is implemented using a Convolutional Neural Network (CNN), which is trained using supervised learning with a training dataset consisting of 8160 images obtained by varying the simulation parameters as described above. The perception LEC has three layers of $24/36/48 \times (5 \times 5)$ filters with ReLU activations and 2×2 strides, two layers of $64/64 \times (3 \times 3)$ filters with ReLU activations and 1×1 strides, three fully connected layers of $100/50/10$ units with ReLU activations and an output layer of size 1 with Sigmoid activation. After 100-epoch training, the mean absolute errors for training and testing are $0.54\,\mathrm{m}$

and 0.56 m respectively and are used to select $L_{\min}$ and ensure safety. The reinforcement learning controller is trained using the DDPG algorithm [107] with 1000 episodes and reward function which aims to stop the vehicle between $L_{\min} = 1\,\mathrm{m}$ and $L_{\max} = 3\,\mathrm{m}$. A simulation run is shown in Figure 3.3. Initially, the distance between the host and the lead car is 98.02 m, and the velocity of the host car is 97.13 km/h (= 26.98 m/s). After 140 steps or 7.00 s, the host vehicle stops at 1.85 m from the lead car.

3.6.1.2 VAE and SVDD training

The dataset with the 8160 images used for training the perception LEC is used as the proper training dataset. In addition, using simulations with the same random parameters, we collect 2040 images for the calibration set. It should be emphasized that the proper training set and the calibration set should be reshuffled before training the VAE or deep SVDD. We use a VAE with four layers of $32/64/128/256 \times (5 \times 5)$ filters with Exponential Linear Unit (ELU) activations and 2×2 max-pooling, one fully connected layer of size 1568 with ELU activation, 1024 latent space, and a symmetric deconvolutional decoder. A simple two-phase learning schedule is employed with initial searching learning rate $\eta = 10^{-4}$ for 250 epochs, and subsequently fine-tuning $\eta = 10^{-5}$ for 100 epochs. This model is used in the VAE-based OOD detection method both with and without the saliency maps in order to compute the NCM.

The deep SVDD is similar with four convolutional layers of $32/64/128/256 \times (5 \times 5)$ filters with ELU activations and 2×2 max-pooling, followed by one fully connected layer of 1568 units. As suggested in [101], we first train a deep convolutional autoencoder (DCAE) to initialize the deep SVDD. After $250\ (\eta = 10^{-4}) + 100\ (\eta = 10^{-5})$ epochs of DCAE training, we copy the weights to the SVDD and set the hypersphere center $\boldsymbol{c}$ to the mean of the reduced space of the initial forward inference. The one-class deep SVDD objective is used as the loss, and the neural network is trained for additional $150\ (\eta = 10^{-4}) + 100\ (\eta = 10^{-5})$ epochs. Since the inputs to the SVDD-based detector are different for the case saliency maps are used, two different deep SVDD networks are trained using the original inputs and saliency-masked inputs, respectively.

3.6.1.3 Experimental Results

To characterize the performance of the OOD detection, we use multiple simulation episodes that include in-distribution and OOD examples. Each episode starts with a random initial velocity v_0 of the host car. The AEBS is activated upon detection of the lead car by the camera as implemented in CARLA. We vary the precipitation parameter r as

$$r = \begin{cases} r_0 & \text{for} \quad t < t_0 \\ r_0 + \beta(t - t_0) & \text{for} \quad t_0 \le t \le t_1 \\ r_0 + \beta(t_1 - t_0) & \text{for} \quad t > t_1 \end{cases}$$

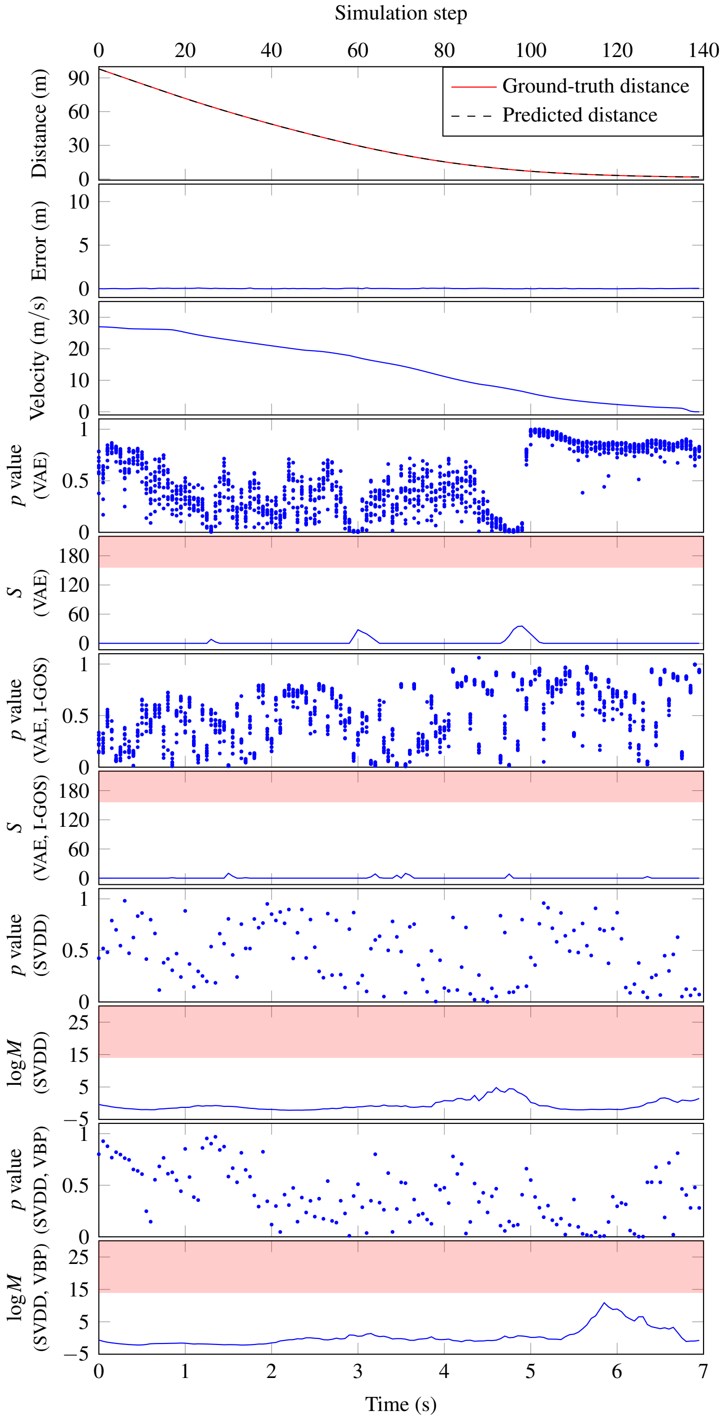


Figure 3.3: An episode with in-distribution inputs in AEBS (detector parameter: (1) VAE-based: $N = 10$, $\omega = 6$, $\tau = 156$; (2) SVDD-based: $N = 10$, $\tau = 14$).

where r_0 is the initial precipitation uniformly sampled from $[0,10]$; $t_0 \in \{10,11,\dots,30\}$ is selected randomly as the time step the precipitation starts to increase; $t_1 \in \{90,91,\dots,110\}$ is selected randomly as the time

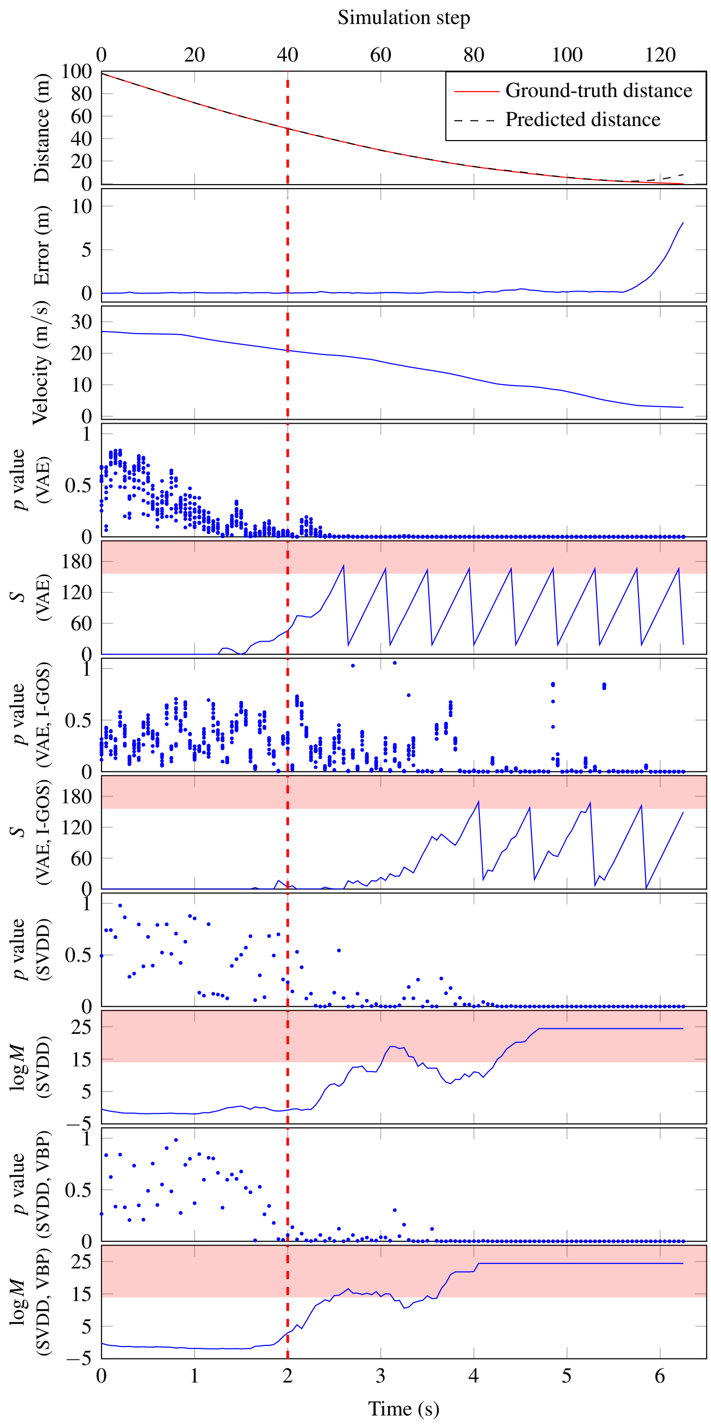


Figure 3.4: An episode with OOD inputs in AEBS (detector parameter: (1) VAE-based: $N = 10$, $\omega = 6$, $\tau = 156$; (2) SVDD-based: $N = 10$, $\tau = 14$).

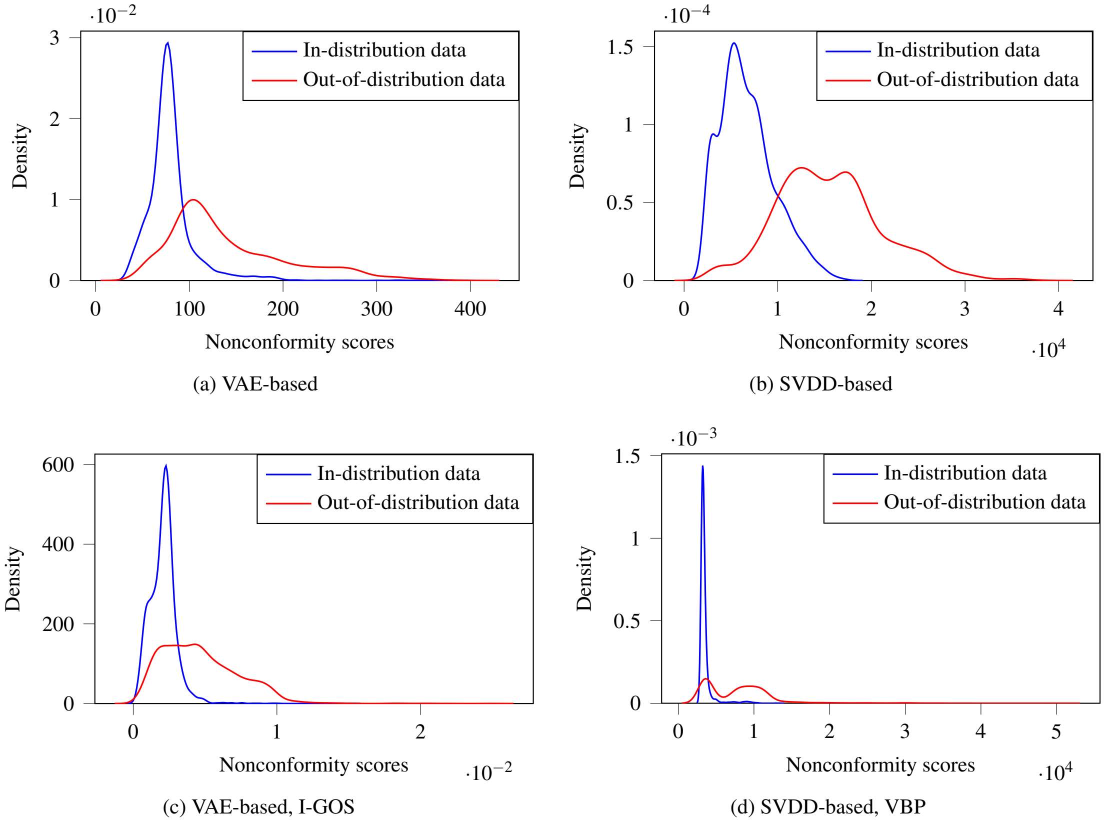
step the precipitation stops increasing; and $\beta \in [0.1, 0.5]$ is a randomly selected slope. In some episodes r is always below 20 (in-distribution), while in other episodes r exceeds 20 and it is assumed that the perception

LEC receives OOD inputs. We simulate 200 episodes, and 108 of them are in-distribution while 92 of them contain OOD inputs.

In order to show that the VAE and deep SVDD model are trained successfully and can be used for OOD detection, we plot the distributions of the nonconformity scores using different NCMs (VAE- and SVDD-based using or not using saliency maps) in Figure 3.5. More specifically, for VAE-based NCMs, an input is fed into the VAE model to generate a single example, and the nonconformity score is computed as the reconstruction error between the input and generated example (if saliency map is used, the nonconformity score should be weighted by the saliency map); for SVDD-based NCMs, SVDD model maps the input to the hypersphere, and the nonconformity score is the distance between the representation to the center (if saliency map is used, the input to SVDD model should be masked by the saliency map). From the plots, we can see that, for all different NCMs, the nonconformity scores of in-distribution data are much smaller than OOD data. Furthermore, we also measure the overlapping area in each plot, which indicates the false alarm rate of the detector. The overlapping area of the distributions is inevitable since the precipitation parameter is increased gradually during our data collection, which results in some OOD data having small changes from the in-distribution data. Comparing Figure 3.5a with Figure 3.5b, the SVDD-based NCMs have better performance than the VAE-based NCMs. Moreover, in order to compare the performance between the methods using one example and multiple examples, we also plot the distributions of the logarithm of martingales using different NCMs (VAE- and SVDD-based using or not using saliency maps) in Figure 3.6. In VAE-based methods, for each input, multiple examples are generated, and the martingale is computed based on the corresponding p-values. As for SVDD-based methods, the martingale value is computed by applying the sliding window technique. The overlapping area in Figure 3.6 using multiple examples are smaller than ones of corresponding plots in Figure 3.5 using a single example, which demonstrates that the performance of the detector can be improved by incorporating multiple examples.

We illustrate the approach using two episodes, and we plot the ground-truth and the predicted distance to the lead car, the velocity of the host car, the p-value of the VAE-based method, and the output of the detector S computed using the logarithm of M_t and $\omega = 6$. Since M_t takes very large values, $\log M_t$ is used. We also plot the p-value of the SVDD-based method and the logarithm of the SVDD-based martingale.

The results of the VAE- and SVDD-based methods with saliency maps are plotted similarly. Figure 3.7 shows one frame of the camera image and its saliency maps generated by I-GOS and VBP. Due to space limitations, we include only the results of the VAE-based using I-GOS and the SVDD-based method using VBP. We use $N = 10$ for the number of examples generated by the VAE For the SVDD-based method, in order to reasonably choose the size of the sliding window, we perform the auto-correlation analysis: we measure the auto-correlation coefficients on the nonconformity scores of the input sequence and compute the mean



(a) VAE-based

(b) SVDD-based

(c) VAE-based, I-GOS

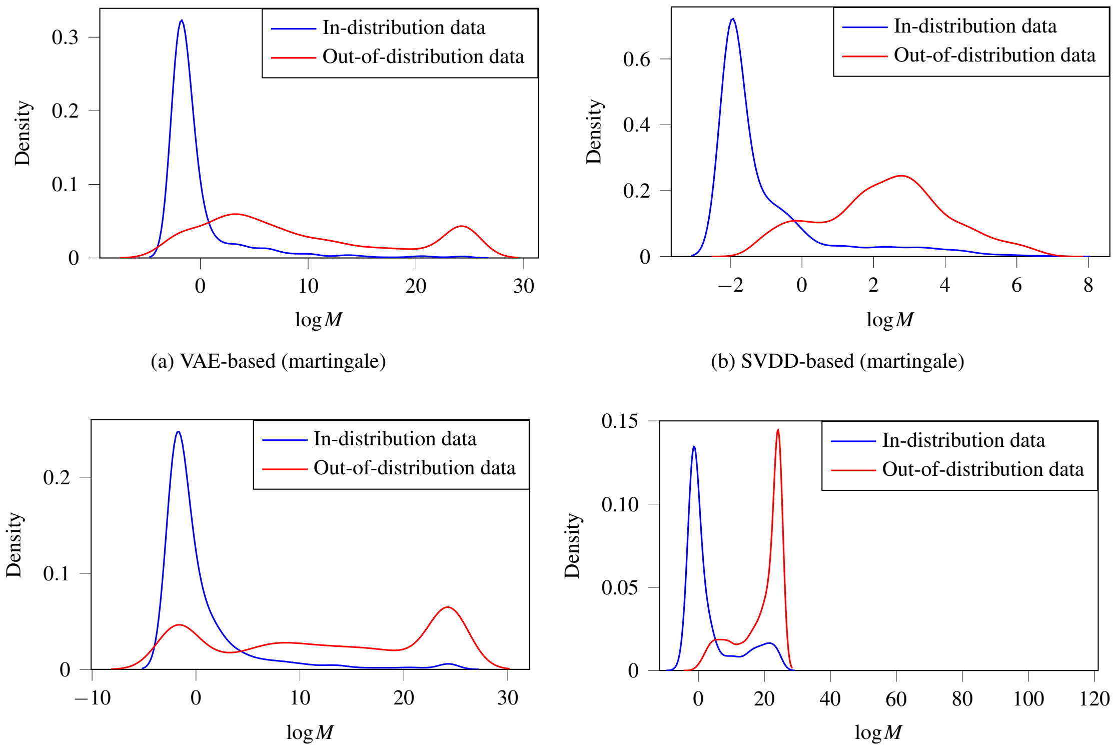
(d) SVDD-based, VBP

Figure 3.5: Distributions of the nonconformity scores in AEBS.

absolute auto-correlation within different time scales, which is shown in Figure 3.8. The auto-correlation becomes smaller than 0.1 when the time scale is greater than 9. Therefore, we can choose the time scale greater than 9. In the illustrative episode, the size of the sliding window is set to 10.

Figure 3.3 shows simulation results for the in-distribution case. Most of the p-values are much greater than 0, and the martingale for all four approaches is small. The VAE-based method is more sensitive than the SVDD as indicated by the larger value around 5 s. In this scenario, there is a speed limit traffic sign that is not accurately reconstructed by the VAE resulting in smaller p-values. After the car passes the traffic sign, the p-values increase, and the martingale decreases. The effect in the SVDD-based method is attenuated, since we use a sliding window.

An episode with OOD inputs is shown in Figure 3.4. The parameter r exceeds 20 at time step 40 (2.0 s). The error of the perception LEC starts increasing and reaches almost 9 m. The controller is misled by the perception LEC and does not stop the car which collides with the lead car (velocity is greater than 0 when ground-truth distance comes to 0). For all four detection methods, the martingale grows as the p-values become smaller.



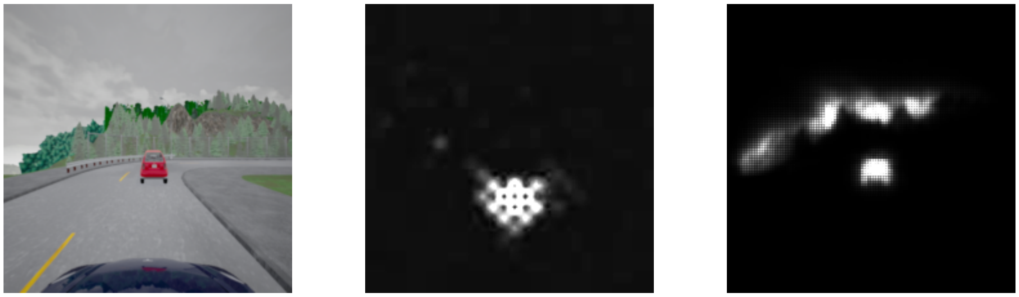
(a) VAE-based (martingale)

(b) SVDD-based (martingale)

(c) VAE-based, I-GOS (martingale)

(d) SVDD-based, VBP (martingale)

Figure 3.6: Distributions of the martingales in AEBS.



(a) Original image

(b) Saliency Map (I-GOS)

(c) Saliency Map (VBP)

Figure 3.7: Original image and its saliency maps for AEBS.

We evaluate the approach for the 200 episodes generated by considering different values of N. We run each episode, and if an alarm is raised, we stop the simulation, and we check if the alarm is false. We compute the detection delay as the number of frames from the time r exceeds 20. We select the detector parameters ω and τ using a simple search for achieving the average detection delay less than 25 frames and the number of false alarms less than 2. It should be emphasized that we can choose the detector parameters only based on the nonconformity scores of in-distribution data as the other unsupervised detection method. However, in our method, some OOD data can be collected to improve the detector's performance. With

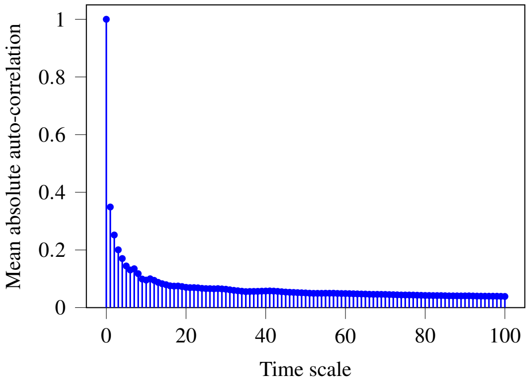


Figure 3.8: Mean absolute auto-correlation of the nonconformity scores in AEBS.

more OOD episodes collected, the parameters can be better tuned, and the detector's performance can be further improved. Table 3.1 reports the results for the VAE and SVDD-based methods. Further, in order to compare the performance of the detectors, we use the same parameters for the methods with saliency maps, and Table 3.1 summarizes the results. The number of false alarms is very small, and the delay for detection is smaller than 20 frames or 1 s for all four detection methods. The VAE-based method uses a cumulative sum (CUSUM) procedure, which is parameterized by ω and τ. These two parameters control the trade-off among false positive, false negative, and detection delay. The large value of ω and τ will result in the detector under-sensitive, and the detector will have a large number of false negatives and long detection delay. On the other side, the small value of ω and τ will have an over-sensitive detector. Similar to the VAE-based method, the SVDD-based method is parameterized by the size of sliding window N and the threshold of stateless detector τ. The large value of N and the small value of τ will lead to an under-sensitive detector, and vice versa.

From the results for the methods with saliency maps, it can be seen that the detector is more robust to small variations in the input that do not affect the LEC prediction (and the prediction error). For example, when the precipitation parameter exceeds slightly the defined threshold for OOD inputs (20), the prediction error remains very small. Figure 3.4 shows that for small changes from the in-distribution data, the p-values are larger in the case of saliency maps, and therefore, OOD detection occurs later which is more consistent with the prediction error. The results are similar for the other experiments as summarized in the increased average delay in Tables 1-4. The detection delay in the methods with saliency maps is slightly greater, which is because the detection methods with saliency maps consider the influence of the input on the LEC output. In addition, missed alarms is an attractive performance metric for such a detection system in CPS. Therefore we tune the parameters achieving 0 false positives and report the missed alarms (false negatives) in Table 3.2. The results reveal that our approach has a small number of missed alarms for different values of N.

Table 3.1: False alarms and average delay in AEBS.

NCM	Parameters $(N,\omega,\tau)/(N,\tau)$	False positive	False negative	Average delay (frames)
VAE	$5,5,42$	$2/108$	$0/92$	17.91
	$5,5,49$	$0/108$	$0/92$	19.84
	$10,6,156$	$0/108$	$0/92$	18.65
	$10,10,106$	$0/108$	$0/92$	19.30
	$20,16,250$	$2/108$	$0/92$	17.63
	$20,18,240$	$0/108$	$0/92$	18.46
VAE, IGOS	$5,5,42$	$1/108$	$0/92$	22.32
	$5,5,49$	$1/108$	$0/92$	24.04
	$10,6,156$	$0/108$	$0/92$	23.36
	$10,10,106$	$0/108$	$0/92$	23.70
	$20,16,250$	$1/108$	$0/92$	21.14
	$20,18,240$	$0/108$	$0/92$	22.83
SVDD	$10,12$	$1/108$	$0/92$	14.38
	$10,14$	$0/108$	$0/92$	17.78
	$15,13$	$2/108$	$0/92$	13.48
	$15,15$	$0/108$	$0/92$	15.36
	$20,16$	$1/108$	$0/92$	12.02
	$20,17$	$0/108$	$0/92$	13.29
SVDD, VBP	$10,12$	$2/108$	$0/92$	15.32
	$10,14$	$1/108$	$0/92$	18.10
	$15,13$	$1/108$	$0/92$	14.68
	$15,15$	$0/108$	$0/92$	16.42
	$20,16$	$1/108$	$0/92$	13.28
	$20,17$	$0/108$	$0/92$	14.32

Table 3.2: Missed alarms and average delay in AEBS.

NCM	Parameters $(N,\omega,\tau)/(N,\tau)$	Missed alarms (False negative)	Average delay (frames)
VAE	$5,6,43$	$1/92$	19.54
	$10,8,132$	$0/92$	19.46
	$20,19,235$	$1/92$	19.02
VAE, I-GOS	$5,5,53$	$1/92$	24.08
	$10,8,130$	$2/92$	24.05
	$20,20,232$	$0/92$	23.96
SVDD	$10,15$	$1/92$	18.12
	$15,14$	$0/92$	14.70
	$20,18$	$0/92$	13.94
SVDD, VBP	$10,15$	$1/92$	19.21
	$15,14$	$0/92$	15.78
	$20,18$	$1/92$	14.76

3.6.2 Self-driving End-to-end Control

3.6.2.1 Experimental Setup

The CARLA simulator comes with a Self-Driving End-to-end Controller (SDEC) trained using imitation learning. The SDEC uses camera images as inputs and computes steering, acceleration, and brake actuation signals applied to the car. The architecture is shown in Figure 3.9.

The SDEC is implemented using a CNN trained by conditional imitation learning with 14 hours of driving

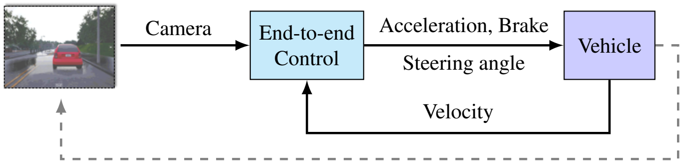
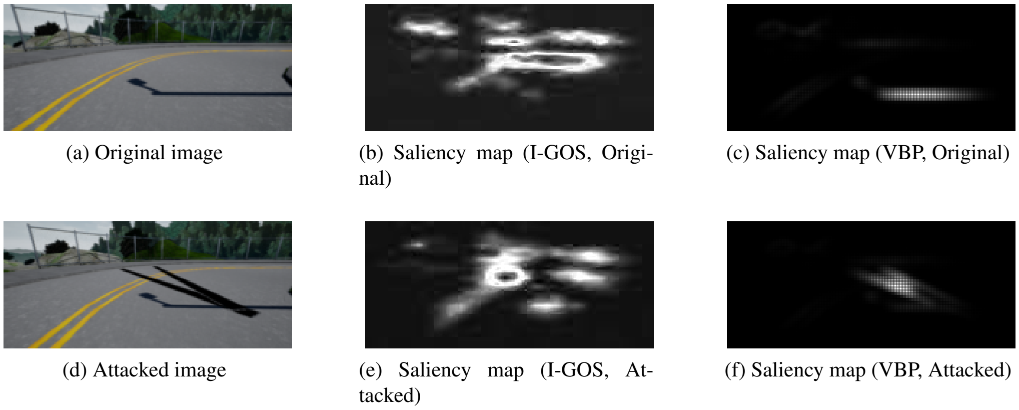


Figure 3.9: Architecture of SDEC.

data recorded by human drivers [103]. The sampling period used here is $\Delta t = 1/10\,\mathrm{s}$. For this example, our objective is to evaluate if the method can be used to detect a class of adversarial attacks. An approach for designing physically realizable attacks in end-to-end autonomous driving is presented in [105], and a novel class of hijacking attacks is introduced where painted lines on the road cause the self-driving car to follow a target path. Figure 3.10b shows an image with the painted pattern on the road.


(a) Original image

(b) Saliency map (I-GOS, Original)

(c) Saliency map (VBP, Original)

(d) Attacked image

(e) Saliency map (I-GOS, Attacked)

(f) Saliency map (VBP, Attacked)

Figure 3.10: Original image, physical adversarial image and their saliency maps for SDEC.

In order to train the VAE and SVDD, we collect training data using episodes without attacks. First, we generate 633 images in two different weather patterns (clear noon and cloudy noon) and three different scenarios (turning right, turning left, and going straight). Then, we reshuffle and randomly split the training data into 506 images for the proper training dataset and 127 images for the calibration set. We use the same VAE and SVDD architectures and hyperparameters as in the AEBS.

3.6.2.2 Experimental Results

The evaluation focuses on the *Right Corner Driving* case, which is reported as more vulnerable [105]. We run 105 simulation episodes described in [105] with different attacks such as positions and rotations of the two black lines which are chosen to cause traffic infractions. In 69 out of the 105 episodes, the attack is successfully causing a vehicle crash. Our approach detects the attacks in all 105 episodes. We plot the p-values and detector output S of the VAE-based method, the p-values and the logarithm of the SVDD-based martingale for episodes with and without attacks shown in Figure 3.11 and 3.12, respectively. The size of the sliding window N for SVDD-based method is chosen based on the auto-correlation analysis on the nonconformity scores of the input sequence. The mean absolute auto-correlation is smaller than 0.1 when the time scale is greater than 7, and therefore, we can choose $N = 10$ in our illustrative episodes. In addition, Figure 3.11 and 3.12 show the results of the detection methods using saliency maps. Figure 3.10 shows the original (in-distribution) image, physical adversarial image, and their saliency maps. For the in-distribution (no-attack) episode, most of the p-values are much greater than 0, while the martingales for all approaches are small. In the adversarial episode, there are two black lines painted on the road as shown in Figure 3.10d, and the vehicle is misled to a crash. The p-values are almost 0, and the martingales grow very large, indicating the input images are out of distribution.

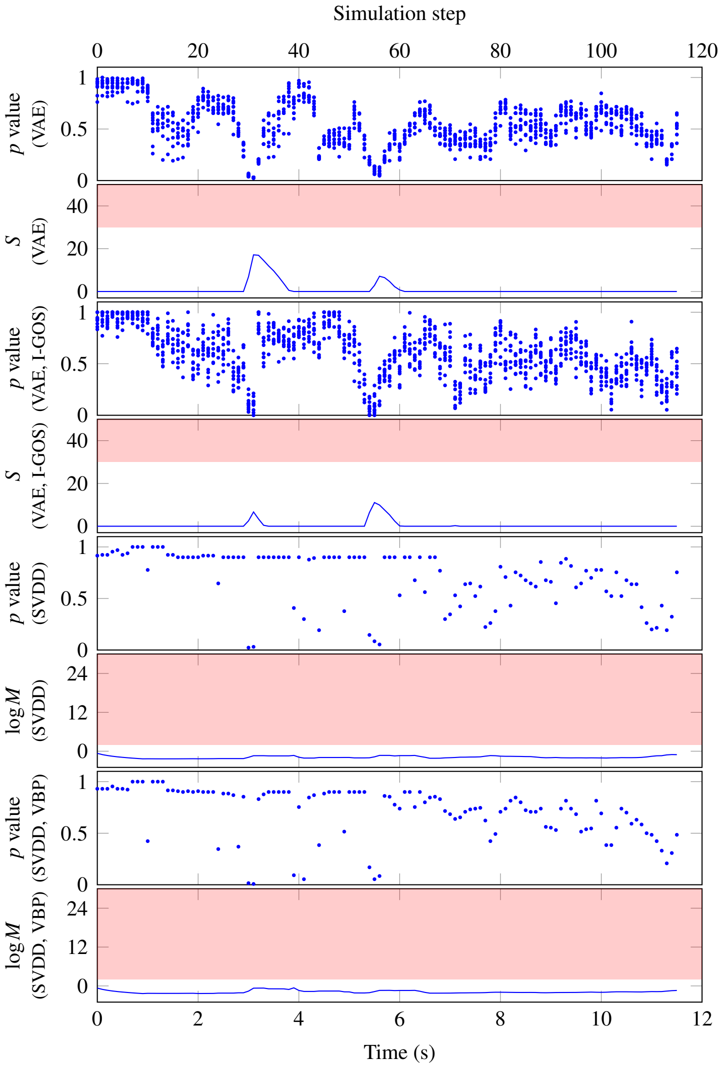


Figure 3.11: An episode with in-distribution inputs in SDEC (detector parameter: (1) VAE-based: $N = 10$, $\omega = 1$, $\tau = 30$; (2) SVDD-based: $N = 10$, $\tau = 2$).

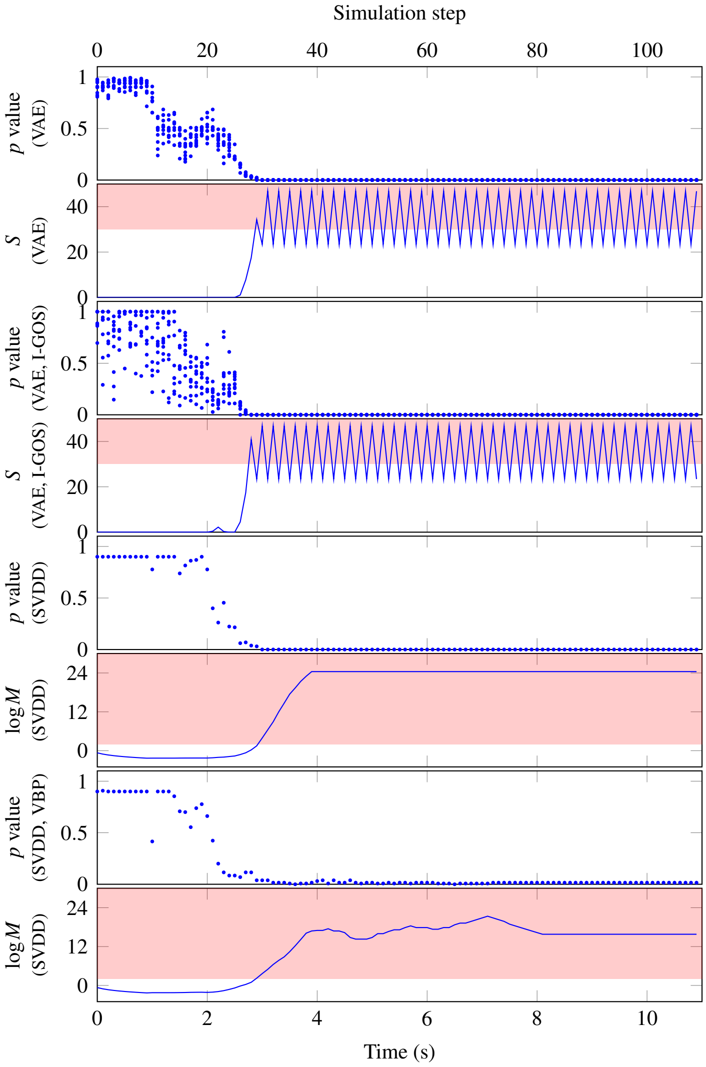


Figure 3.12: An episode with attacked inputs in SDEC (detector parameter: (1) VAE-based: $N = 10$, $\omega = 1$, $\tau = 30$; (2) SVDD-based: $N = 10$, $\tau = 2$).

3.6.3 Autonomous Vehicle Seasonal Dataset

3.6.3.1 Experimental Setup

In order to evaluate the approach in a real-world environment, we use the Ford Autonomous Vehicle Seasonal Dataset (AVSD) [104], which is used to train a perception neural network whose task is to predict the heading changes of the host vehicle from the images captured by a camera. The sampling rate of the camera is $\Delta t = 1/7\,\text{s}$. The dataset provides the raw images and ground-truth pose (position and orientation) of the vehicle in a global frame. The images are collected under seasonal variation in weather and include various lighting, construction, and traffic conditions experienced in typical urban environments [104]. For using the dataset to evaluate our approach, we pre-process the data to compute heading changes of the host vehicle by converting quaternions to Euler angles and calculating the yaw difference, and we synchronize the input images with the heading changes.

We select the dataset for one vehicle (V1) and one drive (Log 1) as the training dataset. The set contains data collected in cloudy weather and freeway, overpass, and bridge drive scenarios. We reshuffle and randomly split the training data into 3556 images for the proper training dataset and 889 for the calibration set. The proper training dataset is used to train the perception, VAE, and SVDD networks.

The perception network is a CNN whose architecture is similar to the NVIDIA end-to-end self-driving controller [108]. After 100-epoch training, the mean absolute error for training and testing are 0.012° and 0.016°, respectively. The VAE and SVDD networks used for detection use the same architectures and hyperparameters as in the AEBS and SDEC examples.

3.6.3.2 Experimental Results

We evaluate the approach using 100 episodes, 50 of which are in distribution and 50 are out of distribution. For OOD data, we use a set (Log 3) with data collected in sunny weather and residential driving scenario. Each episode contains 140 sequential frames and has duration 20 s. We illustrate the approach using two episodes, and we plot the prediction errors of the heading change, the p-values and detector output of the VAE-based method with and without saliency maps, and the p-values and the logarithm of the SVDD-based martingale with and without saliency maps. The results are shown in Figure 3.13 and Figure 3.14, respectively. Similar to the AEBS and SDEC, in order to choose a reasonable size of sliding window N for SVDD-based method, we analyze the auto-correlation function on the nonconformity scores of the input sequence. The mean absolute auto-correlation is smaller than 0.1 when the time scale is greater than 6. In our illustrative episode, we choose $N = 10$.

We also show the in-distribution and OOD input images and their saliency maps in Figure 3.15. For the in-distribution episode, most of the p-values are far away from 0, and the martingales in all approaches are small indicating there is no OOD input detected. In the OOD episode, the predicted errors are greater than the in-distribution episode. For all approaches, the p-values are small, and the martingales grow very large showing the input images are not in the same distribution as the training dataset.

We also report the number of false alarms and average detection delay time in Table 3.3, and the number of missed alarms in Table 3.4 by considering different values of N and detector parameters ω and τ . From the results, the number of false alarms and missed alarms are very small and the delay for detection is smaller than 20 frames. The results also show that the SVDD-based methods have short detection delay than the VAE-based methods.

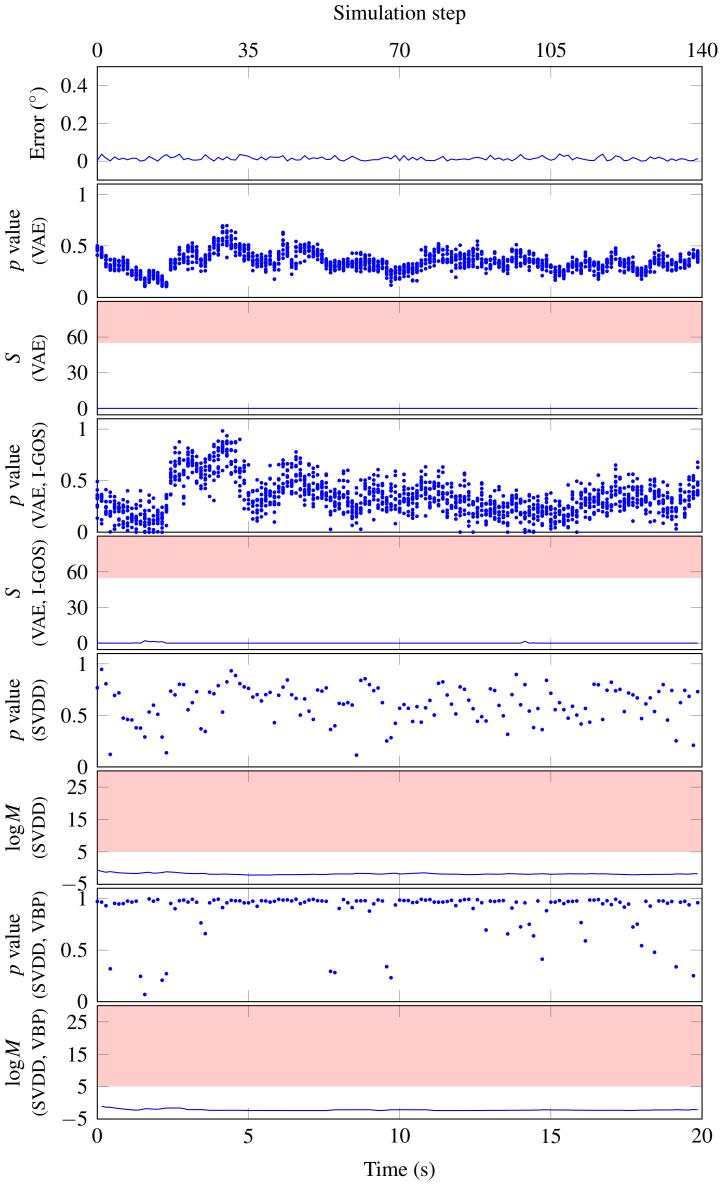


Figure 3.13: An episode with in-distribution inputs in AVSD (detector parameter: (1) VAE-based: $N = 10$, $\omega = 8$, $\tau = 55$; (2) SVDD-based: $N = 10$, $\tau = 5$).

3.6.4 Computational Efficiency

The VAE-based and SVDD-based methods can compute the nonconformity scores in real-time without storing training data. Table 3.5 reports the minimum (min), first quartile (Q_1), second quartile or median (Q_2), third quartile (Q_3), and maximum (max) of (1) the execution times of the LECs in AEBS, SDEC, and AVSD, (2) the execution times of the I-GOS- and VBP-based saliency map algorithms, and (3) the execution times of the VAE-based and SVDD-based detectors with and without saliency maps for different values of N.

The results show that the VBP-based algorithm is slightly more efficient than the I-GOS. Since the VAE-

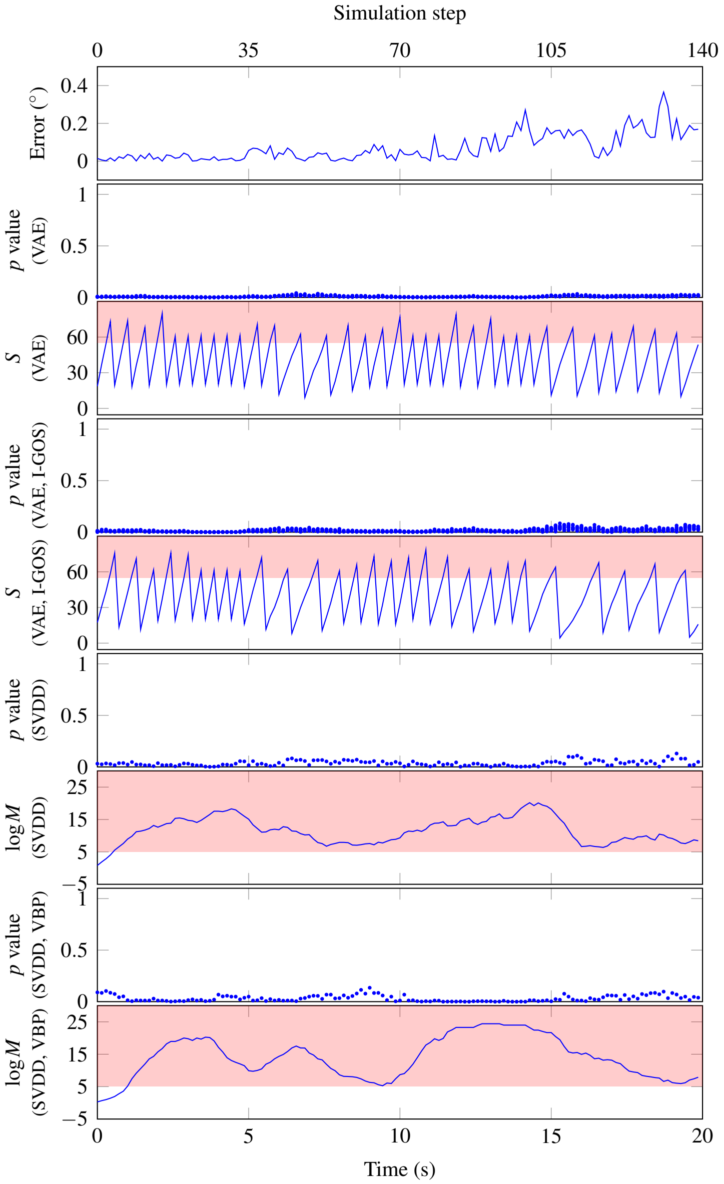
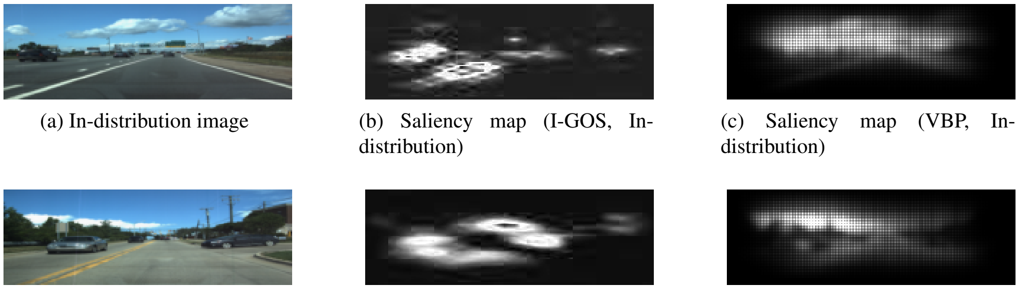


Figure 3.14: An episode with OOD inputs in AVSD (detector parameter: (1) VAE-based: $N = 10$, $\omega = 8$, $\tau = 55$; (2) SVDD-based: $N = 10$, $\tau = 5$).

based method uses N examples in each time step to compute the nonconformity scores, the execution time is larger than the execution time of the SVDD-based method. The execution time of SVDD-based detection method is independent of the window size N since the martingale can be computed recursively for the sliding window. The execution times are similar to the execution times of the perception and end-to-end control LECs and much smaller than the corresponding sampling time (50 ms in AEBS , 100 ms in SDEC and 1000/7 ms in AVSD), and thus, the methods can be used for real-time OOD detection.



(a) In-distribution image

(b) Saliency map (I-GOS, In-distribution)

(c) Saliency map (VBP, In-distribution)

(d) OOD image

(e) Saliency map (I-GOS, OOD)

(f) Saliency map (VBP, OOD)

Figure 3.15: In-distribution and OOD images and their saliency maps for AVSD.

Table 3.3: False alarms and average delay in AVSD.

NCM	Parameters $(N,\omega,\tau)/(N,\tau)$	False positive	False negative	Average delay (frames)
VAE	$5,3,36$	$0/50$	$0/50$	11.32
	$5,5,15$	$0/50$	$1/50$	13.63
	$10,5,105$	$0/50$	$0/50$	12.14
	$10,8,55$	$0/50$	$0/50$	10.78
	$20,10,235$	$0/50$	$0/50$	12.18
	$20,17,140$	$0/50$	$0/50$	11.30
VAE, I-GOS	$5,3,36$	$0/50$	$0/50$	13.42
	$5,5,15$	$0/50$	$0/50$	16.10
	$10,5,105$	$0/50$	$0/50$	15.08
	$10,8,55$	$0/50$	$0/50$	12.74
	$20,10,235$	$0/50$	$0/50$	15.32
	$20,17,140$	$0/50$	$0/50$	14.12
SVDD	$10,4$	$0/50$	$0/50$	10.62
	$10,5$	$0/50$	$0/50$	12.58
	$15,4$	$0/50$	$0/50$	11.24
	$15,6$	$0/50$	$0/50$	12.68
	$20,5$	$0/50$	$0/50$	10.14
	$20,6$	$0/50$	$0/50$	11.66
SVDD, VBP	$10,4$	$0/50$	$0/50$	11.46
	$10,5$	$0/50$	$0/50$	13.32
	$15,4$	$0/50$	$0/50$	11.34
	$15,6$	$0/50$	$0/50$	12.78
	$20,5$	$0/50$	$0/50$	10.74
	$20,6$	$0/50$	$0/50$	12.32

3.7 Conclusion

In this chapter, we demonstrated a method for OOD detection in learning-enabled CPS. The method is based on inductive conformal prediction and anomaly detection but uses VAEs and deep SVDD to learn models to efficiently compute the nonconformity of new inputs relative to the training set and enable real-time detection of high-dimensional OOD inputs. In addition, the saliency maps can be incorporated to improve the robustness of the detector. Our evaluation is based on two simulation case studies of an AEBS and an SDEC as well as a real-world dataset for autonomous driving . The results demonstrate a very small number of false positives and detection delay while the execution time is comparable to the execution time of the original LECs.

Table 3.4: Missed alarms and average delay in AVSD.

NCM	Parameters	Missed alarms	Average delay
	$(N, \omega, \tau)/(N, \tau)$	(False negative)	(frames)
VAE	$5, 4, 24$	$1/50$	12.48
	$10, 6, 90$	$0/50$	11.68
	$20, 15, 180$	$0/50$	12.02
VAE, I-GOS	$5, 4, 24$	$0/50$	15.04
	$10, 6, 92$	$0/50$	14.38
	$20, 15, 175$	$0/50$	14.68
SVDD	$10, 6$	$1/50$	13.21
	$15, 5$	$0/50$	11.74
	$20, 7$	$0/50$	12.10
SVDD, VBP	$10, 6$	$0/50$	13.75
	$15, 5$	$0/50$	12.22
	$20, 7$	$0/50$	12.84

Table 3.5: Execution times of VAE- and SVDD-based detection methods.

	N	min (ms)	Q_1 (ms)	Q_2 (ms)	Q_3 (ms)	max (ms)
AEBS	N/A	3.48	3.85	3.91	3.96	4.20
SDEC	N/A	2.20	2.37	2.45	2.36	3.31
RPNN	N/A	0.51	0.52	0.52	0.53	0.56
I-GOS	N/A	4.32	4.37	4.41	4.43	4.47
VBP	N/A	1.37	1.38	1.38	1.39	1.41
VAE	5	15.43	15.47	15.49	15.50	15.60
	10	31.33	31.41	31.43	31.45	31.77
	20	64.54	64.70	64.72	64.75	65.00
VAE, I-GOS	5	20.18	20.21	20.24	20.29	20.32
	10	36.44	36.46	36.49	34.52	34.61
	20	70.54	70.59	70.64	70.71	70.84
SVDD	10	2.12	2.35	2.44	2.45	2.52
	15	2.16	2.23	2.34	2.41	2.49
	20	2.28	2.33	2.34	2.59	2.60
SVDD, VBP	10	3.43	3.49	3.50	3.65	3.70
	15	3.46	3.51	3.60	3.68	3.79
	20	3.53	3.57	3.69	3.70	3.82

Promising future work is to combine the outputs of the neural network into OOD detection. Another possible direction is to explore the physically realizable adaptive adversarial attacks and to adapt our approach to such attacks.

CHAPTER 4

Out-of-distribution Detection using Variational Autoencoder for Classification and Regression [1]

4.1 Introduction

Recently, machine learning techniques, such as Deep Neural Networks (DNNs), are extensively used in a broad range of domains since they can tackle complex tasks that conventional techniques cannot easily solve. On the other hand, Cyber-Physical Systems (CPSs) are generally deployed in environments with high uncertainty and variability, which requires a high level of autonomy. It is not surprising that CPSs increasingly employ Learning-Enabled Components (LECs) to perform different complex tasks [109]. Although LECs have achieved remarkable performance, their safety and reliability should be analyzed before deploying them to real-world systems, especially safety-critical systems. Unfortunately, the characteristics and complexity of the LECs complicate such analysis. Learning techniques, such as supervised and reinforcement learning, are typically used to train LECs. Such learning techniques are built upon an underlying assumption that the training and test distribution are similar. However, even if an LEC is trained extensively, Out-Of-Distribution (OOD) data are inevitably present when the LEC is used in the real world. OOD data may lead the LEC to be ineffective and incur erroneous predictions, which may undermine the safety of the system. Therefore, runtime OOD detection is very significant and necessary to guarantee the safety and reliability of the system. The objective of OOD detection is to quantify the degree of difference between the new test instances and the training data, and raise false alarms indicating the LEC may compute a large-error output due to the OOD data.

Although many efforts exist for OOD detection in neural networks [7], different types of OOD data are not investigated systematically. The first contribution of this chapter is the definitions for different types of OOD data present in learning-enabled CPS. We first discuss the cause of OOD examples and then categorize them into four different types: OOD data caused by (1) covariate shift, (2) target shift, (3) concept shift, and (4) label concept shift. We also provide typical examples for each type of OOD data aiming at classification and regression tasks. The categorization for the OOD data is based on the categorization of the *dataset shift* for the training and test distributions [110]. Note that dataset shifts focus on differences between the distributions of the training dataset and test dataset, while OOD detection aims at comparing a single test example with the distribution of the training dataset.

The main contribution of the chapter is an approach for detecting a variety of OOD data in learning-

[1] This chapter is adapted with permission from [Detection of dataset shifts in learning-enabled cyber-physcial systems," in *4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, May 2021.]

enabled CPS. Typical OOD detection techniques may result in a large number of false alarms due to the dynamical nature of CPS. In Chapter 3, a method that aims to improve the robustness of detection by using multiple examples sampled from a Variational AutoEncoder (VAE) model is proposed. The method is based on Inductive Conformal Anomaly Detection (ICAD) [63], and it is efficient so it can be used online. The chapter follows a similar approach but utilizes VAE for classification and regression models. The main benefit of such models is that they take into account both the input and output of the LEC, which enables the detection of different types of OOD data present in CPS.

Another contribution of the chapter is the comprehensive evaluation using several datasets for classification and regression tasks. We design experiments for various types of OOD data and use the same model for OOD detection. The experimental results show the proposed approach can detect different types of OOD data with a very small number of false alarms. The execution time is comparable with the sampling period of the typical CPSs, which enables real-time detection.

The outline of this chapter is as follows. Section 4.2 formulates the problem of detection of OOD data and discusses different types of OOD data in learning-enabled CPSs. Section 4.3 introduces the VAE for classification and regression model and presents the detection algorithm based on this model. Section 4.4 shows the evaluation results, and Section 4.5 concludes the chapter.

4.2 Out-of-distribution Data in Learning-enabled Cyber-physical Systems

4.2.1 Formal Definition of Out-of-distribution Data

The chapter focuses on the detection of a variety of OOD data in learning-enabled CPSs. We first analyze the causes of different OOD data and categorize them based on the underlying dataset shifts.

4.2.1.1 OOD data caused by covariate shift

Covariate shift is one of the most basic and common dataset shifts observed in real-world problems [111]. Suppose that an LEC f is trained with the dataset $\mathcal{D}_{\text{train}}$. A typical assumption is that inputs x are IID drawn from $P_{\text{train}}(x)$. The LEC can make predictions for some y based on the conditional probability $P(y|x)$ given x. Covariate shift occurs when the distribution of the input, $P(x)$ changes after training but the conditional probability $P(y|x)$ remains the same. Generalizing this definition, the OOD data caused by covariate shift can be defined as the data where the input variable x is not sampled from the same distribution of the training dataset $P_{\text{train}}(x)$, while the underlying relationship between input and output $P(y|x)$ remains unchanged.

4.2.1.2 OOD data caused by label shift

Label shift describes the case where the distribution over output variable $P(y)$ changes but the output-conditional distribution, $P(x|y)$ remains unchanged. OOD data caused by label shift can be defined as the data where the output variable y is not sampled from the training distribution $P_{\text{train}}(y)$ whereas the conditional probability of x given y remains the same, i.e. $P_{\text{train}}(x|y) = P_{\text{test}}(x|y)$.

4.2.1.3 OOD data caused by concept shift

A concept shift is simply a contextual shift where the underlying relationship between input and output changes while the distribution over the input is preserved [112]. Using the definition, for the OOD data caused by concept shift, we assume that the input variable x is from the same distribution as the training dataset $P_{\text{train}}(x)$ while the relationship between input and output changes, i.e. $P_{\text{train}}(y|x) \neq P_{\text{test}}(y|x)$.

4.2.1.4 OOD data caused by label concept shift

For the case of label concept shift, the distribution over the output stays the same, but the conditional probability of x given y changes. OOD data caused by label concept shift can be defined as the data where the output variable y is sampled from the same distribution as the training dataset $P_{\text{train}}(y)$, whereas the conditional probability of x given y changes such that $P_{\text{train}}(x|y) \neq P_{\text{test}}(x|y)$.

4.2.2 Examples of OOD Data

4.2.2.1 Classification

Consider the well-known digit recognition problem for the MNIST dataset [113]. The classification model is trained on the MNIST dataset, which only contains black and white handwritten digits. However, if a colorful handwritten digit or a handwritten digit with a different background is used as a test input, a classification model is very likely to make erroneous predictions. In this case, the test images are not from the same distribution as the training dataset. However, the classification results should be independent of the color or the background of the digits, and therefore the underlying relationship $P(y|x)$ should not change. Such test examples can be defined as OOD data caused by covariate shift. Further, the classification model can be influenced by OOD data caused by label shift, for example, when the probability distribution for the digit class $P_{\text{train}}(y)$ is not uniform or some classes of digits are not present in the training dataset.

OOD data caused by label concept shift arise in fault diagnosis and identification, where a classification model is used to predict the type of fault based on sensor measurements. For example, consider the fault diagnosis model for a gearbox [114] which aims at classifying the type of damage that may occur. Typically, the model is trained using data obtained under specific load conditions and tested under similar conditions resulting in satisfying accuracy. However, if the model is tested under a higher load condition, the performance will be degraded. In this case, although damage types in the test examples are still the same, the underlying relationship $P(x|y)$ changes due to additional load.

4.2.2.2 Regression

Covariate shifts occur in perception LECs used in autonomous vehicles. Consider, for example, an Advanced Emergency Braking System (AEBS) for an automobile that is designed to detect obstacles in Chapter 3. In this case, the perception LEC performs regression, and its performance can be degraded in the case of OOD data caused by covariate shift which arises when the environmental conditions for the test data are different from conditions considered during training. Such components may also be susceptible to OOD data caused by label shift. Similar to the classification problem, it is typically assumed that the probability distribution of the output, e.g., distance to the obstacle, $P_{\text{train}}(y)$ is uniform. However, in real-life situations, unique traffic patterns may impose a distribution, $P_{\text{test}}(y)$ that does not match this assumption. Further, it is usually assumed that the vehicle types and conform to typical specifications (e.g., size and shape). However, such specifications may change, for example, in response to autonomous vehicle technologies and the regression model may not be able to correctly estimate the distance to a vehicle of type or size not used during training. In this case, the conditional probability $P(x|y)$ changes, and such data can be regarded as OOD data caused by label concept shift. Additional examples and datasets relating to applications in industrial informatics are evaluated in Section 4.4.

4.2.3 Problem Formulation

Consider an LEC $f : \mathcal{X} \to \mathcal{Y}$ that is well trained to perform classification or regression using a training dataset $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$, where each example pair (x_i, y_i) contains the input $x_i \in \mathcal{X}$ and corresponding label $y_i \in \mathcal{Y}$. During the system operation, the LEC receives a sequence of inputs $\{x'_1, \dots, x'_t, \dots\}$ one by one and predicts the targets $\{y'_1, \dots, y'_t, \dots\}$. Such models are deployed with the assumption that the training and test examples are drawn from the same distribution. However, when the test data pair (x'_t, y'_t) is not sampled from the same distribution as the training dataset, the LEC f can become ineffective, make erroneous predictions, and undermine the safety of the system. Therefore, it is crucial to compute a measure quantifying the degree to which OOD data are present in the input sequence. OOD detection should consider all the various types of OOD data that may be present. Further, online detection algorithms must be robust with a small number of false alarms and computationally efficient so they can be executed in real time.

4.3 Detection of Out-of-distribution Data in Learning-enabled Cyber-physical Systems

4.3.1 VAE for Classification and Regression

A Variational AutoEncoder (VAE) is a generative model trying to learn an underlying probability distribution over the high-dimensional input points. Comparing with an autoencoder, a VAE models a relationship between the high-dimensional input data point and a low-dimensional latent representation in a probabilistic manner, which improves the capacity of generating new data [100]. A VAE for regression model, which aims to learn a conditional latent representation on a specific regression target variable, is presented in [115]. The model can be adapted for classification by conditioning the latent representation on a classification target variable. Figure 4.1 shows the architecture of the VAE for classification and regression models. The predictor module in the figure can be a standard classification or regression network.

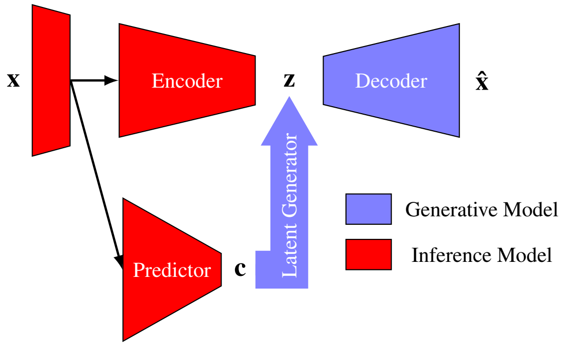


Figure 4.1: Variational autoencoder for classification and regression model. Note that ”Predictor” can be a classification or regression network.

The core idea of the model is to condition the latent encodings z on the target variable c inferred by the predictor, which performs classification or regression. Therefore, the latent distribution can be represented by a conditional Gaussian distribution $p(z|c)$ in lieu of $p(z)$. Specifically, compared with the VAE, there are two additional components: the *predictor* and the *latent generator*. The predictor employs a classification or regression network $q(c|x)$ to infer the target variable c, and the latent generator feeds the target variable c into the latent space conditioning the latent distribution on the predicted variable. In the case of classification, the target variable c is the one-hot vector of the predicted label, and in the case of regression, c is the regression output.

The VAE for classification and regression model can be trained in two phases: the *prediction* phase and the *VAE* phase. In the prediction phase, the predictor is trained to perform the prediction task regularizing the distribution of the prediction variable c with the ground-truth prior $p(c)$. The parameters of the predictor network are fixed after the prediction phase. Then, in the VAE phase, the encoder, decoder, and latent generator are jointly trained using the loss function

$$\begin{aligned}\mathcal{L}(\theta,\phi_c,\phi_z;x) =&\mathbb{E}_{z\sim q_{\phi_z}(z|x)}[\log p_\theta(x|z)]\\ &-\mathbb{E}_{c\sim q_{\phi_c}(c|x)}[D_{\text{KL}}(q_{\phi_z}(z|x)||p(z|c))].\end{aligned} \tag{4.1}$$

Similar to a traditional VAE, the first term is used to enable the decoder to reconstruct the input from the latent representation with a small reconstruction error. The second term aims to minimize the Kulback-Leibler (KL) divergence between the approximate posterior and the prediction specific prior $p(z|c)$.

In practice, the balance between these two terms should be carefully tuned to control the trade-off between the fidelity of reconstruction and quality of samples from the model [116]. Recently, a calibrated decoder architecture called σ-VAE, which can automatically tune the trade-off and improve the quality of the generated samples, is developed in [116], The idea of σ-VAE is to add a weighting parameter σ between the reconstruction term and KL-divergence term. The parameter σ can be computed analytically and does not require manual tuning. Implementation details can be found in [116]. This technique can also be used with the proposed VAE for classification and regression models in a similar fashion by adding the weighting parameter σ in the reconstruction term in Eq. (4.1).

4.3.2 Inductive Out-of-distribution Detection

Our approach is based on Inductive Conformal Anomaly Detection (ICAD), which requires a suitable *Non-Conformity Measure* (NCM) defined to quantify how different a test example is relative to the training dataset [63]. The VAE for classification and regression models are used to define the NCM. There are two significant benefits of using such models. First, the approach can scale up to the high-dimensional inputs and second, the models encode both input and output variables of the regression or classification tasks into the latent representations, and consequently, can be used to detect different types of OOD data.

4.3.2.1 Nonconformity measures

A test example x and its predictive label y' are encoded as z in the latent space of the VAE for classification and regression model, and subsequently, the decoder portion generates a reconstructed example $\hat{x}$ by sampling. If the input-output pair (x,y') is sampled from the same joint distribution of the training dataset $P_{\text{train}}(x,y)$, the test example x should be reconstructed with a relatively small reconstruction error. Therefore, the reconstruction error, or the squared error between the test input x and its reconstructed example $\hat{x}$, can be used as an NCM

$$A_{\text{RC}}(x) = ||x-\hat{x}||^2. \tag{4.2}$$

It is possible that some input features have rare or no contribution to the LEC prediction. Take an image input as an example, the reconstruction-based NCM will result in a large nonconformity score when the generative model has difficulty generating fine-granularity details of the original input. Therefore, the input features should be treated differently based on their influence on the LEC output. *Layer-wise Relevance Propagation* (LRP) is typically used as a tool for interpreting neural networks by identifying which input features contribute most to the LEC predictions [95]. Considering an input x, by running a backward pass in the predictor portion of the VAE for classification and regression model, LRP computes a relevance r, which has the same size as the input x. In order to deal with the problem that different inputs will have different total contributions, the relevance should be normalized by the sum of contributions for all features in the input. Let's define a function $r = G(x)$ to represent the LRP algorithm computing a relevance map r for a given input x. The NCM with LRP is computed by weighting the reconstruction error using the relevance map r

$$A_{\text{RC-LRP}}(x) = ||r \cdot (x - \hat{x})||^2. \tag{4.3}$$

An important property introduced by the VAE for classification and regression models is that the latent representations are disentangled by the target variable since the prior is conditioned on the target variable. Specifically, in the classification problem, the representations will be clustered by the target class. Therefore, for a test example x and its predictive label y', the distance of the representation z to its corresponding class center $c_{y'}$ can be defined as the distance-based NCM

$$A_{\text{dist}}(x) = ||x - c_{y'}||^2, \tag{4.4}$$

where the center $c_{y'}$ can be computed as the mean of the representations of the training data with the class label y'. Note that such distance-based NCM cannot be used for regression since the target variable is continuous.

4.3.2.2 Detection method

The method is divided into offline and online phases. During the offline phase, the training dataset $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$ is split into a proper training dataset $\mathcal{D}_{\text{proper}} = \{(x_i, y_i)\}_{i=1}^{m}$ and a calibration dataset $\mathcal{D}_{\text{calibration}} = \{(x_i, y_i)\}_{i=m+1}^{l}$. Then, we train a VAE for classification and regression model using the proper training set $\mathcal{D}_{\text{proper}}$. For each example $x_j : j \in \{m+1, \dots, l\}$ in the calibration set, the encoder portion of the model approximates the posterior distribution of the latent space and samples a point z_j from it. The nonconformity score α_j^{Γ} of this example can be computed by the NCMs defined earlier (Eq. (4.2), (4.3), and (4.4)). Specifically, for the reconstruction-based NCMs A_{RC} and $A_{\text{RC-LRP}}$, the sampled point z_j is used to reconstruct the input; for the distance-based NCMs A_{dist}, the sampled point z_j is directly used to compute the distance to the cluster center. The precomputed nonconformity scores for data in the calibration set are sorted as $\{\alpha_j\}_{j=m+1}^{l}$ for online detection.

During oneline detection, a sequence of test inputs $(x'_1, \dots, x'_t, \dots)$ arrive to the LEC one by one. Based on the approach in Section 3.3, multiple examples are incorporated to improve the robustness of the detection. For each test input x'_t, N points $\{z'_{t,1}, \dots, z'_{t,N}\}$ are sampled from the learned posterior distribution in the latent space. Then, for each generated point $z'_{t,k}$, the nonconformity score $\alpha'_{t,k}$ can be computed using the same NCM A used for calibration data. The p-value $p_{t,k}$ can be computed as the ratio of calibration nonconformity scores that are at least as large as $\alpha'_{t,k}$,

$$p_{t,k} = \frac{|\{i = m+1, \dots, l\} \,|\, \alpha_i \geq \alpha'_{t,k}|}{l-m}.$$

If the test data x'_t is sampled from a distribution similar to the distribution of the training dataset, most of the values in this p-value set $\{p_{t,k}\}_{k=1}^{N}$ should be relatively larger than 0. However, if there are many small values in the p-value set, the test example x'_t is very likely to be OOD. A martingale can be used to test if there are many small p-values in the set [69]

$$M_t = \int_0^1 M_t^{\varepsilon} d\varepsilon = \int_0^1 \prod_{k=1}^{N} \varepsilon p_{t,k}^{\varepsilon-1} d\varepsilon.$$

If the test example x'_t is OOD, the martingale value M_t will increase dramatically due to many small p-values in the set. Further, as described in Section 3.3.2, a stateful CUSUM detector S can be used to generate alarms when the martingale becomes consistently large. It should be noted that, if the input is not an example in a time sequence, the stateful CUSUM detector should be omitted, and the martingale value can be used directly. Algorithm 5 summarizes the proposed method.

4.4 Evaluation

In this section, we demonstrate the effectiveness of the approach using several datasets for classification and regression. All experiments presented in this chapter are conducted on a 6-core Ryzen 5 desktop with a single GTX 1080Ti GPU.

4.4.1 IoT Network Intrusion

4.4.1.1 Experimental setup

The number of Internet of Things (IoT) devices has increased dramatically, and IoT devices provide a large surface for intruders to deploy malicious cyber-attacks. Intrusion detection in IoT networks is very important

Algorithm 5 OOD data detection using VAE for classification and regression

```
Input: a training set $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$; a test sequence $(x'_1, \ldots, x'_t, \ldots)$; number of calibration examples $l - m$; number of examples $N$ sampled from the posterior; threshold $\tau$ and parameter $\omega$ of CUSUM detector
Output: boolean variable $Anom_t$
Offline:
Split the training set $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$ into the proper training set $\mathcal{D}_{\text{proper}} = \{(x_i, y_i)\}_{i=1}^{m}$ and calibration set $\mathcal{D}_{\text{calibration}} = \{(x_i, y_i)\}_{i=m+1}^{l}$
Train a VAE for classification and regression $f$ using the proper training set $\mathcal{D}_{\text{proper}}$
for $j = m+1$ to $l$ do
    Sample $z_j$ using the trained model
    $\alpha_j = A(x_j)$
end for
Online:
for $t = 1, 2, \ldots$ do
    Compute the relevance map $r'_t = G(x'_t; f)$
    for $k = 1$ to $N$ do
        Generate $\hat{x}'_{t,k}$ using the trained model
        $\alpha'_{t,k} = A(x'_t)$
        $p_{t,k} = \frac{|\{i=m+1,\ldots,l\} \mid \alpha_i \geq \alpha'_{t,k}|}{l-m}$
    end for
    $M_t = \int_0^1 \prod_{k=1}^{N} \varepsilon p_{t,k}^{\varepsilon-1} d\varepsilon$
    if $t = 1$ then
        $S_t = 0$
    else
        $S_t = \max(0, S_{t-1} + M_{t-1} - \delta)$
    end if
    $Anom_t \leftarrow S_t > \tau$
end for
```

for mitigating such attacks. We use two IoT intrusion datasets, N-BaIoT [117] and IoTID20 [118], to evaluate the proposed approach. N-BaIoT is a multivariate sequential dataset collected from 9 commercial IoT devices using two of the most common IoT botnet families: BASHLITE and Mirai. Our experiments focus on the data collected on a WiFi video doorbell. The dataset has 115 features extracted from the network packets, including packet size, packet count, etc. IoTID20 is another IoT botnet dataset collected from 2 typical IoT devices – a smart speaker and a WiFi camera. The dataset has 76 features extracted from raw network packet files. Compared with N-BaIoT, IoTID20 has more types of IoT attacks, including 5 categories and 9 subcategories. However, the data in IoTID20 are not sequential. We use the entire IoTID20 dataset in our experiment. In the following, we design and conduct different experiments using these two datasets to validate the effectiveness of our approach for various types of OOD data.

4.4.1.2 Evaluation metrics

We use different evaluation metrics for non-sequential and sequential data. For non-sequential data, the Area Under Receiver Operating Characteristic (AUROC) curve is used to assess the detection performance. The ROC curve plots the true positive rate against the false positive rate by varying the detection threshold. The AUROC is a threshold-free metric and is considered as the evaluation metric for OOD detection. The worst value of AUROC is 0.5 yielded by an uninformative classifier with a random guess. The best value of AUROC is 1.0, implying that the nonconformity scores for all the OOD data are greater than the score for the in-distribution data. For sequential data, the number of false positives and false negatives are used to evaluate the performance. We run the detection algorithm against multiple in-distribution and OOD sequences. We consider in-distribution sequences as false positives if alarms are raised and OOD sequences as false negatives if no alarm is raised.

4.4.1.3 Novelty detection for unknown classes

Novelty detection for unknown classes is a representative example of OOD data caused by label shift, since the label variable y is not sampled from the distribution of the training dataset, but the output-conditional distribution $P(x|y)$ remains the same. In this experiment, the training dataset includes not only the normal data but also some types of intrusion data. The objective is to detect the unknown types of intrusion data. Specifically, for the experiment using the N-BaIoT dataset, the training dataset consists of the normal data and data under attack by the BASHLITE botnet, and the data under attack by Mirai are considered as the OOD data caused by label shift. In the IoTID20 dataset, normal data and two categories of intrusion data (DoS and Mirai) are included in the training dataset, and the rest two categories (MITM and Scan) are the unknown classes. We note that because this experiment is deliberately designed to evaluate the approach for detecting OOD data caused by label shift, there is no baseline to compare in the literature.

We train both VAE for classification and σ-VAE for classification models. The VAE architecture is similar to the autoencoder architecture in [117]. For the N-BaIoT dataset, we select 25 normal sequences, 25 sequences attacked by BASHLITE, and 50 sequences attacked by Mirai as the test sequences. We report the False positives and False negatives by considering different numbers of generated examples N, CUSUM detection parameters ω and τ, and learning models (VAE and σ-VAE for classification) in Table 4.1. We use the reconstruction-based NCMs and distance-based NCMs for both learning models. The results show that all four different methods can detect novelty for unknown classes with zero false alarms.

For the IoTID20 dataset, the AUROC is reported in Table 4.2 for different learning models and different NCMs. For reconstruction-based NCMs, the method using σ-VAE for classification has a larger AUROC than using VAE for classification showing the σ-VAE for classification model can improve the reconstruction quality, and further improve the detection performance. As for the distance-based NCMs, it is interesting that the method using σ-VAE for classification model has worse performance than one using VAE for classification. This is because that the disentanglement ability of σ-VAE for classification model is not as good as VAE for classification model since the KL divergence loss in σ VAE for classification model is greater than the one in VAE for classification model during training. Further, we also report the AUROC based on generating a single example from the latent space, and the evaluation results demonstrate the performance improvement by incorporating multiple examples.

4.4.1.4 Intrusion detection

In order to compare our approach with existing work, we consider intrusion detection without classifying the type of attack. A deep autoencoder is employed to detect malicious intrusions for the N-BaIoT dataset in [117]. In this experiment, only the normal data are used for training . The intrusion detection experiment can be viewed as a case for OOD data caused by label shift, where only the normal class is included in the training dataset. It should be noted that the VAE for classification model degrades to a VAE model because only the normal data is included in the training dataset. For the N-BaIoT dataset, we select 50 normal sequences, 25 sequences attacked by BASHLITE, and 25 sequences attacked by Mirai as the test sequences. We report the false positives and false negatives by considering different numbers of generated examples N, CUSUM detection parameters ω and τ, NCMs, and learning models in Table 4.1. From the results, both methods can also achieve the same zero false alarms for detecting intrusions as the deep autoencoder method in [117]. In practice, as more data are collected, more categories of labeled intrusion data will be present in the training dataset and a VAE for classification can be used .

Table 4.1: False alarms for detecting OOD data in N-BaIoT dataset.

Types	NCM	N, ω, τ	False positives	False negatives
Novelty detection	A_{RC}	$5, 2, 50$	$0/50$	$0/50$
		$10, 4, 40$	$0/50$	$0/50$
	$A_{\sigma,\text{RC}}$	$5, 2, 50$	$0/50$	$0/50$
		$10, 4, 40$	$0/50$	$0/50$
	A_{dist}	$5, 2, 50$	$0/50$	$0/50$
		$10, 4, 40$	$0/50$	$0/50$
	$A_{\sigma,\text{dist}}$	$5, 2, 50$	$0/50$	$0/50$
		$10, 4, 40$	$0/50$	$0/50$
Intrusion detection	A_{RC}	$5, 2, 50$	$0/50$	$0/50$
		$10, 4, 40$	$0/50$	$0/50$
	$A_{\sigma,\text{RC}}$	$5, 2, 50$	$0/50$	$0/50$
		$10, 4, 40$	$0/50$	$0/50$

For the IoTID20 dataset, we report the AUROC by considering different learning models in Table 4.2.

The results show that the performance of the methods using σ-VAE for classification is better than those using VAE for classification, reflecting again that the improvement in quality of generated examples can also enhance the detection performance.

Table 4.2: AURO for detecting OOD data in IoTID20 dataset.

Types	NCM	Single example	Multiple examples
Novelty detection	A_{RC}	0.797	0.810
	$A_{\sigma,\text{RC}}$	0.881	0.881
	A_{dist}	0.872	0.872
	$A_{\sigma,\text{dist}}$	0.803	0.801
Intrusion detection	A_{RC}	0.768	0.802
	$A_{\sigma,\text{RC}}$	0.825	0.835

4.4.2 Gearbox Fault Detection

We evaluate the performance of detecting OOD data caused by label concept shift using a gearbox fault detection dataset [114]. The objective is to classify the type of damage that may occur on a gearbox. The state of the gearbox is measured using accelerometers attached at various locations. The gearbox can operate under two different loading conditions (low- and high-load) at five different constant shaft speeds. Normal behavior and five fault types are simulated for each shaft speed and loading condition. For each case which is the combination of fault type, shaft speed, and load condition, 4 seconds of data are collected at a sampling rate of 66.67 kHz. In this experiment, we consider the output shaft vibration data. The dataset is divided into two main subsets: low-load and high-load. For each subset, regardless of the shaft speed, the dataset is aggregated with respect to the type of fault. The data is converted into the frequency domain using Short Time Fourier Transform (STFT). Our experiment uses the subset from the low-load (including all normal and fault data) as the training dataset, and a VAE for classification model is trained to perform fault classification and OOD data detection. The data from the high-load subset are regarded as the OOD data. The distribution over output $P(y)$ stays the same, but the conditional probability $P(x|y)$ changes. This experiment is designed to evaluate the approach for OOD detection caused by label concept shift.

There are 4 fully-connected layers with $450/300/200/150$ units in the encoder portion of VAE for classification model, and the decoder has the symmetric architecture of encoder. The evaluation results are shown in Table 4.3. The classification accuracy for In-Distribution (ID) data is much more higher than Out-Of-Distribution (OOD) data since the conditional probability changes. The results reveal that the method using A_{RC} does not have a promising performance as the method using A_{dist}.

Table 4.3: Classification accuracy and AUROC for detecting OOD data in gearbox dataset.

NCM	Single example	Multiple examples	Accuracy(ID/OOD)
A_{RC}	0.573	0.581	99.1% / 60.5%
A_{dist}	0.690	0.698	

4.4.3 MNIST Dataset

In this subsection, we evaluate our approach on the well-known digit recognition dataset MNIST [113], to demonstrate effectiveness of our approach on an image dataset. First, we train a VAE for classification model on MNIST dataset to evaluate the approach in detecting OOD data caused by covariate shift. The encoder and latent generator portions of the model are listed in the Table 4.4. The classifier has a similar architecture with the encoder but the last layer is a fully-connected layer with 10 units, and the decoder is mirrored from the encoder. For the test data, the colorful MNIST [119] and SVHN [120] are used as the OOD dataset. These two datasets have the same labels of ten digits as the MNIST dataset, however, the inputs are from different distributions: for colorful MNIST, the inputs are the MNIST images synthesized with colorful backgrounds; for SVHN, the inputs are the digit images from street view house numbers. The data from these two datasets can be regarded as the OOD data caused by covariate shift since the input variable is sampled from a distribution different than the training dataset, but the conditional probability of y given x stays the same. We report the AUROC of the detection task in Table 4.5 using different NCMs. The AUROC is almost close to 1.0, which demonstrates that the approach can detect the OOD data caused by covariate shift.

Table 4.4: VAE for classification architecture in MNIST.

Encoder	Conv 32 × (4 × 4) (stride 2), Batch norm, LeakyReLU
	Conv 64 × (4 × 4) (stride 2), Batch norm, LeakyReLU
	Conv 128 × (4 × 4) (stride 2), Batch norm, LeakyReLU
	Conv 256 × (4 × 4) (stride 2), Batch norm, LeakyReLU
	FC 512, Batch norm, LeakyReLU
	FC 20 (mean) || FC 20, Sigmoid (Std. dev.)
Latent generator	FC 20

We also evaluate novelty detection for unknown classes using the MNIST dataset, which is a typical example of OOD data caused by label shift. In our experiment, following the experimental settings in [121], we randomly sample 6 classes in MNIST as known classes, and the rest 4 classes are unknowns. The training dataset only contains the 6 known classes, but the test dataset contains all 10 classes. The data from unknown classes can be viewed as the OOD data caused by label shift. The AUROC using different NCMs are reported in Table 4.5. The results also reveal that the reconstruction-based NCM using σ-VAE for classification has a better performance than the NCM using VAE for classification. The distance-based NCM using VAE for classificatin model has a comparable performance with the reconstruction-based NCM using σ-VAE for classification model. Although the performance of our approach is not as good as the state-of-the-art method [121](AUROC: 0.994), we use a more shallow neural network allowing for online detection.

Moreover, we also perform the ablation analysis by directly using the VAE-based method introduced in Chapter 3 to detect the OOD examples. The results of using VAE-based method are reported in Table 4.5 denoted by A_{VAE}. Although there is no difference between the methods using VAE and VAE for classification in the detection of OOD data caused by covariate shift, as for the OOD data caused by label shift, the novel method using VAE for classification obviously reveals better performance than VAE-based method. This is because the VAE for classification take into account not only the input but also the output.

Table 4.5: AUROC for detecting OOD data in MNIST dataset.

Types	NCM	Single example	Multiple examples
Covariate shift	A_{RC}	1.000	1.000
	$A_{\sigma,\text{RC}}$	1.000	1.000
	A_{VAE}	1.000	1.000
Label shift	A_{RC}	0.852	0.878
	$A_{\sigma,\text{RC}}$	0.879	0.879
	A_{dist}	0.874	0.870
	$A_{\sigma,\text{dist}}$	0.803	0.805
	A_{VAE}	0.704	0.723

4.4.4 Advanced Emergency Braking System

We also evaluate the approach using a perception LEC of the Advanced Emergency Braking System (AEBS), which attempts to predict the distance to the nearest front obstacle and apply an appropriate brake force to safely stop the host vehicle. We implement the AEBS in an open-source simulator for autonomous driving research – CARLA [103]. The perception LEC is a typical regression LEC, whose objective is to estimate the distance to the approaching obstacle using the raw images captured by an on-board camera. In order to collect the training dataset, we control the precipitation parameter, available in CARLA, which is randomly sampled between 0 to 20. We totally collect 19900 images, where $P(y)$ is nearly uniformly distributed between 0 m to 50 m so that the training dataset is almost balanced. Then, we randomly split the training dataset into a proper training set (15920 images) and a calibration set (3980 images). We plot the histogram of ground-truth distance for the training data set in Figure 4.2. The histogram shows that there is nearly equal amount of data corresponding to each interval of the distance range under consideration We should note that the probability distribution of the ground-truth distance, $P_{\text{train}}(y)$ is nearly uniformly distributed in the range $[0\,\text{m}, 50\,\text{m}]$ so that the training dataset is almost balanced.

We implement the VAE for regression model using a convolutional neural network, whose encoder and

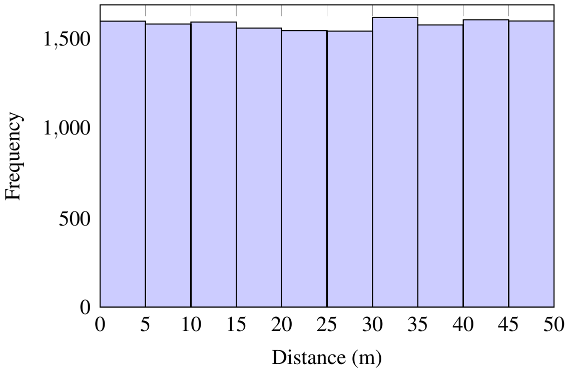


Figure 4.2: Histogram of ground-truth distance in training dataset.

latent generator modules are listed in Table 4.6. The regressor has the almost same architecture as encoder but two additional fully-connected layers with $256/1$ units. The decoder has symmetric architecture of encoder. The regressor is successfully trained with satisfying training and test errors after 250-epoch training. Additionally, we plot the low-dimensional representations of the latent encodings in in Figure 4.3 by applying t-distributed Stochastic Neighbor Embedding (t-SNE) [122]. The distance-related dimensions are well disentangled from the latent space by using VAE for regression model.

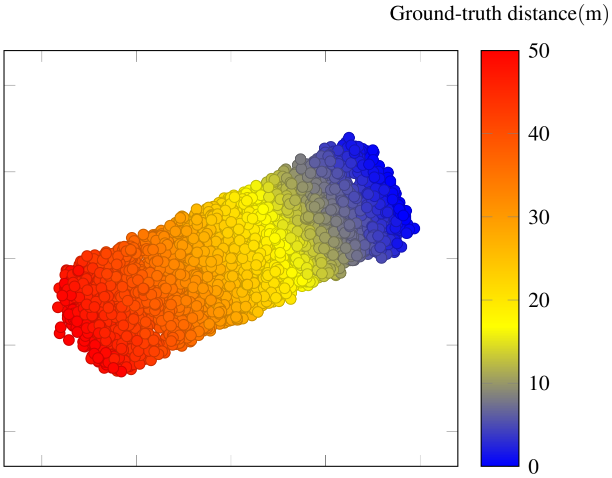


Figure 4.3: 2-dimensional latent representations estimated by VAE for regression.

The nonconformity scores for data in the calibration set are precomputed and sorted for the online detection. For each test example during the online phase, the VAE for regression model generates $N = 10$ examples used for detection. We illustrate our approach using an in-distribution episode firstly and plot the absolute prediction error between the ground-truth and predicted distance to the obstacle, the p-value, and the output of the detector S computed using the logarithm of martingale M_t and $\omega = 4$ in Figure 4.4. In this experiment,

Table 4.6: VAE for regression architecture in AEBS.

Encoder	Conv 32 × (5 × 5) (stride 2), Batch norm, ELU
	Conv 64 × (5 × 5) (stride 2), Batch norm, ELU
	Conv 128 × (5 × 5) (stride 2), Batch norm, ELU
	Conv 256 × (5 × 5) (stride 2), Batch norm, ELU
	FC 1568, Batch norm, ELU
	FC 1024 (mean) || FC 1024, Sigmoid (Std. dev.)
Latent generator	FC 256, Batch norm, ELU
	FC 1024

we use reconstruction-based nonconformity measures with and without LRP in our detection algorithm, and both results are plotted for comparing, which are denoted by (A_{RC}) and ($A_{\text{RC-LRP}}$) respectively in the plots.

The results of the in-distribution episode show that, for nonconformity measure A_{RC}, the p-values are almost much greater than 0, and thus, the detector stays in a low value indicating there is no OOD data presented in the sequence. As for the nonconformity measure $A_{\text{RC-LRP}}$ the p-values are far away from 0 at first, but decrease at the end of the episode. The reason for this phenomenon is, near the end of the episode, the lead vehicle occupies more pixels in the image, more pixels are very relevant to the LEC output, and the nonconformity scores are computed by taking more pixels into account. Although the p-values decrease a little, the detector S is still smaller than the threshold 40 indicating there is no OOD data presented during the episode.

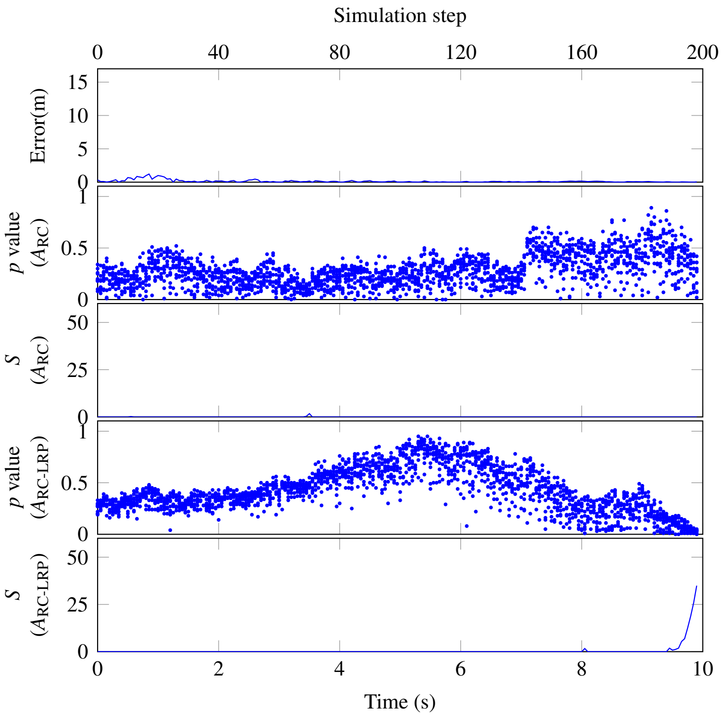


Figure 4.4: An episode with in-distribution data in AEBS (detector parameter: $N = 10$, $\omega = 4$, $\tau = 40$).

In AEBS, the precipitation parameter can be controlled to enforce the input of test examples different from the training dataset. Specifically, the precipitation parameter in the training dataset is randomly sampled from $[0,20]$; however in testing, the precipitation parameter is randomly sampled from $[30,100]$.

An OOD episode caused by covariate shift is shown in Figure 4.5. The error of the perception LEC is palpably larger than the error for in-distribution data and it can exceed 15 m. The p-values come down to almost 0 and the detector indicates the OOD data are present in the sequence. We evaluate the approach using the 50 in-distribution episodes and 50 OOD episodes caused by covariate shift. We report the false positives and false negatives for detection OOD data caused by covariate shift using two different NCMs (A_{RC} and $A_{\mathrm{RC\text{-}LRP}}$) in Table 4.7. From the results, we can see that the approach can detect OOD data caused by covariate shift with few false alarms.

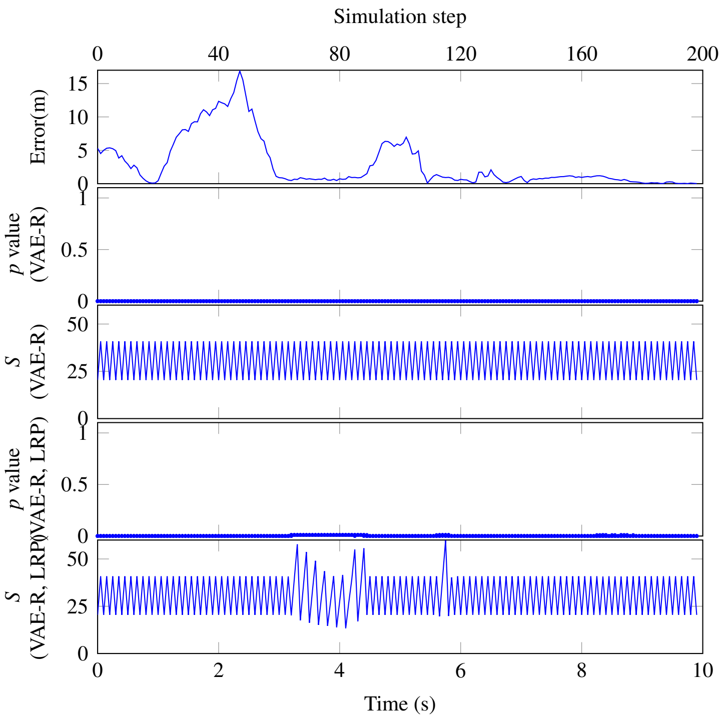


Figure 4.5: An episode with OOD data caused by covariate shift in AEBS (detector parameter: $N = 10$, $\omega = 4$, $\tau = 40$).

For OOD data caused by label shift, all data ranging from 15 m to 45 m are excluded from the training dataset, which is illustrated in Figure 4.6. We retrain the VAE for regression model and report the false alarms for detection such type of OOD data in Table 3.1, which demonstrate the effectiveness of the approach. The method using NCM with LRP has fewer false alarms than the method without LRP, which validates that the LRP algorithm can improve the robustness of the detection. An OOD episode caused by label shift is shown in Figure 4.7, which contains the data between 0 m to 50 m. In this case, the neural network does not work

Table 4.7: False alarms for detecting OOD data in AEBS.

Types	NCM	N, ω, τ	False positives	False negatives
Covariate shift	A_{RC}	5,2,50 10,4,40	0/50 0/50	0/50 0/50
	$A_{\text{RC-LRP}}$	5,2,50 10,4,40	4/50 3/50	0/50 0/50
Label shift	A_{RC}	5,2,50 10,4,40	N/A	13/50 11/50
	$A_{\text{RC-LRP}}$	5,2,50 10,4,40	N/A	3/50 2/50
Label concept shift	A_{RC}	5,2,50 10,4,40	0/50 0/50	0/50 0/50
	$A_{\text{RC-LRP}}$	5,2,50 10,4,40	4/50 3/50	0/50 0/50

properly, and the results show the neural network predicts the outputs with large errors. The p-values come to nearly zero and the detector grows up, which indicates the OOD data are present during the episode.

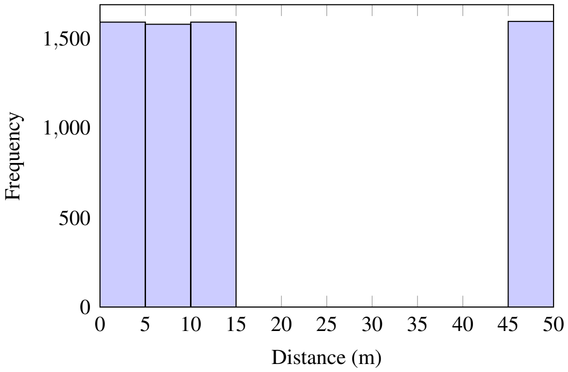


Figure 4.6: Histogram of ground-truth distance of training dataset that excludes data ranging from 15 m to 45 m.

We also evaluate the approach for detecting OOD data caused by label concept shift, and a different size of lead vehicle which never appeared is used in the training dataset. For testing, we collect 50 episodes doubling the size of the leading vehicle, and one episode is illustrated in Figure 4.8. It can be viewed as OOD data caused by label concept shift because the condition probability $P(x|y)$ changes. It is reasonable to see that the predicted distance is much smaller than the ground-truth distance at the beginning of the episode since the double-size car occupies more pixels in the image than the normal-size car. The p-value becomes small and the detector indicates the OOD data are present in the test episode. From the experiment, we observe that the method with LRP has a shorter detection delay compared to the method without LRP since the LRP algorithm makes the NCM focusing on features contributing to the LEC output. We report the false alarms for detecting such OOD data in Table 3.1.

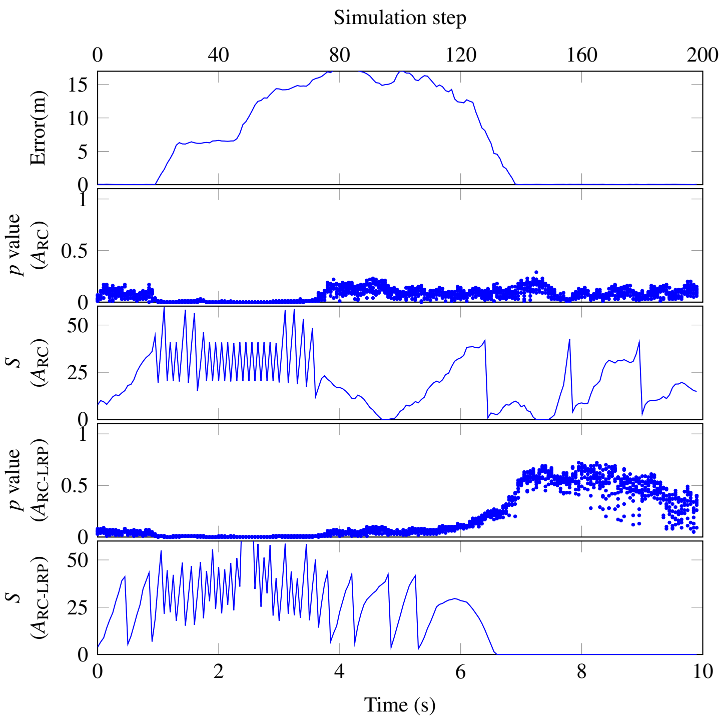


Figure 4.7: An episode with OOD data caused by label shift in AEBS (detector parameter: $N = 10$, $\omega = 4$, $\tau = 40$).

4.4.5 Autonomous Vehicle Seasonal Dataset

4.4.5.1 Experimental setup

We also evaluate our approach in a real-world Autonomous Vehicle Seasonal Dataset (AVSD) provided by Ford [104]. The dataset is collected under the seasonal variation in weather, lighting, construction, and traffic conditions experienced in dynamic urban environments [104], and provides the raw images captured by the on-board camera and ground-truth position and orientation of the host vehicle in a global frame. The perception task is to predict the heading changes of the host vehicle using the images captured by the camera. The dataset (V1) collected in cloudy weather and freeway, overpass, and bridge drive scenarios is selected as the training dataset. For using the dataset for training, we preprocess the data to compute heading changes of the vehicle by converting the quaternions to Euler angles and calculating the yaw difference and we synchronize the input images with the heading changes. Besides, the training dataset is randomly split into 3556 images for the proper training set and 889 for the calibration set. The proper training dataset is used to train the VAE for regression model. In the VAE for regression model, the regressor part is a CNN whose architecture is similar to the NVIDIA end-to-end self-driving controller [108], and the VAE and latent generator is the same as architectures we used in AEBS. The model is trained by $250(\eta = 10^{-4}) + 100(\eta = 10^{-5})$ epochs, and mean absolute error for training and testing are 0.012° and 0.016° respectively.

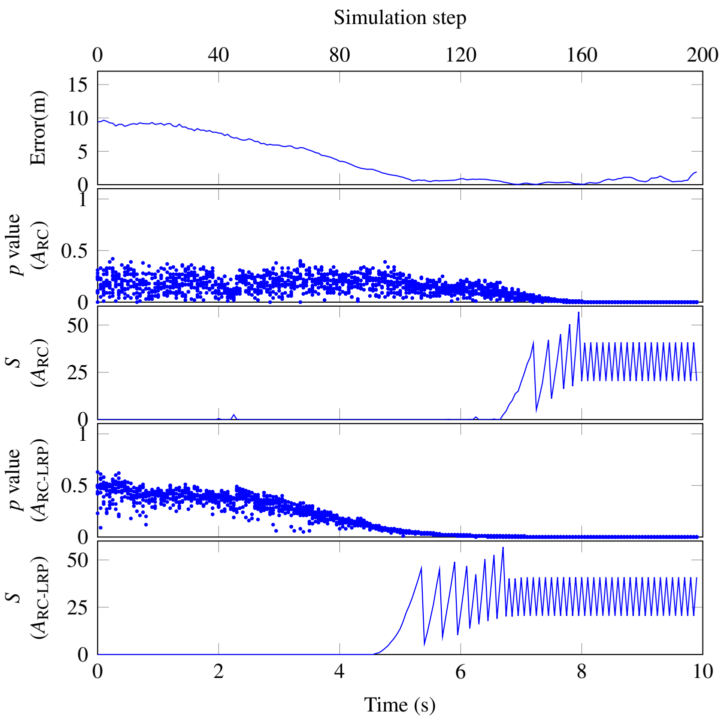


Figure 4.8: An episode with OOD data caused by label concept shift in AEBS (detector parameter: $N = 10$, $\omega = 4$, $\tau = 40$).

4.4.5.2 Experimental Results

For testing, we evaluate our approach using 50 episodes which are from the same distribution as the training dataset, and 50 episodes which are collected in sunny weather and residential driving scenario, which can be regarded as the OOD data caused by covariate shift. We illustrate our approach using two episodes and plot the prediction errors of the heading change, the p-values, and detector output of the reconstruction-nonconformity measures with and without the LRP algorithm. The results are shown in Figure 4.9 and Figure 4.10 respectively.

For the in-distribution episode, the prediction errors are very small, p-values are randomly distributed between 0 and 1, and the detector indicates there are no dataset shifts happening during the episode. In the covariate shift episode, the predicted errors are greater than the in-distribution episode. The p-values are small and the martingales grow very large showing the dataset shift happens in the test episode. We also report the number of false alarms by considering different values of N and detector parameters σ and τ in Table 4.8. From the results, the number of false alarms is zero.

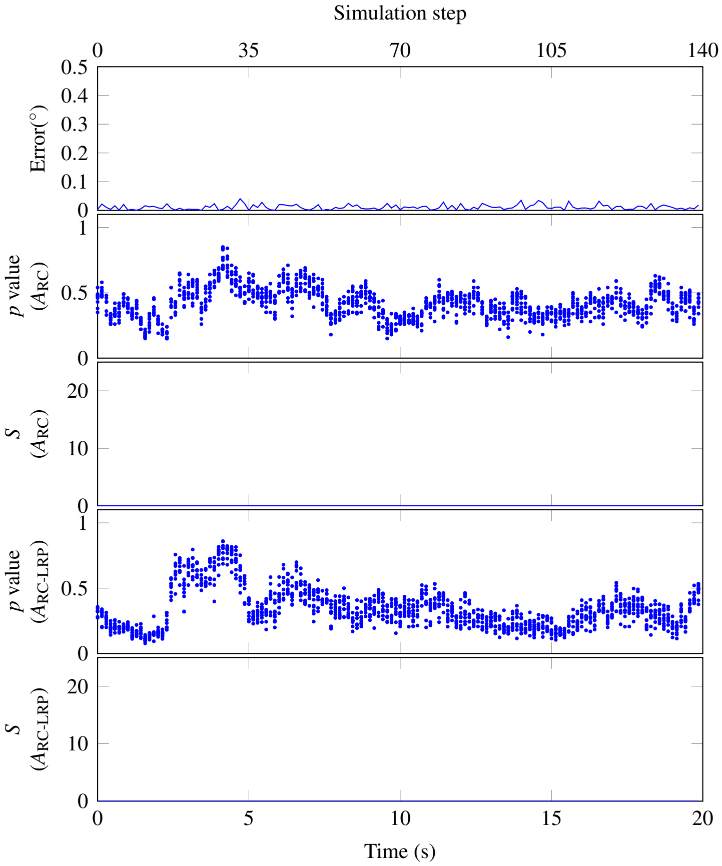


Figure 4.9: An episode with in-distribution data in AVSD (detector parameter: $N = 10$, $\omega = 10$, $\tau = 14$).

Table 4.8: False alarms for detecting OOD data caused by covariate shift in AVSD.

NCM	Parameters $(N,\omega,\tau)/(N,\tau)$	False positive	False negative
A_{RC}	$5,3,36$	$0/50$	$1/50$
	$10,10,14$	$0/50$	$0/50$
	$20,10,235$	$0/50$	$0/50$
$A_{\text{RC-LRP}}$	$5,3,36$	$0/50$	$1/50$
	$10,10,14$	$0/50$	$0/50$
	$20,10,235$	$0/50$	$0/50$

4.4.6 Computational Efficiency

In order to characterize the real-time nature of our detection approach, we consider the AEBS example and we measure the execution time in one episode using different NCMs. We plot the results in the boxplot in Fig 4.11. From the plot we can observe that the execution time of the method using LRP is slightly longer than that of the method without LRP due to the LRP computations. Further, the execution time is shorter than the sampling period of AEBS, 50 ms, and therefore the approach is appropriate for real-time OOD detection. Note that the number of the examples generated from the VAE for regression model is fixed at 10 when

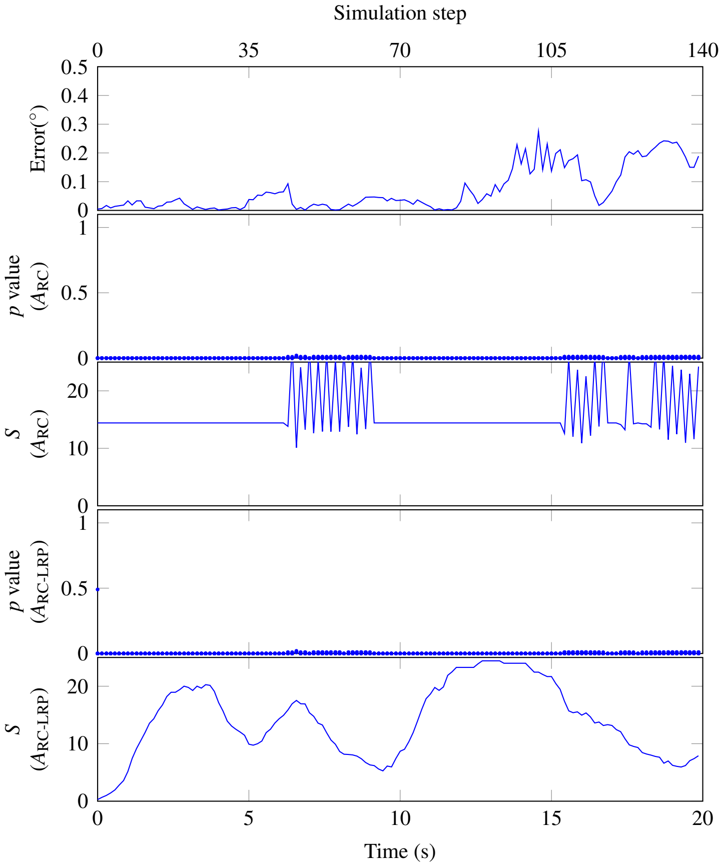


Figure 4.10: An episode with OOD data caused by covariate shift in AVSD (detector parameter: $N = 10$, $\omega = 10$, $\tau = 14$).

measuring these execution times. As the number of N increases, the execution time will also increase.

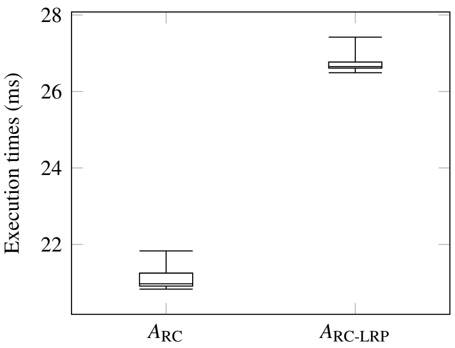


Figure 4.11: Execution times of proposed method.

4.5 Conclusion

In this chapter, we analyze the causes for the OOD data, and categorize them into four different types. Focusing on such OOD, we propose a detection approach based on inductive conformal anomaly detection which utilizes VAE for classification and regression models to compute the nonconformity measures. The main advantage of the proposed method is that it takes both inputs and outputs into consideration for detecting various types of OOD data. Multiple experiments are designed and conducted on different datasets for both classification and regression tasks. The experimental results demonstrate that the approach can detect different types of OOD data with small false alarms. Further, the execution time of the method is very small enabling real-time detection. A promising direction to improve the performance of the method is to incorporate attention mechanisms into VAE models.

CHAPTER 5

Out-of-distribution Detection using Adversarial Autoencoder [1]

5.1 Introduction

Over the past decade, machine learning components, such as Deep Neural Networks (DNNs), have made remarkable achievements, resulting in state-of-the-art performance in various tasks, especially in image classification systems [123; 124]. Nevertheless, there are still several challenges restricting the deployment of machine learning components to safety-critical real-world systems. Machine learning models are built upon an underlying assumption that the training and test data are sampled from the same distribution. In a real-world system, however, even if a machine learning component is well-trained over an extensive training dataset, Out-Of-Distribution (OOD) data are still inevitable during testing, and they may cause the model to make erroneous predictions and degrade the performance considerably. Hence, detection of OOD data is significant for the safety of machine learning components. When OOD data are fed into the predictive model, the detector can raise alarms for human intervention or redesign of the model.

Although there are many studies on OOD detection in machine learning components, especially for classification, the manifestations of OOD data are still unexplored. OOD detection methods in the literature [7; 125] attempt to determine whether an input example is from the same distribution as the training dataset, which only detects the change in the distribution of the input variable. Another research direction is novelty detection for unknown classes [86; 126]. The novelties from unknown classes can be regarded as another type of OOD data, where the change in the distribution of the output variable is also observed. A related research topic to OOD data is *dataset shift*, which occurs when the joint distribution of the input and output variables differ between the training and testing phases [110]. However, there is still a specific difference between dataset shift and OOD data: dataset shift focuses on two distributions – the distributions of the training dataset and test dataset; in contrast, the OOD data focuses on the distribution of the training dataset and a single test example. In Chapter 4, the OOD data are categorized into four types: OOD data caused by covariate shift, label shift, concept shift, and label concept shift. In this chapter, we follow these definitions and put special focus on the detection of OOD data in machine learning components used for classification.

In order to efficiently detect different types of OOD data in machine learning components, we propose the inductive conformal out-of-distribution detection, which is based on the Inductive Conformal Anomaly Detection (ICAD) framework [63]. The core of the ICAD method is the definition of a *nonconformity mea-*

[1] This chapter is adapted with permission from [F. Cai, A. Ozdagli, N. Potteiger, and X. Koutsoukos, “Inductive conformal out-of-distribution detection based on adversarial autoencoder,” in *IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, August 2021.]

sure, which is a function measuring the dissimilarity between a test example and the training dataset. Our approach utilizes a variant of an Adversarial Autoencoder (AAE) [127] to define the nonconformity measure, which can disentangle the label information from the latent representation by estimating a class variable in addition to the latent representation. By using such an architecture, the joint distribution of the input and output variables on the training dataset can be represented. Therefore, both the input and output of the machine learning component can be taken into consideration for OOD detection.

Moreover, the detection method using a single example may result in a large number of false alarms. The robustness of the detector can be improved by incorporating multiple examples into the detection algorithm as Chapter 3. Our method follows this idea and employs an AAE to generate multiple examples for robust detection. Although multiple examples are considered, our approach focuses on comparing a single test example with the training distribution nonetheless. We also design two different nonconformity measures, quantifying the degree to which the test example is not sampled from the same distribution as the training dataset. We conduct extensive experiments on several datasets to evaluate our approach. The results show that our approach can efficiently detect different types of OOD data and can be used for online detection.

The rest of this chapter is organized as follows: Section 5.2 formulates the problem of OOD detection in machine learning components for classification. Section 5.3 describes our proposed approach – inductive conformal OOD detection. Section 5.4 utilizes multiple datasets to demonstrate our detection method, and Section 5.5 provides the concluding remarks for this chapter.

5.2 Out-of-distribution Detection in Machine Learning Components for Classification

In this section, we formulate the OOD detection problem in machine learning components for classification. Since the categorization of the OOD data has already been introduced in Section 4.2.1, we will not repeat it here.

5.2.1 Problem Formulation

Consider a machine learning component f for a classification problem, which is well-trained using a set of labeled samples $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$, where each example (x_i, y_i) consists of the input $x_i \in \mathcal{X}$ and corresponding label $y_i \in \mathcal{Y}$, and it is sampled from a joint distribution $P_{\text{train}}(x, y)$. During the system operation, a test example x_{l+1} is consumed by the component to estimate a predictive class y'_{l+1}. The implicit assumption for the effectiveness of machine learning techniques is that the test example pair (x_{l+1}, y'_{l+1}) is sampled from the same joint distribution of the training dataset $P_{\text{train}(x,y)}$. However, the training dataset $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$ is necessarily incomplete, and therefore, OOD data are commonly present. The machine learning component f may become ineffective due to the OOD data and make predictions with large errors. In this case, it is desirable to raise alarms or retrain the model, and therefore, detection of OOD data is of importance for the safety of the machine learning component.

The detection must be performed efficiently and preferably online, which means that the execution time should be comparable to the execution time of the machine learning component. It is very challenging because machine learning components are increasingly used for tasks with high-dimensional data.

5.3 Adversarial Autoencoder and Out-of-distribution Detection

In this section, we introduce inductive conformal out-of-distribution detection, which is based on a variant of an Adversarial Autoencoder (AAE) and the Inductive Conformal Anomaly Detection (ICAD) framework.

5.3.1 Adversarial Autoencoder

An AAE is a generative model which is trained in an adversarial manner to force the aggregated posterior of the latent coding space of the autoencoder to match an arbitrary known distribution [127]. Specifically, assuming x is the input and z is the low-dimensional latent representation, a basic AAE model consists of an encoder (generator in adversarial network) $G(x)$ trying to encode the input into the low-dimensional latent representation, a decoder $De(z)$ trying to reconstruct the original input data from the encodings, and a discriminator $D(z)$ trying to identify the hidden samples z generated by the generator or sampled from the true prior. The whole architecture is trained jointly in two phases: the *reconstruction* phase and *regularization* phase. In the reconstruction phase, the encoder $G(x)$ and decoder $De(z)$ are updated to minimize the reconstruction error. In the regularization phase, the discriminator $D(z)$ is trained to distinguish the true samples (sampled from the prior distribution $p(z)$) from the generated samples (sampled from the posterior distribution $q(z|x)$), while the generator $G(x)$ is trained to deceive the discriminator $D(z)$ by outputting samples that closely resemble data sampled from the prior distribution $p(z)$.

In order to disentangle the label information from the latent representation, a class variable y can be predicted by the encoder $G(x)$ in addition to the latent variable z, and the one-hot vector of the predicted class is provided to the decoder $De(y,z)$ to generate class-conditioned output (Figure 4.1). This architecture can be regarded as a supervised variant of a semi-supervised AAE introduced in [127]. A *supervised classification* phase is performed after the reconstruction and regularization phases, whose objective is to minimize the cross-entropy cost between the target distribution $p(y)$ and the approximation of the target distribution $q(y|x)$.

5.3.2 Inductive Conformal Out-of-distribution Detection

The task of anomaly detection is to determine whether the test example conforms to the normal data. Inductive Conformal Anomaly Detection (ICAD) is an anomaly detection framework with the property of

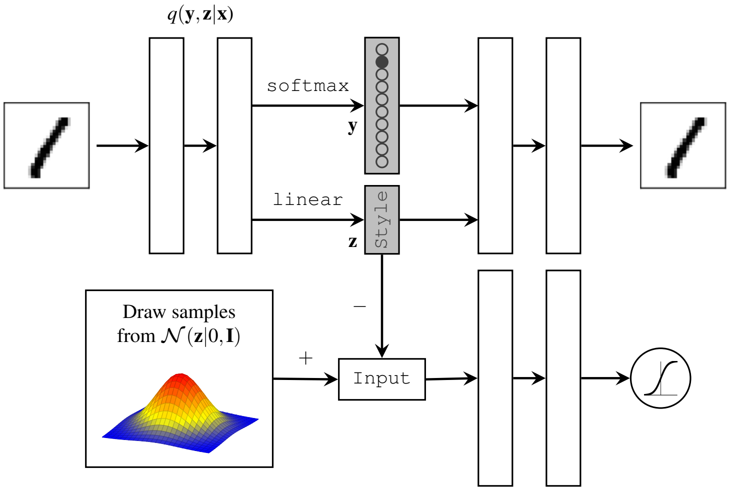

$q(\mathbf{y},\mathbf{z}|\mathbf{x})$
softmax
y
linear
Style
z
Draw samples from $\mathcal{N}(\mathbf{z}|0,\mathbf{I})$
−
+
Input

Figure 5.1: A variant of the adversarial autoencoder model.

well-calibrated false alarms [63]. The ICAD method is based on a definition of a *nonconformity measure*, which is a function measuring the dissimilarity between a test example and the training dataset. Recently, in order to enable online detection for high-dimensional data, learning models are trained to represent the distribution of the input variable of the training dataset and are utilized for the computation of a nonconformity measure [128]. However, nonconformity measures considering only the input variable may not be sufficient for the detection of some specific types of OOD data because the output variable can also lead the input and output pair out of the joint distribution of the training dataset. The variant of AAE can encode a label variable in addition to a latent variable, and consequently, both input and output can be represented in such a model. In the following, we first introduce two different nonconformity measures based on AAE. Subsequently, we describe the inductive conformal OOD detection method based on these nonconformity measures.

5.3.2.1 Nonconformity measures

For a test example x, the encoder portion of the AAE represents x and its predictive label y' in a latent space, and the decoder portion generates a new example $\hat{x}$ by sampling from the encodings. If x and its predictive label y' are from the same joint distribution of the training dataset, the example x should be reconstructed with a relatively small reconstruction error. Therefore, the reconstruction error between the input x and generated output $\hat{x}$ can be used as the reconstruction-based nonconformity measure A_{rc} defined as

$$A_{\text{rc}} = ||x - \hat{x}||^2. \tag{5.1}$$

The reconstruction-based nonconformity measure treats all features of the input equally. However, a relatively small part of the features in the input may have a significant effect on the final prediction. Therefore, it is not reasonable to treat all input features equally when they contribute to the output of the predictive model differently. A novel nonconformity measure based on *saliency maps* is introduced to compensate for such a defect. A saliency map algorithm aims to quantify the contributions of the input features to the predictive result of a machine learning model [92]. Specifically, we utilize the gradient-based saliency map algorithm [92]. It generates the saliency map by computing the derivative of the SoftMax score S_y of class y with respect to the input x at a given point x_0

$$w = \frac{\partial S_y}{\partial x}|_{x_0}.$$

The derivative w reflects the influence of input features on the final prediction and is used to define the saliency-based nonconformity measure by weighting the reconstruction error as

$$A_{\text{saliency}} = ||w \cdot (x - \hat{x})||^2. \tag{5.2}$$

The saliency map w is computed using the portion used for the classification task in the encoder of the AAE, and the reconstructed output $\hat{x}$ is also generated by the AAE.

5.3.2.2 Detection method

Given a test input x_{l+1} and its predictive label y'_{l+1}, the OOD detection method aims to determine whether the test input-prediction pair (x_{l+1}, y'_{l+1}) is sampled from the same joint distribution of the training dataset $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$. The proposed method is based on the framework of ICAD, and therefore, the detection algorithm is divided into offline and online phases. During the offline phase, the training dataset $\mathcal{D}_{\text{train}}$ is split into two sets: a proper training set $\mathcal{D}_{\text{proper}} = \{(x_i, y_i)\}_{i=1}^{m}$ and a calibration set $\mathcal{D}_{\text{calibration}} = \{(x_i, y_i)\}_{i=m+1}^{l}$. An AAE F is trained over a proper training set $\mathcal{D}_{\text{proper}}$ for the computation of nonconformity measures. Let A be either nonconformity measure function defined before. After that, for each data $x_j, j = m+1, \ldots, l$ in the calibration set, a new example $\hat{x}_j$ is generated using the trained AAE F, and its corresponding nonconformity score α_j^{Γ} is computed according to the nonconformity measure A. In order to reduce the time complexity of the p-value computation during the online phase, nonconformity scores of the calibration data are sorted and stored as $\{\alpha_j\}_{j=m+1}^{l}$.

At the online detection stage, given the test example x_{l+1}, in order to improve the robustness of the detection, N examples $\{\hat{x}_{l+1,k}\}_{k=1}^{N}$ are generated from the AAE. For each generated example $\hat{x}_{l+1,k}$, its nonconformity score $\alpha_{l+1,k}$ can be computed using the same nonconformity measure A as the calibration set.

Subsequently, two different techniques can be applied to aggregate these N nonconformity scores for detection.

One option is to compute the expected nonconformity score $\bar{\alpha}_{l+1}$ of N nonconformity scores, and the p-value p_{l+1} can be computed as the ratio of calibration nonconformity scores that are at least as large as $\bar{\alpha}_{l+1}$:

$$p_{l+1} = \frac{|\{i = m+1, \ldots, l\} \,|\, \alpha_i \geq \bar{\alpha}_{l+1}|}{l-m}. \tag{5.3}$$

A smaller p-value reflects an unusual test example with respect to the training examples. If the p-value p_{l+1} is smaller than a threshold ε, this test example will be classified as an OOD instance.

Additionally, we can use a martingale test [69; 128] for N nonconformity scores to detect OOD data. For each nonconformity score of a generated example $\alpha_{l+1,k}$, the corresponding p-value $p_{l+1,k}$ is calculated using Eq. (5.3). Then, a simple mixture martingale [69] is applied, which is defined as

$$M_{l+1} = \int_0^1 \prod_{k=1}^{N} \varepsilon p_{l+1,k}^{\varepsilon-1} d\varepsilon. \tag{5.4}$$

Such martingale value will grow only if there are many small p-values in $\{p_{l+1,k}\}_{k=1}^{N}$, and the detector will raise an alarm when the martingale value M_{l+1} is greater than a predefined threshold τ. The martingale test is expected to have a better performance than the expected p-value of nonconformity scores since it can enlarge the nonconformity gap between in-distribution and OOD instances.

The whole procedure of the detection algorithm is summarized in Algorithm 6.

5.4 Evaluation

To demonstrate the effectiveness of the proposed approach, we conduct extensive experiments for the detection of different types of OOD data using several datasets. In this section, we describe the implementation details first. Then, we describe the experimental setup and present evaluation results for three different types of OOD. Finally, we measure and report the execution time of the proposed method.

5.4.1 Experiment Implementation

5.4.1.1 Neural network architecture

The AAE is trained to perform both classification and detection tasks. For different types of inputs, we use different architectures of the AAE. For the image input (Experiment 1-2), in order to allow for online detection, we implement the AAE with a relatively shallow convolutional network: the encoder contains three convolutional layers and one fully connected layer. The decoder has symmetric one fully connected layer and three deconvolutional layers. Furthermore, three fully connected layers form the discriminator. For

Algorithm 6 Inductive Conformal Out-of-distribution Detection using Adversarial Autoencoders

```
Require: Input training set $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$; number of calibration examples $m$; number of examples $N$ generated by the adversarial autoencoder; test example $x_{l+1}$; threshold $\varepsilon$ of $p$-value of expected nonconformity score, or threshold $\tau$ of martingale value
Offline:
Split the training set $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{l}$ into the proper training set $\mathcal{D}_{\text{proper}} = \{(x_i, y_i)\}_{i=1}^{l}$ and calibration set $\mathcal{D}_{\text{proper}} = \{(x_i, y_i)\}_{i=m+1}^{l}$
Train an adversarial autoencoder $F$ using the proper training set $\mathcal{D}_{\text{proper}}$
for $j = m + 1$ to $l$ do
    Generate $\hat{x}_j$ using the trained adversarial autoencoder
    $\alpha_j^{\Gamma} = A(x_j, \hat{x}_j)$
end for
$\{\alpha_{m+1}, \dots, \alpha_l\}$ = sort($\{\alpha_{m+1}^{\Gamma}, \dots, \alpha_l^{\Gamma}\}$)
Online ($p$-value of expected nonconformity score):
for $k = 1$ to $N$ do
    Generate $\hat{x}_{l+1,k}$ using the trained adversarial autoencoder
    $\alpha_{l+1,k} = A(x_{l+1}, \hat{x}_{l+1,k})$
end for
$\bar{\alpha}_{l+1} = \frac{1}{N} \sum_{k=1}^{N} \alpha_{l+1,k}$
$p_{l+1} = \frac{|\{i=m+1,\dots,l\} \mid \alpha_i \geq \bar{\alpha}_{l+1}|}{l-m}$
$Anom_{l+1} \leftarrow p_{l+1} > \varepsilon$
Online (martingale test):
for $k = 1$ to $N$ do
    Generate $\hat{x}_{l+1,k}$ using the trained adversarial autoencoder
    $\alpha_{l+1,k} = A(x_{l+1}, \hat{x}_{l+1,k})$
    $p_{l+1,k} = \frac{|\{i=m+1,\dots,l\} \mid \alpha_i \geq \alpha_{l+1,k}|}{l-m}$
end for
$M_{l+1} = \int_0^1 \prod_{k=1}^{N} \varepsilon p_{l+1,k}^{\varepsilon-1} d\varepsilon$
$Anom_{l+1} \leftarrow M_{l+1} > \tau$
```

the non-image input (Experiment 3), the AAE is implemented with three fully connected layers. The decoder has a symmetric architecture, and the discriminator contains three fully-connected layers.

5.4.1.2 Evaluation metrics

The Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate by varying the detection threshold. The Area Under ROC (AUROC) curve is a threshold-free metric and is considered as the evaluation metric for OOD detection. The worst value of AUROC is 0.5 yielded by an uninformative classifier with a random guess. The best value of AUROC is 1.0, implying that the nonconformity scores for all the OOD data are greater than the score for any in-distribution data.

5.4.2 Out-of-distribution Data Caused by Covariate Shift

Experiment 1: The OOD data caused by covariate shift is present when the input variable x is sampled from a different distribution of training dataset, but the underlying relationship between the input and output $P(y|x)$ remains unchanged. MNIST [113], colorful MNIST [119], and SVHN [120], are the image classification datasets with the same labels of ten digits, while the inputs are from different distributions: for MNIST, the inputs are black and white images with handwritten digits; for colorful MNIST, the inputs are the MNIST images synthesized with colorful backgrounds; for SVHN, the inputs are the digit images from the street view house numbers. In our experiment, we train the AAE with the MNIST dataset and test it with colorful MNIST (Experiment 1-1) and SVHN (Experiment 1-2). It can be regarded as the OOD data caused by covariate shift since the input variable is sampled from a different distribution of training dataset, but the conditional probability of y given x stays the same.

Results of Experiment 1: We report the accuracy of the classification task and the AUROC of the detection task in Table 5.1 using different nonconformity measures and different techniques applied to nonconformity scores (”Ave” is for the technique using expected p-value of N nonconformity scores; ”Mart” is for the technique using martingale test). As it can be seen from the table, the accuracy of the classification for the in-distribution data is not degraded. Further, the AUROC in Experiment 1-1 and 1-2 are almost close to 1.0. Therefore, the proposed method can be used to detect the out-of-distribution data caused by covariate shift. Besides, it should be noted that for this experiment, no baselines can be compared in the literature.

Table 5.1: AUROC for inductive out-of-distribution detection.

	Accuracy	A_{rc}(Ave/Mart)	A_{saliency}(Ave/Mart)
Experiment 1-1	99.3%	0.998/0.999	0.999/0.999
Experiment 1-2		1.000/1.000	1.000/1.000
Experiment 2-1	99.5%	0.943/0.932	0940/0.931
Experiment 2-2	97.2%	0.840/0.847	0.829/0.821
Experiment 2-3	90.1%	0.692/0.683	0.683/0.690
Experiment 3	96.3%	0.682/0.693	0.674/0.681

5.4.3 Out-of-distribution Data Caused by Label Shift

Experiment 2: Novelty detection for unknown classes is a representative example of the OOD data caused by label shift, where the label variable y is not sampled from the same distribution of the training dataset, but the output-conditional distribution $P(x|y)$ remains the same. In our experiment, following the experimental settings in [126], we randomly sample 6 classes in MNIST (Experiment2-1) [113], SVHN (Experiment 2-2) [120], and CIFAR10 (Experiment 2-3) [129] as known classes, and rest 4 classes are unknowns. The training dataset only contains the 6 known classes, but the test dataset contains all 10 classes.

Results of experiment 2: In this case, we report the evaluation results in Table 5.1. The results demonstrate the effectiveness of our approach for detecting the OOD data caused by label shift. Although the AUROC of our approach is smaller than the other state-of-the-art methods [126; 130], our approach uses a more shallow neural network allowing for online detection.

5.4.4 Out-of-distribution Data Caused by Concept Shift

Experiment 3: The OOD data caused by concept shift is present when the output variable y is sampled from the training dataset, but the conditional probability of $P(x|y)$ changes. A gear dataset is used in the experiment to evaluate our approach for detecting such OOD data. Gearbox fault detection dataset [114] focuses on classifying the type of damage that may occur on a generic gearbox. The state of the gearbox is measured using accelerometers attached at various locations. The gearbox can operate at five different constant shaft speeds under two different loading conditions (low- and high-load). For each shaft speed and loading conditions, six fault types are simulated (normal, chipped gear tooth, broken gear tooth, bent shaft, imbalanced shaft, broken gear tooth with bent shaft). For each case which is the combination of fault type, shaft speed, and load condition, about 4 seconds of data are collected at a sampling rate of 66.67 kHz twice. To make the dataset suitable for this research, preprocessing is performed: In this chapter, only the output shaft vibration data is considered. The dataset is divided into two main subsets; those are low-load and high-load. For each subset, regardless of shaft speed, the dataset is aggregated with respect to the type of fault. All available data is converted into the frequency domain using Short Time Fourier Transform. Our experiment uses the subset from the low-load as the training dataset and both subsets from low- and high-load as the testing dataset. The distribution over output $P(y)$ stays the same, but the conditional probability $P(x|y)$ changes. Thus, it can be regarded as the OOD data caused by concept shift.

Results of experiment 3: Evaluation results corresponding to the experiment are shown in Table 5.1. Although there is no baseline in the literature used to compare with, as it can be seen from this table, the approach can detect OOD data caused by concept shift using different nonconformity measures without loss of classification accuracy.

5.4.5 Execution Time

In order to characterize the efficiency of the approach, we measure and report execution times for Experiment 1-1 using two different nonconformity measures and martingale test using a box plot in Figure 5.2. The execution times are measured on a 6-core Ryzen 5 desktop with a single GTX 1080Ti GPU.

From the results, the execution times of the method using nonconformity measure A_{saliency} are slightly longer than the method using A_{rc} due to the extra execution time for computing the saliency maps. The execution times for all two nonconformity measures are very short, which is comparable to the inference time of typical machine learning component [1]. Therefore, our approach is applicable for online detection. Moreover, the number of the examples generated from AAE N is fixed at 10 in experiments. As the number of N increases, the execution time will also increase since the AAE model needs to be inferred N times to generate N examples.

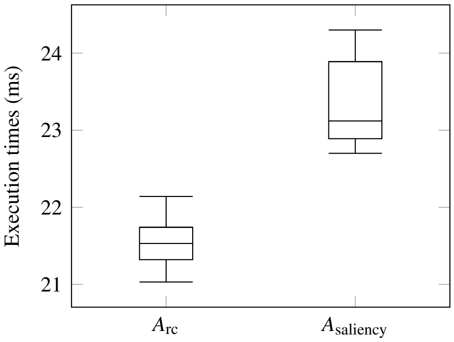


Figure 5.2: Execution times of proposed method.

5.5 Conclusion

In this chapter, we formalize the problem of detecting OOD data in machine learning components for classification and categorize the OOD data according to their causes. Then, we present an approach based on inductive conformal anomaly detection. An adversarial autoencoder model is adopted to characterize the joint distribution of the training dataset, allowing online detection for high-dimensional data. Experiments using several datasets demonstrate the effectiveness of the approach for the detection of different types of OOD data. Moreover, the execution time is very small, and consequently, the approach can be used for online out-of-distribution detection. Evaluation with real-world image datasets is a part of future work.

CHAPTER 6

Real-time Detection of Sensor Replay and Controller Integrity Attacks in Cyber-Physical Systems

6.1 Introduction

Cyber-Physical Systems (CPSs) involve the integration of sensors, actuators, and computation into physical systems using networking, and they are increasingly deployed in a wide range of domains such as aerospace, automotive, healthcare, manufacturing, and transportation [131]. Networking interconnections between components expose attack surfaces to adversaries who can launch various cyber attacks. Attacks to networked CPS can be deployed on either the sensor network that sends the sensor measurements to the controller or the actuator network that sends the control signals to physical system. In general, attacks can be categorized into two possible classes in [16]: (1) Denial of Service (DoS) attacks, where the adversary prevents the information from being transmitted, and (2) deception or integrity attacks, where the adversary sends false information to the controller or actuator.

Recent work demonstrates that successful attacks can lead the system to abnormal behavior which may significantly undermine the safety of the system and even cause loss of human life [10; 11]. Therefore, detection of attacks is paramount to the safe and reliable operation of CPS and has received considerable attention in the literature [132; 21]. In this chapter, we narrow down the scope to (1) replay attacks, a special kind of integrity attack to the sensor network and (2) integrity attacks to the actuator network. We consider the CPS domain of autonomous driving and we present an approach for efficiently and robustly detecting such attacks in real time.

The detection method relies only on high-dimensional observations from a camera in contrast to approaches that use additional input signals and need to change the architecture of the system [20; 21]. Detection of integrity attacks needs to consider time-series observation sequences because point-wise detection methods, such as the methods introduced in previous chapters, can only detect very simple integrity attacks and can be easily bypassed by more nuanced attacks. Although considerable efforts are made for detecting anomalies and attacks in time-series data [133; 10], typical approaches focus on low-dimensional data. The problem is challenging for high-dimensional data, for example, from camera and LIDAR sensor measurements in autonomous vehicles. Further, the detection must be performed in real time as observations become available during the system operation.

The first contribution of the chapter is a generative model used for detecting anomalies of high-dimensional time-series data. The model is inspired by the world model [134] for learning the long-term behavior of dynamical systems. By training this model with empirical data, we can capture the spatial and temporal characteristics of the normal dynamical system behavior. The model has the capacity of predicting future observations by (1) first utilizing the encoder of a VAE to compress a high-dimensional observation into a latent representation, (2) then using an RNN to predict the future encoding based on the compressed latent encoding and historical information, and (3) finally converting the predicted encoding to a predicted observation using the decoder of the VAE. The expected observation predicted by the model is compared with the actual observation to quantify the nonconformity of actual behavior relative to normal behavior learned during training.

The second contribution is an approach for real-time detection of sensor replay and controller integrity attacks in CPS. It is well known that RNNs typically lack the ability to capture the long-term dependencies. We propose to recursively use the RNN to predict the observations for multiple time steps in the future. During testing, the expected current observations predicted from multiple steps in the past are compared with the current actual observation, resulting in a series of nonconformity scores. These nonconformity scores are then combined using Inductive Conformal Anomaly Detection (ICAD) [63] allowing detection of abnormal behavior in a long sequence. Benefiting from the deep learning generative model, the approach is computationally efficient, thereby enabling real-time detection.

The final contribution of the chapter is the comprehensive evaluation for the approach using two case studies: (1) an Advanced Emergency Braking System (AEBS) implemented in CARLA simulator [103], and (2) an autonomous car racing example implemented in OpenAI Gym [135]. For the AEBS, we focus on detecting sensor replay attacks that affect the perception component that is used to estimate the distance to a front obstacle. The adversary has access to the sensor observations during the system operation and uses prerecorded data when the host vehicle is approaching the obstacle in order to deceive the controller and affect the braking, which in turn causes the vehicle to collide with the obstacle. Three attack scenarios with different sensor replay attacks are used to evaluate the proposed approach. The simulation results show that the method can detect these sensor replay attacks with a small number of false alarms and a short detection delay. For autonomous car racing, a controller is trained to perform the autonomous driving task using the world model [134]. We consider an integrity attack to this controller where the adversary modifies the control signal with a malicious command to lead the car off the racing track. The evaluation against such an attack validates the effectiveness of the proposed approach for detecting controller integrity attacks using the high-dimensional sensor observations. For both examples, the execution time of the detection method is much shorter than the sampling period of the system, which demonstrates proposed approach can be used for real-time detection.

The rest of the chapter is organized as follows: Section 6.2 presents the system and threat model, and then formulates the detection problem. Section 6.3 reviews the world model. Inspired by the world model, Section 6.4 introduces a generative model used for attack detection, which is a fundamental component in the proposed approach. Section 6.5 presents the algorithm for detecting sensor replay and controller integrity attacks. Section 6.6 presents the evaluation results, and Section 6.7 concludes the chapter.

6.2 Problem Formulation

In order to formulate the problem, let us consider an autonomous vehicle, whose simplified architecture is illustrated in Figure 6.1. The sensors which may include Inertial Measurement Unit (IMU), Global Positioning System (GPS), camera, and LIDAR observe the states of the autonomous vehicle and the environment, and feed the information to the controller through the sensor network. In order to realize a specific task, the controller interprets the sensor measurements, makes control decisions following predefined control logic, and sends the control commands, such as gas, brake, and steer signals, to the actuator through the actuator network. In response to the control commands, the states of the vehicle change, along with the environment of the vehicle, and the sensors are used again to close the operation loop of the system.

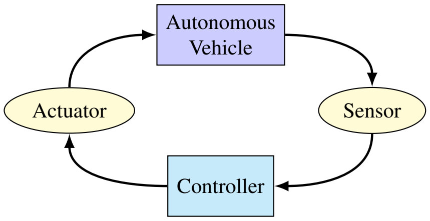


Figure 6.1: A typical CPS architecture.

Although the system may be well designed and tested, the sensor and actuator networks can still be vulnerable to cyber attacks when the system operates in the real world. A successful attack may lead to abnormal behavior of the system and greatly undermine the safety of the system. Therefore, detecting cyber attacks on CPS is of great significance.

In this chapter, we focus on replay attacks on the sensor networks and integrity attacks on the actuator networks, and consider the problem of efficiently and robustly detecting such attacks using only the sequence of high-dimensional observations from a camera. An alternative method could be to design detection methods that explore between sensors that observe the environment and vehicle sensors such as the IMU and GPS. However, such methods need to assume that some sensors are not attacked and require fusion of sensor measurements which may increase the attack surface. For this reason, the objective in the proposed approach is to detect attacks using only the high-dimensional observations. For the controller integrity attacks, it is also assumed that the sensor network and the control network are not attacked at the same time since in this case the adversary will be able to completely hide the effects of the attack.

During system operation, the time-series high-dimensional observations become available sequentially. At each time step, the objective of the detection is to quantify how strange the time series up to the current step is relative to the normal system behavior. Online detection requires the algorithm to use only the actual observation sequence that is not complete.

6.3 World Model

The world model is proposed for learning both spatial and temporal representations of a complex physical environment in [134]. By using compressed features, a controller can be trained to perform the required task. The architecture of the world model is shown in Figure 6.2, and it contains two components, Vision (V) and Memory (M). The vision model is a Variational Autoencoder (VAE), which encodes the current high-dimensional observation x_t into the low-dimensional latent representation z_t. The memory model is a Recurrent Neural Network (RNN), which takes the low-dimensional encoding z_t and the historical information embedded in the hidden states of RNN h_t, to predict the latent encoding z_{t+1} in the next time step. In [134], a simple single-layer linear controller C is trained separately from V and M to perform a specific task. It should be noted that the control signal can be used when the memory RNN makes the predictions for the future, which allows the memory M to take into account the influence of the control signal on the dynamical behavior. An important property of the world model is that after training, it can be used to imagine a virtual environment, called *dream environment* by recursively running the RNN to predict the latent encodings for future steps. This can be illustrated using Figure 6.2 where instead of using the compressed representations z_t of the actual observation x_t, the prediction from the last time step $z'_{(t-1)\to t}$ can be used as the input to the RNN to predict $z'_{(t-1)\to(t+1)}$ for the next step. In this fashion, the controller can be trained within this dream environment by only using a single seed observation instead of an actual environment.

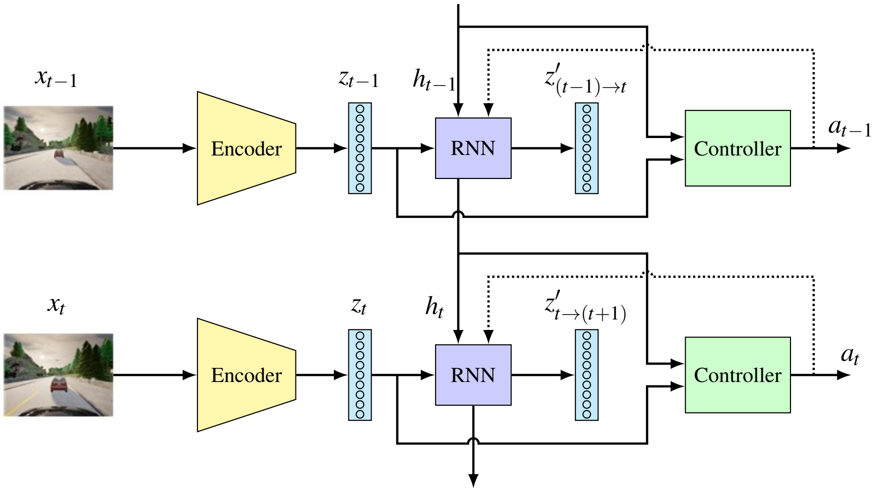


Figure 6.2: The architecture of the world model.

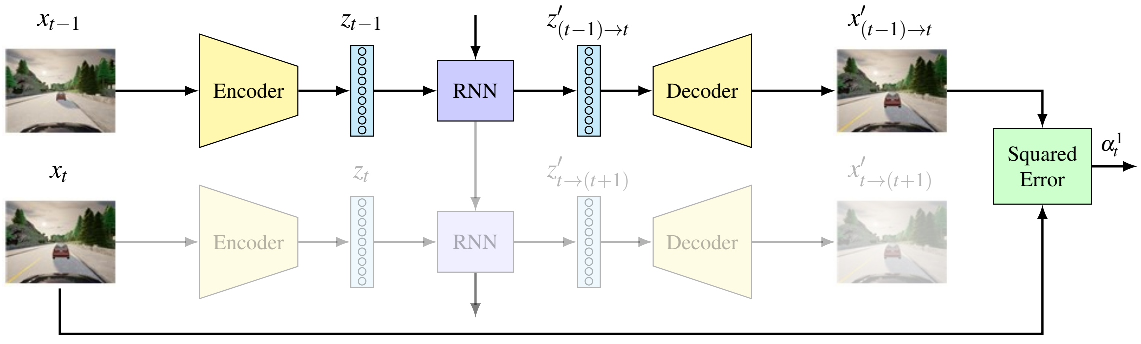


Figure 6.3: Illustration of 1-step nonconformity measure using the proposed generative model.

6.4 Generative Model for Detection

In this section, we propose a sequential generative model that is used for the detection of anomalies in high-dimensional sequences. The proposed model is inspired by the world model for learning the temporal and spatial representations of physical environments [134]. The model contains a VAE to encode the spatial information into a latent space representation and an RNN to encode the temporal dependencies into the hidden state. The idea is to compute a representation of the normal dynamical system behavior by training this generative model using empirical normal data.

The main advantage of the proposed model is that it allows predicting observations for future steps. Specifically, given the observation (image) x_{t-1}, the encoder of the VAE compresses the high-dimensional data point to a latent representation z_{t-1}. Then, the RNN is used to predict the latent representation z_t in the next step t. This part is similar to the world model. However, the world model directly utilizes the predicted low-dimensional representation z_t to train the agent policy, while the proposed generative model uses this representation to reconstruct the original image $x'_{(t-1)\to t}$. The expected observation $x'_{(t-1)\to t}$ can be predicted based on the previous observation x_{t-1} as illustrated in the top of Figure 6.3. Similar to the world model, the influence of the control signals can be considered by feeding the signals to the RNN to predict the next state.

The proposed generative model predicts the observations for multiple steps in the future similarly to the dream environment in the world model. In Figure 6.3, after getting the $z'_{(t-1)\to t}$ in the first inference of the RNN, the RNN continues to predict the next latent representation $z'_{(t-1)\to(t+1)}$ by using a virtual encoding $z'_{(t-1)\to t}$ instead of the actual encoding z_t. Recursively, the RNN can be used to compute m latent encodings $(z'_{(t-1)\to t},\dots,z'_{(t-1)\to(t+m-1)})$. Using the decoder, these encodings are converted to a series of predicted observations $(x'_{(t-1)\to t},\dots,x'_{(t-1)\to(t+m-1)})$. In this manner, the model can generate sequential data by only using a single seed input.

We should emphasize that it is also possible to include the control signals to make predictions for multiple steps. The control signals should be recursively generated by controller to feed to the RNN for predicting the next latent encoding. However, the actual observations are not available during the prediction process, and therefore, it is not applicable to use the observation directly in the controller. To overcome this limitation, the controller can use a low-dimensional representation encoded by the VAE as the input, similar to the controller in the world model [134].

The training procedure of the generative model consists of the following steps. After collecting the empirical training dataset, the VAE is trained first by minimizing the reconstruction error and the KL divergence between the prior and posterior [100]. Next, we compute and store the latent representations for the training data by feeding them into the trained VAE model. The last step is to train the RNN using the sequences of the latent representations of the training data. The objective is to predict the probability distribution of the latent representation in the next time step. In order to model the probability distribution of the latent representation, a Mixture Density Network (MDD) [136] is connected after the RNN as the output layer as in the training for the world model [134]. After training, the model can generate a sequence of consecutive images from a single input and the sequence is very similar to the actual sequence. Further, the model can predict the sequence of observations for N steps, and this prediction can be repeated at every time step resulting in N predictions corresponding to a specific time step in the future from N different time steps in the past.

6.5 Attack Detection

The detection method is based on Inductive Conformal Anomaly Detection (ICAD), which requires a suitable NonConformity Measure (NCM) defined to measure the difference between a test example and the training dataset. In this section, we first introduce a novel NCM that utilizes the proposed generative model. Then, based on this NCM, we present an ICAD algorithm for detecting sensor replay and controller integrity attacks.

6.5.1 Nonconformity Measure

Let us consider a set of normal sequences denoted by $\Gamma_{\text{train}} = \{\mathcal{X}_i\}_{i=1}^n$, where $\mathcal{X}_i$ is a time-series of data points $(x_1^i, \dots, x_{l_i}^i)$ with variable length l_i. During runtime, a sequence of test observations $(x_1^{n+1}, \dots, x_{l_{n+1}}^{n+1})$ arrive to the system one by one. At time step t, the objective is to first quantify how different the time-series up to t $(x_1^{n+1}, \dots, x_t^{n+1})$ is from the set of normal sequences Γ_{train} used for training, and then raise an alarm if the test sequence deviates considerably from the sequences used at design time.

Most of the NCMs used in ICAD, such as k-Nearest Neighbor (k-NN) NCM [64] and VAE-based NCM defined in Chapter 3, however, focus on point-wise detection. Namely, they aim to check if a single test instance x_t conforms to the empirical training dataset Γ_{train}. Such methods lack the capacity of capturing the temporal dependencies of time-series data, and cannot be used for detection of sensor replay and controller integrity attacks.

As discussed in Section 6.4, we propose a generative neural network model which aims to learn compressed spatial and temporal representations of the physical system and environment, and can be used to predict future observations. By training the generative model using the normal sequences Γ_{train}, the model captures the distribution of complex high-dimensional observation sequences. Therefore, if the actual observations do not conform to the predictions from the generative model, the sequence of observations can be regarded as anomalous behavior. Following the description of the model in Section 6.4 and Figure 6.3, we compute the squared error between the current actual observation x_t and predicted current observation from the last time step $x'_{(t-1)\to t}$ to compute the nonconformity, which is termed as 1-step NCM.

Although an RNN is used in the proposed model to take into account dependence on information at the previous step, it is very hard to learn long-term dependencies, especially for high-dimensional sequences. The 1-step NCM defined above can be used only across two adjacent time steps, and it is insensitive to attacks where the time-series sequence is changed gradually and slowly. In order to overcome this limitation, the generative model is used to predict the observations for multiple time steps in the future. The difference between the actual observation and the expected observation that is predicted from multiple steps in the past can be used to monitor long-term dependencies. The detailed description for predicting observations for multiple steps can be found in Section 6.4. We denote the process of predicting observation $x'_{(t-k)\to t}$ after k steps based on the observation x_{t-k} at $t-k$ as

$$x'_{(t-k)\to t} = \mathcal{P}(x_{t-k}, k),$$

and the k-step NCM at time t can be naturally defined as

$$\alpha_{t,k} = ||x_t - x'_{(t-k)\to t}||^2. \tag{6.1}$$

It should be noted that the control signal can be incorporated in the prediction process by feeding the control signal into the RNN. Therefore, using the same definitions of NCM, the NCM can capture the anomalies in the sequence of observations caused by malicious control signals.

6.5.2 Detection Algorithm

After the definition of the NCM based on the proposed generative model, we introduce the detection algorithm. The algorithm is divided into offline and online phases as describe below.

6.5.2.1 Offline phase:

During the offline phase, the training set of sequences $\Gamma_{\text{train}} = \{\mathcal{X}_i\}_{i=1}^{n}$, is split into a proper training set $\Gamma_{\text{proper}} = \{\mathcal{X}_i\}_{i=1}^{m}$ and a calibration set $\Gamma_{\text{calibration}} = \{\mathcal{X}_i\}_{i=m+1}^{n}$. The proper training set Γ_{proper} to train the generative model.

The next step is to compute the nonconformity scores for the sequences in the calibration set. In order to model the long-term dependencies, the difference between the actual current observation and the expected current observation predicted from multiple steps earlier is incorporated into the computation of the nonconformity. For each observation x_t^i in the calibration set from the sequence $\mathcal{X}_i : i \in \{m+1, \dots, n\}$, we compare the observation with N expected observations predicted from 1 to N steps earlier. Therefore, we compute N different nonconformity scores by using 1-step, 2-step, ..., N-step NCM (Equation (6.1)). It is worth noting that the choice of N depends on the dynamic evolution of the system. For a slowly evolving system, a large N should be chosen in order to be able to capture a significant change within N steps, which will be beneficial for detection. Moreover, for each observation x_t^i in the calibration dataset, we predict the observations for the next N steps $(x'_{t\to t+1}, x'_{t\to t+2}, \dots, x'_{t\to t+N})$. Although these observations are not used at the current time step, they are available for the calculation of nonconformity scores at the next N time steps.

Because the prediction accuracy will decrease as the number of prediction step k increases, the nonconformity score computed by comparing with a prediction from a longer past is very likely to be greater than the one computed by comparing with prediction from a shorter past. The nonconformity scores of the calibration data are used for comparing with nonconformity scores of the test data during online phase. Therefore, for fairly comparison, after computing the N nonconformity scores for each observation in the calibration data, the nonconformity scores are clustered into N different sets based on the number k of steps between the actual and predicted observations. Therefore, the k-th calibration set $\mathcal{C}_k$ contains all the nonconformity scores using the k-step NCM, that is $\mathcal{C}_k = \{\alpha_{t,k}^i : (i,t) = \{m+1, \dots, n\} \times \{1, \dots, l_i\}\}$. Each cluster of nonconformity scores is sorted and stored for use during the online phase. The detailed algorithms for the offline phase is shown in Algorithm 7.

6.5.2.2 Online phase:

During the online phase, we consider a sequence of observation $(x_1^{n+1}, \dots, x_{l_{n+1}}^{n+1})$ arriving at the detector one by one. At time step t, for the observation x_t^{n+1}, we compute N nonconformity scores $\{\alpha_{t,1}^{n+1}, \dots, \alpha_{t,N}^{n+1}\}$ in the same way as calibration data, i.e., by computing N squared errors between the actual observation x_t^{n+1} and the expected observations $\{x'^{n+1}_{(t-N)\to t}, \dots, x'^{n+1}_{(t-1)\to t}\}$ predicted from time steps $t-N$ to $t-1$. Using N nonconformity scores improves detection by considering long-term dependencies and also improves robustness.

Next, for each nonconformity score $\alpha_{t,k}^{n+1} : k \in \{1, \dots, N\}$, we compute its corresponding p-value as the

Algorithm 7 Offline phase of detecting sensor replay and controller integrity attacks.

```
Input: a training set of sequences $\Gamma_{\text{train}} = \{\mathcal{X}_i\}_{i=1}^{n}$; number of calibration sequences $n-m$; number of observations predicted from past $N$;
Output: $N$ calibration sets of nonconformity scores $\{\mathcal{C}_i\}_{i=1}^{N}$
Split the training set of sequences $\Gamma_{\text{train}} = \{\mathcal{X}_i\}_{i=1}^{n}$ into a proper training set of sequences $\Gamma_{\text{proper}} = \{\mathcal{X}_i\}_{i=1}^{m}$ and a calibration set of sequences $\Gamma_{\text{calibration}} = \{\mathcal{X}_i\}_{i=m+1}^{n}$
Train the generative model $\mathcal{P}$ (a VAE and a RNN) using the proper training set of sequences $\Gamma_{\text{proper}}$
for $i = m+1$ to $n$ do
    for $t = 1$ to $l_i$ do
        ▷ Compute the $N$ nonconformity scores
        for $k = 1$ to $N$ do
            $\alpha_{t,k}^{i} = ||x_t^i - x'^{i}_{(t-k)\to t}||^2$
        end for
        ▷ Predict the observations for next $N$ time steps
        for $k = 1$ to $N$ do
            $x'^{i}_{t\to(t+k)} = \mathcal{P}(x_t^i, k)$
        end for
    end for
end for
Construct $N$ calibration sets of nonconformity scores $\{\mathcal{C}_i\}_{i=1}^{N}$, where $\mathcal{C}_k = \{\alpha_{t,k}^{i} : i = m+1, \ldots, n; t = 1, \ldots, l_i\}$
```

fraction of nonconformity scores in k-th calibration set $\mathcal{C}_k$ that are greater or equal to $\alpha_{t,k}^{n+1}$. This p-value can be expressed as

$$p_{t,k} = \frac{|\{\alpha_k \in \mathcal{C}_k | \alpha_k \geq \alpha_{t,k}^{n+1}\}|}{|\mathcal{C}_k|}.$$

It is possible that $\alpha_{t,k_1}^{n+1} < \alpha_{t,k_2}^{n+1}$ for $k_1 < k_2$ since the observation predicted from $t-k_2$ may not be as accurate as the one predicted from $t-k_1$. By clustering the nonconformity scores of the calibration data based on the number of prediction step k used in their computation, we do not need to compare the actual observations with predictions made at different time steps which may be of different quality. Merging these nonconformity scores, for example, using average is not appropriate and will result in loss of information since the p-value for α_{t,k_1}^{n+1} could be larger than the p-value for α_{t,k_2}^{n+1}. Again, since the nonconformity scores for the calibration data are sorted in the offline phase, the calculation of p-values can be accelerated by using binary search algorithm.

The next step is to use the martingale to test if there are many small p-values in the set of $\{p_{t,k}\}_{k=1}^{N}$ [69], which is defined in Equation (3.3). If there are many small p-values, M_t will be very large indicating a sequence of observations that are not conformal to the training data. By incorporating the control signal into the prediction process, the NCM can measure the anomalies caused by malicious control signals, and our detection method can thus detect the controller integrity attacks.

Using the same idea as Section 3.3.2, a CUSUM stateful detector test is applied to the sequence of martingale values in order to detect the anomaly robustly. The detailed algorithm of the online phase can be found in Algorithm 8.

Algorithm 8 Online phase of detecting sensor replay and controller integrity attacks.

```
Input: a test sequence $\mathcal{X}_{n+1}$, where each sequence $\mathcal{X}_i = (x_1^i, \dots, x_{l_i}^i), i = 1, \dots, n+1$; $N$ calibration sets of
    nonconformity scores $\{\mathcal{C}_i\}_{i=1}^N$; number of observations predicted from past $N$; threshold $\tau$ and parameter
    $\omega$ of CUSUM detector
Output: boolean variable $Anom_t$
for $t = 1$ to $l_{n+1}$ do
    ▷ Compute the $N$ nonconformity scores and $p$-values
    for $k = 1$ to $N$ do
        $\alpha_{t,k}^{n+1} = ||x_t^{n+1} - x'^{n+1}_{(t-k)\to t}||^2$
        $p_{t,k} = \frac{|\{\alpha_k \in \mathcal{C}_k | \alpha_k \geq \alpha_{t,k}^{n+1}\}|}{|\mathcal{C}_k|}$
    end for
    ▷ Compute the martingale value
    $M_t = \int_0^1 \prod_{k=1}^N \varepsilon p_{t,k}^{\varepsilon-1} d\varepsilon$
    ▷ CUSUM procedure
    if $t = 1$ then
        $S_t = 0$
    else
        $S_t = \max(0, S_{t-1} + M_{t-1} - \delta)$
    end if
    $Anom_t \leftarrow S_t > \tau$
    ▷ Predict the observations for next $N$ timesteps
    for $k = 1$ to $N$ do
        $x'^{n+1}_{t\to(t+k)} = \mathcal{P}(x_t^{n+1}, k)$
    end for
end for
```

6.6 Evaluation

In this section, we evaluate the proposed approach for detecting (1) sensor replay attacks using an Advanced Emergency Braking System (AEBS) implemented in the CARLA simulator [103], and (2) controller integrity attacks using an autonomous car racing example implemented in OpenAI Gym [135]. The experiments are performed on a Ubuntu Linux virtual machine with a 48-core CPU and an RTX 6000 GPU, which is provided by Chameleon Cloud Platform [137] supported by the National Science Foundation.

6.6.1 Advanced Emergency Braking System

Advanced Emergency Braking System (AEBS) is a typical CPS, whose objective is to detect the an approaching obstacle and safely stop the host vehicle. A perception component receives the image captured by an onboard camera and uses a convolutional neural network to estimate the distance to a front obstacle. The distance, along with the velocity of the host vehicle, is used by a reinforcement learning controller to generate an appropriate brake force to stop the host vehicle avoiding a potential collision. More details about the AEBS can be found in Section 3.6.1.

6.6.1.1 Experimental setup

We collect 100 episodes by varying the start position of the host vehicle. The data is split into two partitions: 80 episodes are used as the proper training set and 20 as the calibration set. We collect 100 additional episodes that are used as normal test sequences.

The detection method utilizes a generative model consisting a VAE and an RNN, which are trained using the proper training set. We use similar network architectures and training hyperparameters as in [134]. Since the controller is not considered in this experiment, the RNN in the generative model uses only the latent representations from the VAE and does not include the controller signals in its input. We compare the proposed method against a VAE-based method that considers only individual frames presented in Chapter 3.

6.6.1.2 Replay attack I

The sensor replay attack occurs when an adversary (1) collects the sensor observations and (2) replays the collected data at different time instants of an episode. Using a replay attack, the adversary can spoof the camera images with images recorded earlier from the same camera. The objective of the adversary is to cause the predicted distance to the obstacle to be larger than the actual distance. In this case, the controller will be deceived to apply a soft brake force even when the host vehicle is very close to the obstacle, which will eventually cause the host vehicle to collide with the obstacle.

We use 100 episodes of normal sequences and 100 episodes of sequences under attack to evaluate the detection method. In the sequences under attack, the video segment that is recorded between the time period t_0 to t_0+40 is replayed in a forward order at time step t_1. The time instants t_0 and t_1 are randomly sampled from $\{5,\dots,30\}$ and $\{80,\dots,100\}$, respectively.

In order to characterize the detection performance, we report the false alarms and the average detection delay by considering different values for the prediction horizon N and CUSUM parameters ω and τ that are shown in Table 6.1. The results show that the proposed method can detect such sensor replay attack with a few false alarms and a very short delay. In contrast, the VAE-based method that uses only individual frames performs considerably worse than the proposed method exhibiting large number of false positives and negatives and large delay of detection.

We show the simulation results for a specific sequence under attack in Figure 6.4 where we plot the predicted distance to the obstacle, p-values, and stateful detector S-values for the VAE-based and proposed method. In the episode shown, the replay attack starts at time step 96 corresponding to 4.8 s from the beginning of the episode (with a sampling rate of 50 ms.) The frames used in the attacks are form the same

Table 6.1: False alarms and detection delay for detecting sensor replay attack I in AEBS.

Types	N, ω, τ	False positive	False negative	Average delay (frames)
VAE	$10, 1, 147$	$14/100$	$82/100$	12.44
	$10, 0, 243$	$17/100$	$81/100$	14.21
	$20, -2, 282$	$26/100$	$71/100$	12.38
	$20, 3, 260$	$14/100$	$83/100$	10.82
Proposed	$10, 11, 6$	$6/100$	$7/100$	0.08
	$10, 13, 3$	$5/100$	$8/100$	0.05
	$20, 13, 15$	$13/100$	$3/100$	0.16
	$20, 22, 4$	$5/100$	$10/100$	0.05

distribution as the training dataset, and therefore, the VAE-based method that uses individual frames cannot detect the attack and trigger an alarm. Since the proposed method uses a model that captures the temporal dependencies of the observation sequences, the p-values become very small once the attack occurs, and can be detected using the approach.

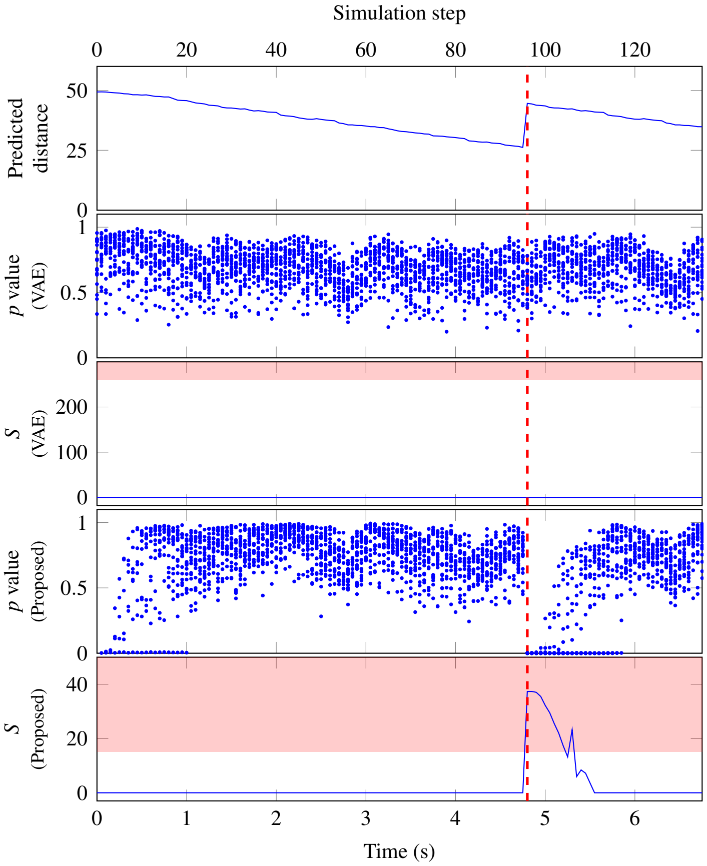


Figure 6.4: An episode under the sensor replay attack I in AEBS (detector parameter: (1) VAE: $N = 20$, $\omega = 3$, $\tau = 260$; (2) Proposed: $N = 20$, $\omega = 13$, $\tau = 15$).

6.6.1.3 Replay attack II

In the experiment described above, the replay attack results in abrupt changes in the observation sequences that can be easily detected by the proposed approach. In this experiment, we consider a type of replay attack that changes the observations gradually and slowly. In this scenario, at the time step t_1, an adversary starts to replay the collected from time t_0 in reverse, that is from the camera point of view, the vehicle starts moving backwards. Suppose the normal sequence is denoted as $(x_1,\ldots,x_{t_1-1},x_{t_1},x_{t_1+1},\ldots,x_l)$. Using this attack, this sequence becomes $(x_1,\ldots,x_{t_1-1},x_{t_1},x_{t_1-1},\ldots,x_1)$, where t_1 is randomly selected in $\{80,\ldots,100\}$. We collect 100 episodes with this replay attack for evaluation.

Figure 6.5 illustrates the detection process. It is not surprising that the VAE-based detector trained on individual frames cannot detect this type of attack. In the proposed approach, the p-values start to decrease when the attack is deployed and the detector generates alarms indicating that the sequence is abnormal. Table 6.2 reports the false alarms and average detection delay for the two methods. The results demonstrate that the approach can detect the slowly changing replay attack with a small false alarms. The number of false alarms for detecting the replay attack II is larger than the replay attack I because in replay attack II, the changes at the observation sequences are very gradual. The performance of detector for a large time horizon (e.g., $N = 20$) is improving since it takes a longer time for such an attack to change the observation sequence.

Table 6.2: False alarms and detection delay for detecting sensor replay attack II in AEBS.

Types	N,ω,τ	False positive	False negative	Average delay (frames)
VAE	$10,1,148$	$14/100$	$78/100$	38.68
	$10,0,227$	$17/100$	$75/100$	38.08
	$20,-3,0$	$100/100$	$0/100$	0.04
	$20,27,296$	$0/100$	$100/100$	N/A
Proposed	$10,7,6$	$14/100$	$21/100$	25.62
	$10,8,3$	$15/100$	$20/100$	25.81
	$20,4,149$	$12/100$	$15/100$	25.46
	$20,7,98$	$11/100$	$16/100$	25.02

6.6.1.4 Stuck sensor attack

The sensor may get stuck due to an attack. In this experiment, we simulate a scenario where the attack results in the camera sensor getting stuck which can be viewed as a particular type of replay attack. We assume that the camera sensor is stuck at time step t_1, where t_1 is randomly sampled in $\{80,\ldots,100\}$ and therefore, the observation sequence can be denoted as $(x_1,\ldots,x_{t_1},\ldots,x_{t_1})$.

We plot the p-values and the detector S-values for both two detection approaches in Figure 6.6. For the VAE-based method, the p-values are between 0 and 1, and S-values are almost 0 through the whole episode reflecting that it cannot be used for detection of such a sequence. In contrast, the proposed method can detect such sequence with a very small delay. Once the camera gets stuck, the p-values start to decrease

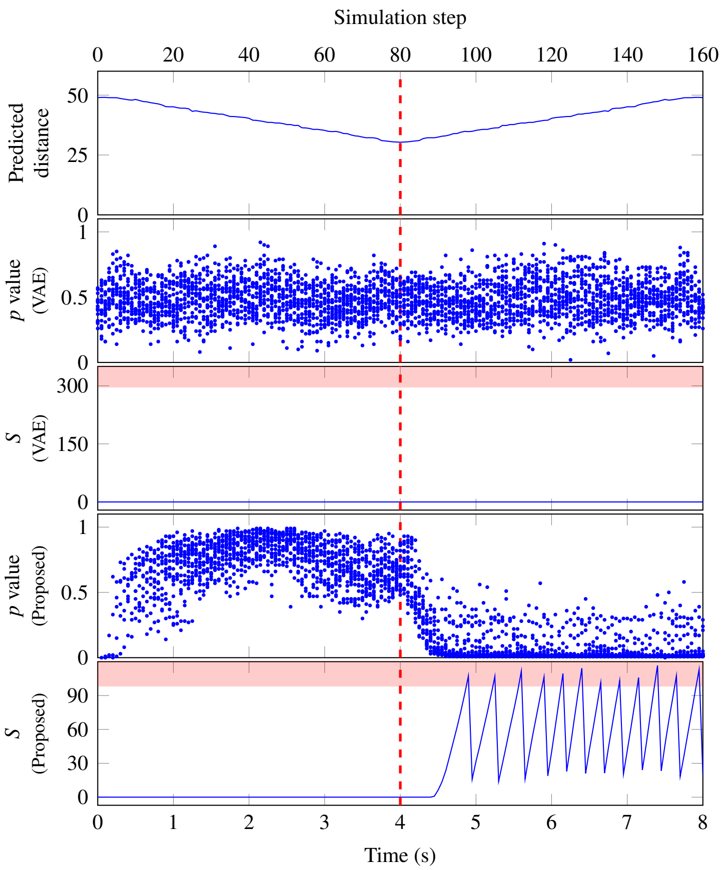


Figure 6.5: An episode under the sensor replay attack II in AEBS (detector parameter: (1) VAE: $N = 20$, $\omega = 27$, $\tau = 296$; (2) Proposed: $N = 20$, $\omega = 7$, $\tau = 98$).

approaching 0, and the S-value increases and exceeds the threshold of the CUSUM detector. We also report the false alarms and average detection delay in Table 6.3. The number of false alarms for this scenario is larger than the two previous attacks because there is no considerable change in the observations.

Table 6.3: False alarms and detection delay for detecting stuck sensor attack in AEBS.

Types	N, ω, τ	False positive	False negative	Average delay (frames)
VAE	$10, 12, 210$	$0/100$	$100/100$	N/A
	$10, 23, 4$	$0/100$	$100/100$	N/A
	$20, -3, 98$	$64/100$	$25/100$	11.70
	$20, 0, 28$	$58/100$	$30/100$	9.73
Proposed	$10, -2, 64$	$16/100$	$21/100$	17.08
	$10, -2, 72$	$14/100$	$23/100$	17.89
	$20, -2, 57$	$14/100$	$16/100$	16.00
	$20, -1, 35$	$12/100$	$18/100$	14.53

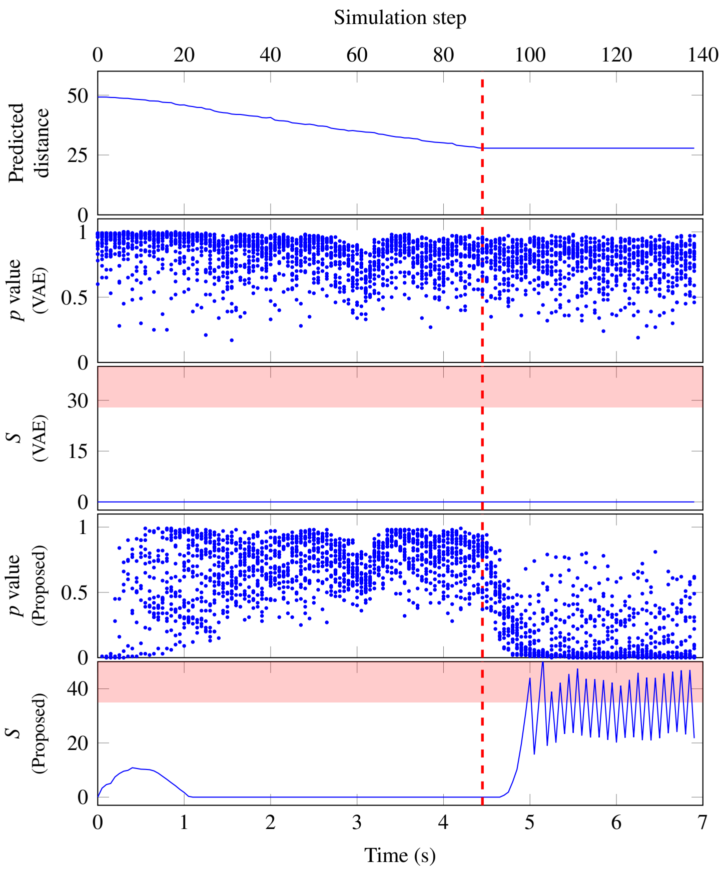


Figure 6.6: An episode under the stuck sensor attack in AEBS (detector parameter: (1) VAE: $N = 20$, $\omega = 0$, $\tau = 28$; (2) Proposed: $N = 20$, $\omega = -1$, $\tau = 35$).

6.6.2 Autonomous Car Racing

We evaluate the approach for detecting controller integrity attacks in an autonomous car racing example from OpenAI Gym [135]. The task here is to use the pixel inputs from the top-down camera looking at a racing environment to learn a controller for the throttle, brake and steer signal so that the autonomous car follows the track. In [134], a world model is trained to extract the spatial and temporal representation of the autonomous car racing environment, and a reinforcement controller is trained by using the compressed features from the world model. It should be noted that in this example, the control signal along with the latent space representation, are used as inputs to the RNN in the generative model to predict the future states and observations. Therefore, we can evaluate if the approach can detect anomalies caused by the integrity attacks on the control signal using only the observation sequence from the top-down camera.

6.6.2.1 Experimental setup

We use the same network architecture and training process as in [134] to train the world model including the controller. Using the trained model, we collect 900 episodes by randomizing the starting position of the car. 800 of these episodes are used as calibration data while 100 episodes are used for testing as normal sequences. As discussed in Sec. 6.5, the control signals must be incorporated in the detection method because they are used in the RNN to predict the future observations.

6.6.2.2 Experimental results

In order to generate the observation sequences under attack, the original control signal is replaced with a full gas, zero brake, and full opposite steer control signal at a random time step $t \in \{50, \dots, 79\}$, which will cause the car to drive off the track. We collect 100 episodes that are used for testing positive abnormal sequences. We evaluate both the VAE-based and the proposed method using 100 episodes with normal sequences and 100 episodes with abnormal sequences. The attack parameters are selected to evaluate how fast we can detect an attack with catastrophic consequences.

We plot the detection results for an abnormal sequence in Figure 6.7 for both approaches. In this episode, the attack starts at time step 64, and the car then turns sharply to the left off the track. Such abnormal behavior of the car deviates significantly from what is expected by the world model as illustrated in Figure 6.8. The proposed method can capture such deviations because the p-values become small and the CUSUM detector generates alarms. The VAE-based detector performs much worse because it can reconstruct the observations for individual frames quite well.

We also report the number of false alarms and average detection delay in Table 6.4 by considering different values of the prediction horizon N and CUSUM parameters ω and τ. From the results, we can see that the number of false alarms is decreasing with larger prediction horizon. Not surprisingly, the VAE-based method performs much worse than the proposed method.

Table 6.4: False alarms and detection delay for detecting controller integrity attacks in autonomous car racing example.

Types	N, ω, τ	False positive	False negative	Average delay (frames)
VAE	$10, -2, 53$	$62/100$	$29/100$	11.63
	$10, 0, 9$	$55/100$	$39/100$	9.68
	$20, -3, 98$	$64/100$	$25/100$	11.70
	$20, 0, 28$	$58/100$	$30/100$	9.73
Proposed	$10, -2, 33$	$27/100$	$12/100$	8.82
	$10, -1, 15$	$22/100$	$20/100$	11.49
	$20, -3, 73$	$12/100$	$13/100$	8.75
	$20, 0, 6$	$7/100$	$18/100$	7.87

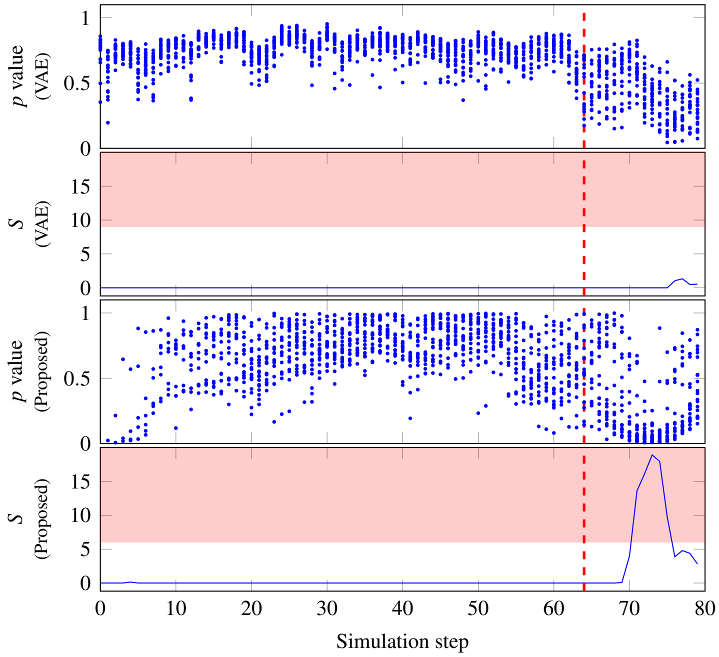


Figure 6.7: An episode under the controller integrity attacks in autonomous car racing example (detector parameter: (1) VAE: $N = 20$, $\omega = 0$, $\tau = 9$; (2) Proposed: $N = 20$, $\omega = 0$, $\tau = 6$).

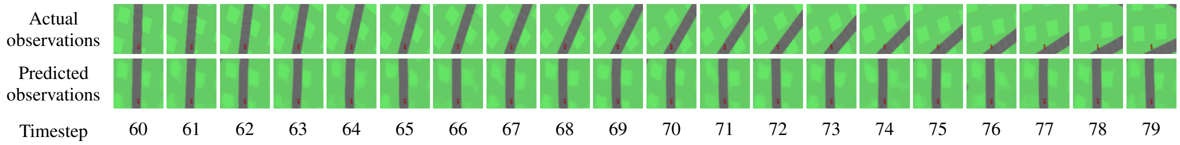


Figure 6.8: Comparison between the actual and predicted observations when the control signal is under attack in autonomous car racing example; all the predicted observations are predicted at time step 59.

6.6.3 Computational Efficiency

In order to detect the attacks in real time, the execution times of the detector must be smaller than the sampling period of the system. Because the proposed detector in both examples use similar network architectures as in [134], we focus on the AEBS and report the minimum (min), first quartile (Q_1), second quartile or median (Q_2), third quartile (Q_3), and maximum (max) of the execution times for for different values of N. Of course, the execution times become longer as N increases. The execution times are much smaller than the sampling period 50 ms, and therefore, the method can be used for real-time detection.

Table 6.5: Execution times (ms) for detecting sensor replay attack in AEBS.

N	min	Q_1	Q_2	Q_3	max
10	13.22	13.43	13.47	13.55	14.92
20	25.95	26.16	26.27	26.41	30.77

6.7 Conclusion

In this chapter, we propose and demonstrate an approach for detecting sensor replay and controller integrity attacks in CPS. The method is based on inductive conformal anomaly detection and utilizes a novel generative model inspired by the world model for efficiently measuring the nonconformity of the high-dimensional observation sequences relative to the normal system behavior, thereby allowing for real-time detection. The evaluation is performed using two simulation case studies: an advanced emergency braking system and an autonomous car racing example. The results show a small number of false alarms and small detection delay, and the execution time of the detection algorithm is much smaller than the sampling period of the systems. The proposed approach can be extended to detect abnormal behaviors that are not limited to attacks.

CHAPTER 7

Conclusions

Cyber-physical systems are greatly benefited by using learning-enabled components that can handle the uncertainty and variability of the real world. However, during the system operation, the learning-enabled components may still encounter data that are different from the data used for training. Out-of-distribution data may lead to a large prediction error and compromise the safety of the system. This dissertation proposes several detection algorithms based on inductive conformal anomaly detection framework. By using different learning models, the training distribution is represented, and the nonconformity between the test example and training data can be efficiently quantified, which enables the real-time detection for high-dimensional inputs. Additionally, the effectiveness of algorithms is supported by extensive evaluation on multiple simulated systems and datasets. Although further efforts are still needed to improve the robustness of detection in the face of more complicated scenarios, we hope our work will be helpful for future exploration.

List of Publications[1]

Submitted

1. **F. Cai** and X. Koutsoukos, “Real-time detection of sensor replay and controller integrity attacks in cyber-physical systems.”★

2. **F. Cai** and X. Koutsoukos, “Real-time out-of-distribution detection in cyber-physical systems with learning-enabled components.”★

3. B. Potteiger, A. Dubey, **F. Cai**, X. Koutsoukos, and Z. Zhang, “Moving target defense for the security and resilience of mixed time and event triggered cyber-physical systems.”

Published

1. B. Potteiger, **F. Cai**, Z. Zhang, and X. Koutsoukos, “Data space randomization for securing cyber-physical systems,” in *International Journal on Information Security*, pp.1–14, 2021.

2. J. Li, **F. Cai**, and X. Koutsoukos, “Byzantine resilient aggregation in distributed reinforcement learning,” in *18th International Conference on Distributed Computing and Artificial Intelligence (DCAI)*, October 2021.

3. **F. Cai**, A. Ozdagli, N. Potteiger, and X. Koutsoukos, “Inductive conformal out-of-distribution detection based on adversarial autoencoder,” in *IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, August 2021.★

4. **F. Cai**, A. Ozdagli, and X. Koutsoukos, “Detection of dataset shifts in learning-enabled cyber-physcial systems,” in *4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, May 2021.★

5. **F. Cai**, J. Li, and X. Koutsoukos, “Detecting adversarial examples in learning-enabled cyber-physical systems using variational autoencoder for regression,” in *IEEE Workshop on Assured Autonomous Systems (In conjunction with 2020 IEEE S&P)*, May 2020.★

6. B. Potteiger, **F. Cai**, A. Dubey, X. Koutsoukos, and Z. Zhang, “Security in mixed time and event triggered cyber-physical systems using moving target defense,” in *23rd IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, May 2020. **Nominated for Best Paper Award**

7. **F. Cai** and X. Koutsoukos. “Real-time out-of-distribution detection in learning-enabled cyber-physical systems,” in *11th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, April 2020.★ **Best Paper Award Finalist**

[1] The publications of high relevance for this dissertation are marked with ★.

8. HD. Tran, **F. Cai**, ML. Diego, P. Musau, TT. Johnson, X. Koutsoukos. "Safety verification of cyber-physical systems with reinforcement learning control," in *ACM Transactions on Embedded Computing Systems (TECS)*, October 2019.

9. HD. Tran, **F. Cai**, ML. Diego, P. Musau, TT. Johnson, X. Koutsoukos. "Safety verification reinforcement learning control," in *2nd Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS)*, July 2019.

BIBLIOGRAPHY

[1] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint*, 2018.

[2] S. M. Grigorescu, B. Trasnea, L. Marina, A. Vasilcoi, and T. T. Cocias, "Neurotrajectory: A neuroevolutionary approach to local state trajectory learning for autonomous vehicles," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3441–3448, 2019.

[3] S. Lefevre, A. Carvalho, and F. Borrelli, "Autonomous car following: A learning-based approach," in *IEEE Intelligent Vehicles Symposium*, pp. 920–926, IEEE, 2015.

[4] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Sim-atav: Simulation-based adversarial testing framework for autonomous vehicles," in *21st International Conference on Hybrid Systems (HSCC)*, 2018.

[5] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," *Journal of Automated Reasoning*, vol. 63, no. 4, pp. 1031–1053, 2019.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[7] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *5th International Conference on Learning Representations*, 2017.

[8] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," *arXiv preprint*, 2017.

[9] Y.-C. Hsu, Y. Shen, H. Jin, and Z. Kira, "Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR*, 2020.

[10] E. Habler and A. Shabtai, "Using LSTM encoder-decoder algorithm for detecting anomalous ADS-B messages," *Computer & Security*, vol. 78, pp. 155–173, 2018.

[11] M. T. Garip, M. E. Gursoy, P. Reiher, and M. Gerla, "Congestion attacks to autonomous cars using vehicular botnets," in *NDSS Workshop on Security of Emerging Networking Technologies (SENT)*, 2015.

[12] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," in *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[13] R. McAllister, G. Kahn, J. Clune, and S. Levine, "Robustness to out-of-distribution inputs via task-aware generative uncertainty," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[14] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," *arXiv preprint*, 2016.

[15] X. Gu and A. Easwaran, "Towards safe machine learning for cps: infer uncertainty from training data," in *10th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2019.

[16] A. A. Cárdenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *28th IEEE International Conference on Distributed Computing Systems Workshops*, 2008.

[17] Y. Mo and B. Sinopoli, "Integrity attacks on cyber-physical systems," in *1st International Conference on High Confidence Networked Systems (HiCoNS)*, 2012.

[18] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 13:1–13:33, 2011.

[19] Z.-H. Pang, G. Liu, and Z. Dong, "Secure networked control systems under denial of service attacks," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 8908–8913, 2011.

[20] A. Hoehn and P. Zhang, "Detection of replay attacks in cyber-physical systems," in *American Control Conference (ACC)*, pp. 290–295, IEEE, 2016.

[21] Y. Mo, S. Weerakkody, and B. Sinopoli, "Physical authentication of control systems: Designing watermarked control inputs to detect counterfeit sensor outputs," *IEEE Control Systems Magazine*, vol. 35, no. 1, pp. 93–109, 2015.

[22] F. Nizam, S. Chaki, S. Al Mamun, M. S. Kaiser, *et al.*, "Attack detection and prevention in the cyber physical system," in *International Conference on Computer Communication and Informatics (ICCCI)*, 2016.

[23] E. M. Ferragut, J. Laska, M. M. Olama, and O. Ozmen, "Real-time cyber-physical false data attack detection in smart grids using neural networks," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, 2017.

[24] A. Anwar, A. Mahmood, B. Ray, M. A. Mahmud, and Z. Tari, "Machine learning to ensure data integrity in power system topological network database," *Electronics*, vol. 9, no. 4, p. 693, 2020.

[25] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[26] H. Wang, M. J. Bah, and M. Hammad, "Progress in outlier detection techniques: A survey," *IEEE Access*, vol. 7, pp. 107964–108000, 2019.

[27] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms: Experiments and analyses," *Pattern Recognition*, vol. 74, pp. 406–421, 2018.

[28] A. Holst, J. Ekman, and S. Larsen, "Abnormality detection in event data and condition counters on regina trains," in *IET International Conference On Railway Condition Monitoring*, 2006.

[29] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. Springer, 1986.

[30] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[31] L. Tarassenko, P. Hayton, N. Cerneaz, and M. Brady, "Novelty detection for the identification of masses in mammograms," in *4th International Conference on Artificial Neural Networks*, 1995.

[32] B. Abraham and A. Chuang, "Outlier Detection and Time Series Modeling," *Technometrics*, vol. 31, no. 2, pp. 241–248.

[33] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[34] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in *Applications of data mining in computer security*, pp. 77–101, Springer, 2002.

[35] E. M. Knorr and R. T. Ng, "A unified approach for mining outliers," in *Conference of the Centre for Advanced Studies on Collaborative research*, 1997.

[36] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.

[37] R. Laxhammar and G. Falkman, "Sequential conformal anomaly detection in trajectories based on hausdorff distance," in *14th International Conference on Information Fusion*, 2011.

[38] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM SIGMOD international conference on Management of data*, 2000.

[39] J. MacQueen *et al.*, “Some methods for classification and analysis of multivariate observations,” in *5th Berkeley symposium on mathematical statistics and probability*, 1967.

[40] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *8th IEEE International Conference on Data Mining*, 2008.

[41] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, “Support vector method for novelty detection,” in *Advances in neural information processing systems (NIPS)*, pp. 582–588, 2000.

[42] D. M. Tax and R. P. Duin, “Support vector data description,” *Machine learning*, vol. 54, no. 1, pp. 45–66, 2004.

[43] D. Xu, E. Ricci, Y. Yan, J. Song, and N. Sebe, “Learning deep representations of appearance and motion for anomalous event detection,” *arXiv preprint*, 2015.

[44] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, “High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning,” *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[45] G. Pang, C. Shen, L. Cao, and A. v. d. Hengel, “Deep learning for anomaly detection: A review,” *arXiv preprint*, 2020.

[46] E. J. Candès, X. Li, Y. Ma, and J. Wright, “Robust principal component analysis?,” *Journal of the ACM (JACM)*, vol. 58, no. 3, pp. 1–37, 2011.

[47] P. Li, T. J. Hastie, and K. W. Church, “Very sparse random projections,” in *12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 287–296, 2006.

[48] J. Andrews, T. Tanay, E. J. Morton, and L. D. Griffin, “Transfer representation-learning for anomaly detection,” in *Journal of Machine Learning Research*, 2016.

[49] A. Mahmood, M. Bennamoun, S. An, and F. Sohel, “Resfeats: Residual network based features for image classification,” in *IEEE international conference on image processing (ICIP)*, 2017.

[50] S. Hawkins, H. He, G. Williams, and R. Baxter, “Outlier detection using replicator neural networks,” in *International Conference on Data Warehousing and Knowledge Discovery*, 2002.

[51] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis, “Learning temporal regularity in video sequences,” in *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[52] J. An and S. Cho, “Variational autoencoder based anomaly detection using reconstruction probability,” *Special Lecture on IE*, 2015.

[53] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, “Lstm-based encoder-decoder for multi-sensor anomaly detection,” *arXiv preprint*, 2016.

[54] W. Luo, W. Liu, and S. Gao, “Remembering history with convolutional lstm for anomaly detection,” in *IEEE International Conference on Multimedia and Expo (ICME)*, 2017.

[55] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery,” in *International conference on information processing in medical imaging*, Springer, 2017.

[56] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, “Efficient gan-based anomaly detection,” *arXiv preprint*, 2018.

[57] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, “f-anogan: Fast unsupervised anomaly detection with generative adversarial networks,” *Medical image analysis*, vol. 54, pp. 30–44, 2019.

[58] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, “Ganomaly: Semi-supervised anomaly detection via adversarial training,” in *Asian conference on computer vision*, Springer, 2018.

[59] W. Liu, W. Luo, D. Lian, and S. Gao, “Future frame prediction for anomaly detection–a new baseline,” in *IEEE Conference on Computer Vision and Pattern Recognition (ICCV)*, 2018.

[60] C. Huang, Y. Wu, Y. Zuo, K. Pei, and G. Min, “Towards experienced anomaly detector through reinforcement learning,” in *32nd AAAI Conference on Artificial Intelligence*, 2018.

[61] M.-h. Oh and G. Iyengar, “Sequential anomaly detection using inverse reinforcement learning,” in *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[62] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*. Springer-Verlag, 2005.

[63] R. Laxhammar and G. Falkman, “Inductive conformal anomaly detection for sequential detection of anomalous sub-trajectories,” *Annals of Mathematics and Artificial Intelligence*, vol. 74, pp. 67–94, 2015.

[64] J. Smith, I. Nouretdinov, R. Craddock, C. Offer, and A. Gammerman, “Anomaly detection of trajectories with kernel density estimation by conformal prediction,” in *International Conference on Artificial Intelligence Applications and Innovations (AIAI)*, 2014.

[65] R. Laxhammar and G. Falkman, “Conformal prediction for distribution-independent anomaly detection in streaming vessel data,” in *1st international workshop on novel data stream pattern mining techniques*, 2010.

[66] R. Laxhammar and G. Falkman, “Online learning and sequential anomaly detection in trajectories,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 6, pp. 1158–1173, 2014.

[67] V. Ishimtsev, A. Bernstein, E. Burnaev, and I. Nazarov, “Conformal *k*-NN anomaly detector for univariate data streams,” vol. 60 of *Proceedings of Machine Learning Research*, pp. 213–227, PMLR, 2017.

[68] V. Vovk, I. Nouretdinov, and A. Gammerman, “Testing exchangeability on-line,” in *20th International Conference on Machine Learning (ICML)*, 2003.

[69] V. Fedorova, A. J. Gammerman, I. Nouretdinov, and V. Vovk, “Plug-in martingales for testing exchangeability on-line,” in *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[70] D. Volkhonskiy, E. Burnaev, I. Nouretdinov, A. Gammerman, and V. Vovk, “Martingales for changepoint detection,” in *Workshop on Conformal and Probabilistic Prediction and Applications*, 2017.

[71] S. 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 (CVPR)*, 2016.

[72] S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, and D. Song, “Anomalous instance detection in deep learning: A survey,” *arXiv preprint*, 2020.

[73] W. Lawson, E. Bekele, and K. Sullivan, “Finding anomalies with generative adversarial networks for a patrolbot,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

[74] J. Guo, G. Liu, Y. Zuo, and J. Wu, “An anomaly detection framework based on autoencoder and nearest neighbor,” in *15th International Conference on Service Systems and Service Management (ICSSSM)*, IEEE, 2018.

[75] I. Golan and R. El-Yaniv, “Deep anomaly detection using geometric transformations,” in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[76] T. DeVries and G. W. Taylor, “Learning confidence for out-of-distribution detection in neural networks,” *arXiv preprint*, 2018.

[77] K. Lee, K. Lee, H. Lee, and J. Shin, “A simple unified framework for detecting out-of-distribution samples and adversarial attacks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[78] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, “On detecting adversarial perturbations,” *arXiv preprint*, 2017.

[79] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, “Pixeldefend: Leveraging generative models to understand and defend against adversarial examples,” *arXiv preprint*, 2017.

[80] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, “Conditional image generation with pixelcnn decoders,” in *Advances in neural information processing systems (NIPS)*, 2016.

[81] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, “Detecting adversarial samples from artifacts,” *arXiv preprint*, 2017.

[82] F. Carrara, F. Falchi, R. Caldelli, G. Amato, R. Fumarola, and R. Becarelli, “Detecting adversarial example attacks to deep neural networks,” in *15th International Workshop on Content-Based Multimedia Indexing*, 2017.

[83] Z. Zheng and P. Hong, “Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[84] S. Ma and Y. Liu, “Nic: Detecting adversarial samples with neural network invariant checking,” in *26th Network and Distributed System Security Symposium (NDSS)*, 2019.

[85] X. Sun, Z. Yang, C. Zhang, K. V. Ling, and G. Peng, “Conditional gaussian distribution learning for open set recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[86] A. Bendale and T. E. Boult, “Towards open set deep networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[87] Z. Ge, S. Demyanov, and R. Garnavi, “Generative openmax for multi-class open set classification,” in *British Machine Vision Conference (BMVC)*, 2017.

[88] L. Shu, H. Xu, and B. Liu, “DOC: deep open classification of text documents,” in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[89] A. Sedlmeier, T. Gabor, T. Phan, L. Belzner, and C. Linnhoff-Popien, “Uncertainty-based out-of-distribution classification in deep reinforcement learning,” in *12th International Conference on Agents and Artificial Intelligence (ICAART)*, 2020.

[90] A. M. Nguyen, J. Yosinski, and J. Clune, “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[91] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling network architectures for deep reinforcement learning,” in *33nd International Conference on Machine Learning (ICML)*, 2016.

[92] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” in *Workshop on the 2nd International Conference on Learning Representations*, 2014.

[93] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[94] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," in *Workshop Track Proceedings 3rd International Conference on Learning Representations*, 2015.

[95] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.

[96] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[97] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[98] M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. J. Ackel, U. Muller, P. Yeres, and K. Zieba, "Visualbackprop: Efficient visualization of cnns for autonomous driving," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[99] V. Balasubramanian, S.-S. Ho, and V. Vovk, *Conformal Prediction for Reliable Machine Learning: Theory, Adaptations and Applications*. Morgan Kaufmann Publishers Inc., 2014.

[100] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations (ICLR)*, 2014.

[101] L. Ruff, N. Görnitz, L. Deecke, S. A. Siddiqui, R. A. Vandermeulen, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *35th International Conference on Machine Learning (ICML)*, 2018.

[102] Z. Qi, S. Khorram, and F. Li, "Visualizing deep networks by optimizing with integrated gradients," in *34th AAAI Conference on Artificial Intelligence*, 2020.

[103] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: an open urban driving simulator," in *1st Annual Conference on Robot Learning (CoRL)*, 2017.

[104] S. Agarwal, A. Vora, G. Pandey, W. Williams, H. Kourous, and J. McBride, "Ford multi-av seasonal dataset," *arXiv preprint*, 2020.

[105] A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, "Attacking vision-based perception in end-to-end autonomous driving models," *Journal of Systems Architecture*, p. 101766, 2020.

[106] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., 1993.

[107] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations (ICLR)*, 2016.

[108] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *arXiv preprint*, 2016.

[109] C. Yan, C. Wang, X. Xiang, Z. Lan, and Y. Jiang, "Deep reinforcement learning of collision-free flocking policies for multiple fixed-wing uavs using local situation maps," *IEEE Transactions on Industrial Informatics*, 2021.

[110] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. The MIT Press, 2009.

[111] M. Sugiyama, M. Krauledat, and K.-R. MÃžller, "Covariate shift adaptation by importance weighted cross validation," *Journal of Machine Learning Research*, vol. 8, no. May, pp. 985–1005, 2007.

[112] P. Vorburger and A. Bernstein, "Entropy-based concept shift detection," in *6th International Conference on Data Mining*, 2006.

[113] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[114] "Phm data challenge." http://www.phmsociety.org/competition/09, 2009. Accessed: 2009-09-28.

[115] Q. Zhao, E. Adeli, N. Honnorat, T. Leng, and K. M. Pohl, "Variational autoencoder for regression: Application to brain aging analysis," in *22nd International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2019.

[116] O. Rybkin, K. Daniilidis, and S. Levine, "Simple and effective VAE training with calibrated decoders," in *38th International Conference on Machine Learning (ICML)*, 2021.

[117] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot - network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[118] I. Ullah and Q. H. Mahmoud, "A scheme for generating a dataset for anomalous activity detection in iot networks.," in *Canadian Conference on AI*, 2020.

[119] "Getting started with gans part 2: Colorful mnist." https://www.wouterbulten.nl/blog/tech/getting-started-with-gans-2-colorful-mnist/.

[120] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Workshop of the 25th Annual Conference on Neural Information Processing Systems*, 2011.

[121] G. Chen, L. Qiao, Y. Shi, P. Peng, J. Li, T. Huang, S. Pu, and Y. Tian, "Learning open set network with discriminative reciprocal points," in *European Conference on Computer Vision (ECCV)*, August 2020.

[122] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[123] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *26th Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.

[124] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations (ICLR)*, 2015.

[125] D. Hendrycks, M. Mazeika, and T. G. Dietterich, "Deep anomaly detection with outlier exposure," in *7th International Conference on Learning Representations (ICLR)*, 2019.

[126] P. Oza and V. M. Patel, "C2AE: class conditioned auto-encoder for open-set recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[127] A. Makhzani, J. Shlens, N. Jaitly, and I. Goodfellow, "Adversarial autoencoders," in *Workshop of the 4th International Conference on Learning Representations*, 2016.

[128] F. Cai and X. D. Koutsoukos, "Real-time out-of-distribution detection in learning-enabled cyber-physical systems," in *11th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2020.

[129] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[130] G. Chen, P. Peng, X. Wang, and Y. Tian, "Adversarial reciprocal points learning for open set recognition," *arXiv preprint*.

[131] S. K. Khaitan and J. D. McCalley, “Design techniques and applications of cyberphysical systems: A survey,” *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, 2014.

[132] M. Zhou, Z. Zhang, and L. Xie, “Permutation entropy based detection scheme of replay attacks in industrial cyber-physical systems,” *Journal of The Franklin Institute*, vol. 358, no. 7, pp. 4058–4076, 2021.

[133] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, “Robust anomaly detection for multivariate time series through stochastic recurrent neural network,” in *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 2019.

[134] D. Ha and J. Schmidhuber, “Recurrent world models facilitate policy evolution,” in *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018.

[135] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint*, 2016.

[136] C. BISHOP, “Mixture density networks.,” *Technical Report*, 1994.

[137] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, “Lessons learned from the chameleon testbed,” in *USENIX Annual Technical Conference*, 2020.