Interpretable Detection of Distribution Shifts in Learning Enabled Cyber-Physical Systems

Yahan Yang
yangy96@seas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania, USA

Ramneet Kaur
ramneetk@seas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania, USA

Souradeep Dutta
duttaso@seas.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania, USA

Insup Lee
lee@cis.upenn.edu
University of Pennsylvania
Philadelphia, Pennsylvania, USA

ABSTRACT

The use of learning based components in cyber-physical systems (CPS) has created a gamut of possible avenues to use high dimensional real world signals generated from sensors like camera and LiDAR. The ability to process such signals can be largely attributed to the adoption of high-capacity function approximators like deep neural networks. However, this does not come without its potential perils. The pitfalls arise from possible over-fitting, and subsequent unsafe behavior when exposed to unknown environments. One challenge is that, in high dimensional input spaces it is almost impossible to experience enough training data in the design phase. What is required here, is an efficient way to flag out-of-distribution (OOD) samples that is precise enough to not raise too many false alarms. In addition, the system needs to be able to detect these in a computationally efficient manner at runtime. In this paper, our proposal is to build good representations for in-distribution data. We introduce the idea of a memory bank to store prototypical samples from the input space. We use these memories to compute probability density estimates using kernel density estimation techniques. We evaluate our technique on two challenging scenarios : a self-driving car setting implemented inside the simulator CARLA with image inputs, and an autonomous racing car navigation setting, with LiDAR inputs. In both settings, it was observed that a deviation from in-distribution setting can potentially lead to deviation from safe behavior. An added benefit of using training samples as memories to detect out-of-distribution inputs is that the system is interpretable to a human operator. Explanation of this nature is generally hard to obtain from pure deep learning based alternatives. Our code for reproducing the experiments is available at *https://github.com/yangy96/interpretable_ood_detection.git*

KEYWORDS

anomalous inputs, autonomy, safety, vision, perception systems

1 INTRODUCTION

Learning-enabled components (LEC) are being increasingly used in modern cyber-physical systems (CPS) especially self-driving cars. Designing such systems typically rely on data-driven techniques to achieve desired performance. An LEC learns to operate by having access to a large corpus of human-labelled input-output data during the design phase. For instance, a learning-enabled component in a modern car [2] could be performing simple lane keeping function from the video feed it receives from the camera. The other end of the spectrum would be a fully self-driving car equipped with an automated driving system (ADS). Here, the software is in full control of the car, and is capable of making all decision concerning navigation and maneuvering the vehicle. This has the potential to dramatically reduce accidents and general vehicle safety. Additionally, such technologies have the potential to bolster the independence and mobility of seniors and those who cannot drive.

Deep Neural networks (DNNs) perform most of the the heavy lifting regarding LEC's with rich sensors like camera and LiDAR. DNNs are high-capacity function approximators, and form a fundamental building block for most machine learning applications. The downside is that DNNs do not learn to interpret an image input the way humans do. It largely happens through an effort to fit a high-capacity function to the training data by reducing its classification error rate. DNNs are complex computation graphs and can potentially have millions of nodes and parameters. This makes them particularly difficult to be analyzable by a human expert. What this translates to is that computer vision systems are prone to errors in a way that people are not. For instance it is fairly easy to come up with imperceptible changes to an input image that can fool the ADS system. It can be shown that its fairly simple to change the number on a speed limit sign, or even change a stop sign to a speed limit sign [24].

The challenge with most LECs is that it is necessarily the case that the input space is insufficiently sampled. That is, during the design phase the system does not see all the training samples in the vicinity of the operating region of the system. Hence it is almost impossible to provide rigorous guarantees on the operating limits at design time. This is where efficient runtime monitoring techniques can play a significant role in ensuring safe operations of the system. Neural networks due to the nature of the training algorithms do not perform well when pushed outside of its training zone. Even though statistical machine learning tools can provide upper bounds on the test error rate of a learnt model, they are often too conservative and do not capture a realistic error rate.

Detecting out-of-distribution (OOD) [16, 23] samples has gained widespread popularity in the recent years to counter the fragility of DNNs when operating in unknown environments. However, it could not been extended easily to a CPS setting, and can have high false alarm rates. The main contribution of this paper is a novel technique to detect anomalous inputs in real time. The idea is to build a representation system which captures the essence of in-distribution data, using only a few samples from the training set. These prototype samples are referred to as *memories* in our paper. We use well established tools from computer vision literature and combine it with kernel density estimation techniques to compute the probability density estimates. In order to improve the robustness for detecting distribution shifts, we implement a sliding window based technique to flag an alarm.

Additionally, our technique can provide interpretability using samples from the data set. A widely accepted mode of explanation for machine learning systems, is in the form of comparisons drawn between a test sample and a witness from the data set [10, 30]. In a similar fashion we can provide explanations, when a sample is detected either as in-distribution or out-of-distribution. In our experiments we consider two broad sets of case studies, one involving video inputs from camera, and the other involving LiDAR inputs. For video inputs the application is an advanced emergency braking system , and an end-to-end self-driving system introduced in [9]. We consider different sources of distribution shift such as a shift from the training weather (low precipitation), lighting conditions (day), leading obstacles (car), and clean (or non-adversarial) images. In the context of LiDAR inputs such anomalous inputs come in the form of light rays getting reflected by different surfaces in a real setting. This pushes the network outside the trusted zone causing deviation from safe behavior. In both cases, we were able to achieve good OOD detection, which could save the system from a crash.

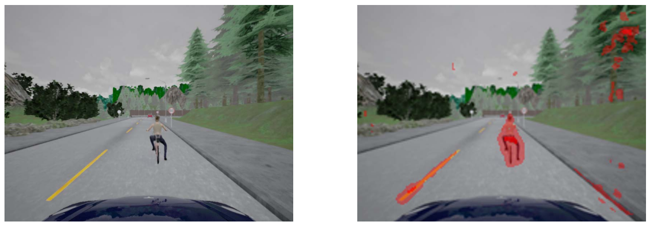
2 RELATED WORK

OOD detection has been extensively studied in the classification problem settings for standalone learning enabled components [16, 19, 20, 23, 32, 36]. These approaches either use differences in the geometrical or statistical properties of the in-distribution and OOD data for detecting a shift in the model behavior. OOD detection through envelopes, in CPS with low-dimensional input space sensors such as GPS has been studied in the past [33]. Recently, there has been a growing interest for detection of OODs in closed-loop CPS using high-dimensional sensors like camera [9, 13, 27, 31].

Cai and Koutsoukos [9] propose using reconstruction error by variational autoencoder (VAE) on the input image (or frame) as a non-conformity score in the inductive conformal anomaly detection (ICAD) framework [21] for detection of OOD frames. They further apply martingale test [34] along with the cumulative sum procedure (CUSUM) [6] with a window of the past and present predictions for robust detection of OOD traces. Ramakrishna et al. [27] use KL-divergence between the disentangled feature space of $\beta-$VAE and normal distribution as the non-conformity score in ICAD for OOD detection of a single frame. They also use martingale test along with CUSUM for detecting OOD traces. Similarly, Sundar et al. [31] also propose using KL-divergence in the latent space of $\beta-$VAE for detection of OOD frames. Feng et al. [13] propose using KL-divergence in the horizontal and vertical latent sub-space of the 3D convolutional VAE from the specified prior for detection of OOD traces. The input to 3D convolutional VAE is a sequence of frames (or the trace to be detected).

To the best of our knowledge, all the existing approaches for OOD detection in CPS with LEC are tied to VAE. Either reconstruction error from VAE on the input image or KL-divergence in the latent space of the VAE is used for OOD detection in these approaches. Training VAEs often requires careful manual tuning [5], and the quality of the training decides the efficacy of the downstream processes. Here we set ourselves apart by not having to depend on a well functioning VAE. Also, unlike our approach none of the existing approaches except for Ramakrishna et al. [27]'s approach provides interpretability on the source of OOD-ness of the input. We show that our approach can be extended to the case of LiDAR inputs as well without any conceptual modification.

3 MOTIVATION AND PROBLEM STATEMENT



(a) Car doesn't detect biker, leading to a crash

(b) Frame detected as OOD, the region in red shows the pixels responsible for deviation

Figure 1: Deviation from training data leads to a crash with biker as front object. Training data only had cars as front objects. Our proposed method could detect deviations from in-distribution data for detecting such OODs.

As mentioned before, learning enabled components have the ability to bolster the level of autonomy a cyber-physical system has to offer. Detection of OOD is one of the ways we can safe-guard systems from unwarranted behavior. In Figure 1a we show an example of a setting where the car is running an advanced emergency braking system controller. The controller uses the system states and the video feeds from the camera to sense the positions of the closest leading object on the road. The controller's job is to automatically brake the car if it crosses a certain distance threshold from the leading vehicle. What we observe is that because during training the DNN experienced just cars, it never learnt to react to bikes on the road. What happens next, is that the DNN completely misjudges a bike in the video, and ends up causing an accident. What we would like to target here is to propose a method to detect such a shift in distribution.

Problem Formulation : In this paper, we would like to solve the problem of being able to alarm the system about distribution shifts in real-time. It is extremely challenging to sample high-dimensional inputs space in an exhaustive fashion. This would mean careful analysis of the training time in-distribution data to come up with an effective detector which can act in real time. Additionally, it is desirable that such an alarm system produces interpretable behavior. It is often the case that DNNs due to their black-box nature do not offer an explanation to their decisions. Here, we would like to take up the challenge of being able to point to an explanation when samples are in-distribution or out-of-distribution. We demonstrate this in Figure 1b, the system not only flags the image with the biker ahead as OOD, but selects a set of pixels demarcating the biker to communicate as to why it decided to label it as an OOD.

4 PRELIMINARIES

4.1 Clustering with Medoids

Similar to *k-means* clustering, we wish to form partitions of the data into distinct groups or clusters. Clustering with *k-means* is a well known tool but has its challenges when used in the context of images. An issue with *k-means* is that it can potentially produce virtual cluster centers which are absent in the original data set. This is essentially because a simple mean of two (or more) images might not correspond to a real image. The other issue being that it is often susceptible to outliers in the data. Hence, we restrict ourselves to partitioning around points which are present in the data. The algorithm which achieves this is PAM [3], which is short for *Partitioning Around Medoids*. Intuitively the algorithm tries to search for centrally located objects called *medoids*, and are used to define the cluster boundaries in a nearest medoid sense.

Let us assume that the set $\mathcal{S}$ is equipped with a distance metric $\mathcal{D} : (s_1, s_2) \longrightarrow \mathbb{R}$, for $s_i \in \mathcal{S}$ and $n = |\mathcal{S}|$. Given a data set $\mathcal{S}$, PAM tries to select a set of r medoids - $M_r : \{m_1, m_2, \ldots, m_r\}$ such that the following cost is minimized,

$$Cost(M_r) = \sum_{i=1}^{n} \underset{m_j \in M_r}{min} \mathcal{D}(m_j, s_i) \tag{1}$$

We assume that the inner minimization is always possible, and we are able to break ties arbitrarily among distinct members of the set $\mathcal{S}$.

Algorithms : The challenge with PAM is that the naive implementation has a runtime complexity of $O(n^2 r^2)$ [28]. Even though there exists faster variants, but is still largely inaccessible for applications at the scale of image data sets generated from autonomous driving scenarios. In order to circumvent this challenge we introduce a variant of the *Clustering Large Applications based upon Randomized Search* (CLARANS) [25] algorithm in Section 5.2. It combines randomized global search with local cluster improvement method to improve the quality of clustering. The medoids identified by minimizing the objective in Equation 1, are referred to as *memories* from here on.

4.2 Kernel Density Estimation

Kernel Density Estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable. Let us assume that the set $\mathcal{S} = (s_1, s_2, \ldots, s_n)$ is independently and identically drawn from a fixed but unknown distribution f, and we wish to estimate the probability density for an element x. This according to kernel density estimation methods it is given by the following equation,

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(|x - s_i|) \tag{2}$$

Where K is the Kernel function, and h is a smoothing term. The kernel function measures the influence of sample s_i on the query point x. The choice of kernel, smoothing parameter, and distance function influences the quality of estimates that one obtains from this method. We refer the reader to [29] for further details on KDE.

4.2.1 Abstract Kernel Centers. Standard KDE is often difficult to estimate at runtime. Placing demands on both memory requirements and computational efficiency. To handle this, we use the concept of *abstract kernel centers* introduced in [26]. Instead of using the full data set, the idea is to use a smaller representative set to estimate the probability density. In order to obtain useful density estimates, the first step is to identify centrally located data points called *kernel centers*, which are able to capture the distribution closely. Let us denote these kernel centers with the set $\mathcal{M}_r : (m_1, m_2, \ldots, m_r)$. The density estimates are computed using the k-nearest-neighbors (kNN) from the test point x in the set $\mathcal{M}_r$. Since, the kernel centers are identified using the partitioning algorithm mentioned above, we can alternately call it the k nearest memories as well. Suppose this set of k nearest memories is $\mathcal{M}_k \subset \mathcal{M}_r$. The weighted kernel density estimate [15] is given by,

$$\hat{f}_A(x) = \sum_{j=1}^{k} w_j K_h(|x - m_j|) \tag{3}$$

$$w_j = \frac{\#\text{ points with closest memory as } m_j}{\#\text{ points which have closest memory in } \mathcal{M}_k}$$

Thus, the data partitions built around the memories can now be used to compute the density estimation function $\hat{f}_A$. In this paper we use the *Epanechnikov* kernel. Since, this kernel permits us to demarcate a boundary around the center beyond which the distance function stops being meaningful. The bandwidth parameter h can be chosen depending on the application. Note that, for us the kernel centers are not *abstract* as compared to [26], but in our case we keep the terminology to be coherent with the original idea.

4.3 Structural Similarity Index Metric

A fundamental challenge in dealing with images is to capture human perceptual similarity with a mathematically meaningful distance function. Techniques like KDE necessitate the use of a distance metric to compute the probability density. To the best of our knowledge the right candidate for this purpose is the *structural similarity index metric* (SSIM). This was first introduced in [35], and has gained widespread popularity. It computes the degree to which two images are similar to a human eye, and was used to compute the degradation quality of the image. SSIM metric is designed to capture statistical similarity between images. This makes our system more robust to random noise in comparison to vanilla DNNs. It also has been used to capture image similarity for adversarial sticker attacks as well [22].

We state the original SSIM distance function next. Assume we have two images $\mathsf{A}_1 \in \mathbb{R}^N$ and $\mathsf{A}_2 \in \mathbb{R}^N$. This allows us to compute three terms : a luminance distortion term, a contrast distortion term, and a correlation term.

$$l(\mathsf{A}_1, \mathsf{A}_2) = \frac{2\bar{\mathsf{A}}_1\bar{\mathsf{A}}_2 + c_1}{\bar{\mathsf{A}}_1^2 + \bar{\mathsf{A}}_2^2 + c_1} \tag{4}$$

$$c(\mathsf{A}_1, \mathsf{A}_2) = \frac{2s_{\mathsf{A}_1}s_{\mathsf{A}_2} + c_2}{s^2_{\mathsf{A}_1} + s^2_{\mathsf{A}_2} + c_2} \quad (5)$$

$$s(\mathsf{A}_1, \mathsf{A}_2) = \frac{s_{\mathsf{A}_1,\mathsf{A}_2} + c_3}{s_{\mathsf{A}_1}s_{\mathsf{A}_2} + c_3} \quad (6)$$

Where $\bar{\mathsf{A}_1}, \bar{\mathsf{A}_2}, s^2_{\mathsf{A}_1}, s^2_{\mathsf{A}_2}$ and $s_{\mathsf{A}_1,\mathsf{A}_2}$ are the local mean, local variance, and local covariance between A_1 and A_2. The scalar terms c_1, c_2, c_3 aim to capture the saturation effects of the visual system, and provide numerical stability. The terms computed above capture the local difference in some chosen window in the image. The combination across all such local windows gives the SSIM index. With $c_3 = c_2/2$, SSIM index can be written in the following form :

$$\begin{aligned} SSIM(\mathsf{A}_1, \mathsf{A}_2) &= S_1(\mathsf{A}_1, \mathsf{A}_2)S_2(\mathsf{A}_1, \mathsf{A}_2) \\ S_1(\mathsf{A}_1, \mathsf{A}_2) &= l(\mathsf{A}_1, \mathsf{A}_2) \\ S_2(\mathsf{A}_1, \mathsf{A}_2) &= c(\mathsf{A}_1, \mathsf{A}_2)s(\mathsf{A}_1, \mathsf{A}_2) \end{aligned} \quad (7)$$

SSIM can be implemented efficiently in tools like Pytorch [1] and accelerated using a GPU. This permits a scalable and efficient implementation inside our OOD detection framework. On the downside, SSIM does not have the mathematical properties to be a distance *metric*. But with some modifications it can be turned into one. The details of this modification and the associated proof can be found in [8]. We use the modified SSIM to define a distance metric $\mathcal{D}(A_1, A_2)$ in this paper and hyperparameters (e.g. c_1, c_2, c_3) are set same as in [1]. The use of a proper distance metric for images allows us to compute the probability density function, and subsequently distribution shifts in a more meaningful way.

5 METHODOLOGY

5.1 Initializing the Memory Set

Algorithm 1 Generate Initial Memories

```
Input: Data set $\mathcal{S} : \{s_1, s_2, \ldots, s_n\}$
Output: Memories $M : \{m_1, m_2, m_3, \ldots, m_r\}$
Parameter : Distance Threshold $d$
1: $M = \phi$
2: RejectedSet = $\mathcal{S}$
3: while RejectedSet $\neq \phi$ do
4:     $s_m$ = pickRandomPoint(RejectedSet)
5:     for $s_i \in$ RejectedSet do
6:         if $\mathcal{D}(s_m, s_i) < d$ then
7:             RejectedSet = RejectedSet $\setminus s_i$
8:     $M = M \cup \{s_m\}$
9: return $M$
```

The intuition here is that high dimensional data like images, and LiDAR scans generated from a real world setting, cluster well in practice. Hence, the first step is to identify these broad categories in a quick and efficient fashion. One of the questions however is that the number of partitions to be made is often not known apriori. But drawing on the intuitions from an image distance metric, only small enough distances have perceptual meaning. Thus, the intuition here is to populate the input space densely enough with memories such that, every training point is within a threshold distance d of some memory. Algorithm 1 summarizes our approach. We pick a data point at random, and compute the distance score across all the samples in the currently *RejectedSet* in a single linear pass. The data points which are *similar* enough are admitted as being close to a *memory*, and are not considered as candidates for new memories in the next iteration. We continue this process until all data points are admitted into the set of memories M. This allows the subsequent algorithms to have a warm start. Algorithm 1 always terminates. This is because the *RejectedSet* decreases by at least 1 at each step. In the worst case we have as many memories as the number of data points. But in most practical datasets this is not the case.

5.2 Learning Memories

To restate, we are given a dataset $\mathcal{S}$, with n elements, and we wish to compute an r size memory set $M = \{m_1, m_2, \ldots, m_r\}$ with certain desirable properties. The search for memories can be simplified by viewing this as a search through a graph $\mathcal{G}$ [25] with subsets $\mathcal{S}_r \subset \mathcal{S}$ as its nodes. Each subset of size r defines a choice for the memory set M.

Definition 5.1 (Memory Search Graph $\mathcal{G}$). The undirected graph $\mathcal{G}$ is represented by an ordered pair (V, E). The set of nodes V is the collection of subsets of original dataset $\mathcal{S}_r \subset \mathcal{S}$. An edge $e \in E$ exists between two nodes $\mathcal{S}^1_r$ and $\mathcal{S}^2_r$ iff $|\mathcal{S}^1_r \cap \mathcal{S}^2_r| = r - 1$. That is, they differ by at most one memory.

Each node of the graph has an associated cost given by Equation 1. Hence starting from some node it is possible to visit neighboring nodes with decreasing costs in the search process. What we present next is a combination of *Global* resets and *Local* minimization to approximate the optimal choice.

Algorithm 2 Generate Memories

```
Input: $\mathcal{S} : \{s_1, s_2, \ldots, s_n\}$
Output: Memories $M : \{m_1, m_2, m_3, \ldots, m_r\}$
Parameter : (Max Global Steps : $Z_g$ , Max Local Steps : $Z_l$,
Distance Threshold $d$)
1: BestCost = $\infty$
2: for $1 \leq g \leq Z_g$ do
3:     Memory Set $M$ = GenerateInitialMemories($\mathcal{S}, d$)
4:     $v$ = FindNode($M, \mathcal{G}$)
5:     $\mathcal{G}$ = CreateGraph($\mathcal{S}$, $|M|$)        ▷ The memory search graph
6:     CurrentCost = ComputeCost($v$)
7:     for $1 \leq l \leq Z_l$ do
8:         $v'$ = PickNeighbor($v, \mathcal{G}$)
9:         NewCost = ComputeCost($v'$)
10:        if NewCost < CurrentCost then
11:            $v \leftarrow v'$
12:            CurrentCost $\leftarrow$ NewCost
13:    if CurrentCost < BestCost then
14:        BestCost = CurrentCost
15: return $M$
```

Algorithm 2 picks the eventual memories used in OOD detection. Similar to standard *CLARANS* algorithm each node in $\mathcal{G}$ has $r(n-r)$ neighbors, where r is the number of memories. Which can be quite large given the scale of modern machine learning data sets with large n. What we do here is that start with a reasonable choice

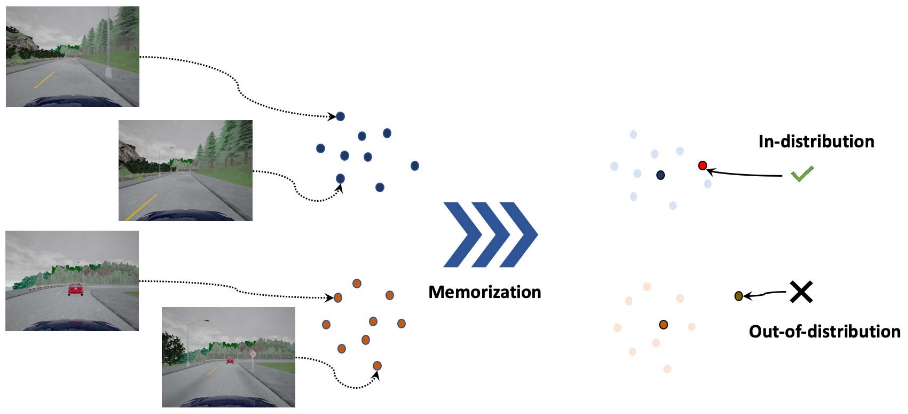


Figure 2: This figure summarizes our approach. The memorization phase of the algorithm picks prototypical samples as memories. At run-time the algorithm computes kernel density estimates to assess the likelihood of a new data being in-distribution.

for initial node in $\mathcal{G}$, and greedily look for local improvements for a fixed number of iterations. The global search starts by using Algorithm 1, in order to generate the initial set of memories as node v in $\mathcal{G}$. Notice that we do not choose the number of memories apriori but instead gets picked as a consequence of *distance score d*. The partitioning cost for the choice of memories is computed by the function *ComputeCost* which evaluates Equation 1. Note this can be expensive since it needs a total of $r \times n$ distance computation operations. The *local* search (Lines 7 – 12) implements a greedy strategy to pick the neighborhood node which produces a descent. The outer loop of the algorithm keeps track of the node with the minimum cost for each such reset produced in line 3. Algorithm 2 trivially terminates, since each search proceeds for a fixed number of steps.

Definition 5.2 (Memory System $\mathcal{M}_{\mathcal{S}}$). A memory system is a collection of pairs $\mathcal{M}_{\mathcal{S}} := \{(m_1, q_1), (m_2, q_2), \ldots, (m_r, q_r)\}$, where,

$$q_i = |Q_{m_i}|$$
$$Q_{m_i} = \bigcup_{s \in \mathcal{S}} \mathbb{I}(m_i = \underset{i}{argmin} \mathcal{D}(s, m_i)) \tag{8}$$

Thus the memory system $\mathcal{M}_{\mathcal{S}}$ keeps track of the number of points for every memory which belongs to the cluster defined by it. The OOD detection algorithm which follows uses these memories as abstract kernel centers in Equation 3 to compute the probability density at a test point.

5.3 Scaling Memory Search

In order to compute the probability density estimates given by Equation 3, we need to do a linear time search through the current set of memories in $\mathcal{M}_{\mathcal{S}}$. Even though the number of memories produced in Algorithm 2 might be small enough compared to the full data set, $\mathcal{S}$ a search through the list of memories might still be challenging. To remedy this potential drawback we deploy a simple hashing technique first introduced in [14] . The distance metric $\mathcal{D}$ discussed in Section 4.3, was a proper distance metric, which implies that the distance function respects triangle inequality. In what follows, we describe a possible avenue to speed up the search for the k nearest memories. The intuition being that for sufficiently different memories computing a single distance pair can be used to reject other memories from further consideration.

We are interested in computing the nearest neighbor, that is $k = 1$ in the set $\mathcal{M}_{\mathcal{S}}$ for a test point x_t. Assume that we wish to compute the distance between a test point x_t, and some memory m_j and the distance $\mathcal{D}(x_t, m_i)$ is known. Then in the triangle formed by the triplet (m_i, x_t, m_j), the following two equations are true:

$$\mathcal{D}(m_i, x_t) - \mathcal{D}(m_i, m_j) \leq \mathcal{D}(m_j, x_t)$$

and

$$\mathcal{D}(m_i, m_j) - \mathcal{D}(m_i, x_t) \leq \mathcal{D}(m_j, x_t)$$

Meaning that $\mathcal{D}(m', x_t)$ is lower bounded by : $|\mathcal{D}(m_i, m_j) - \mathcal{D}(m_i, x_t)|$. Since, if we are interested in memories which are within a certain threshold (say h) of m', we do not actually need to compute the distance $\mathcal{D}(m', x_t)$ if the following equation holds *True*.

$$|\mathcal{D}(m_i, m_j) - \mathcal{D}(m_i, x_t)| > h \tag{9}$$

For each memory m_i, we can pre-compute a look-up table for the inter-memory distance $Q : \{(m_j, \mathcal{D}(m_i, m_j))|1 \leq j \leq l, j \neq i\}$. This can lead to reduction in the search space in practice by pruning out memories from further consideration each time the distance of a memory from x_t gets measured. For $k > 1$, similar reasoning holds. The only difference being that, in this case the search algorithm tracks the distance of the k^{th}-memory furthest from the test point.

5.4 Detecting Distribution Shifts

To summarize, we know how to go from the set of training data $\mathcal{S}$ to the set of memories $\mathcal{M}_{\mathcal{S}}$. This happens through a smart initialization of the set of memories (Algorithm 1), followed by a further refinement using a medoid based partitioning technique discussed in Algorithm 2. Additionally, to handle any potential slow downs, we briefly discuss how one can use the inter-memory distance to prune out large parts of the search space. Thus allowing the system to scale to larger memory systems. We are now at a stage to discuss our runtime algorithm for detecting distribution shifts.

Algorithm 3 Detect Distribution Shifts

Input: Time Series Data x_t, Memory System $\mathcal{M}_{\mathcal{S}}$
Output: Distribution Shift Flags $\mathcal{F}_t$
Parameter : Window Threshold - τ, Window - W, probability threshold - α

```
1: Frame = $\phi$
2: for $1 \leq t \leq \infty$ do
3:     $Flag_{OOD}$ = DetectOOD($\mathcal{M}_{\mathcal{S}}$, $x_t$, $\alpha$)
4:     Frame ← UpdateFrame($Flag_{OOD}$, $W$, Frame)
5:     $\mathcal{F}_t$ ← CountOOD(Frame) ≥ $\tau$
6: return $\mathcal{F}_t$
```

In practical scenarios, detecting a shift in distribution needs a robust mechanism. We achieve this using a sliding window based implementation to track the number of out of distribution samples it sees. Algorithm 3 summarizes our approach. For each input sample at runtime, the function *DetectOOD* simply computes the probability estimates for a test point x_t, from a memory system $\mathcal{M}_{\mathcal{S}}$ using Equation 3. Additionally it compares this probability density with a threshold α to *Flag* a sample as OOD. The function *UpdateFrame* keeps track of the OOD Flags in the last W frames. The algorithm outputs a distribution shift once this count cross threshold τ.

6 CASE STUDY 1 - SIMULATED AUTONOMOUS DRIVING SCENARIO USING CARLA

System Description: Here we consider an advanced emergency braking system (AEBS) from [9]. The system overview is shown in Figure 11 of A.2. It is a closed loop system composed of a perception based LEC, which estimates distance of the object ahead of the ego vehicle. This distance combined with the velocity is the input to the braking controller. The objective of the AEBS system is to brake the ego vehicle to avoid a collision. The controller is trained using standard reinforcement learning on in-distribution data. The distance estimating LEC is trained using supervised learning techniques. For details about the model architecture and training hypeparameters, please refer to [9].

In-distribution data: We evaluate our approach on OOD detection with the dataset provided in [9]. The dataset is generated using CARLA [11], an open-source simulator for autonomous driving. The in-distribution traces consists of daytime frames with slight rain (i.e. precipitation level in $\{0, 1, \ldots 10\}$), and with cars as the front object. The sampling rate is $20Hz$ [9].

Types of OODs: We evaluate our approach on the following four different sources of OOD-ness in the traces.

(1) OOD-ness due to weather change from slight rain in the in-distribution traces to heavy rain (precipitation level greater than or equal to 20) and foggy traces in OOD traces.
(2) OOD-ness due to change in the lighting conditions from day in the in-distribution traces to night in the OOD traces.
(3) OOD-ness due to change in the front obstacles from cars in the in-distribution traces to bikes in the OOD traces.
(4) OOD-ness due to perturbation of in-distribution frames with adversarial attack.

Evaluation metrics: We refer to OOD traces as positive and in-distribution as negative. We report false positive (FP) as the number of in-distribution traces that were falsely detected as OOD. False negatives (FN) are the number of OOD traces falsely detected as in-distribution. We also report an average delay in the OOD detection as the number of windows required to detect the start of the OOD-ness in the traces averaged over the total number of detected OOD traces. We conduct all our experiments in this case study on a single GPU (NVIDIA GeForce RTX 2080 Ti).

6.1 OOD-ness due to change in weather and lighting

Here we generate OOD traces in which OOD-ness gradually increases with time. We increase the precipitation (or fog) parameter gradually in sequential frames of a trace to generate heavy rain (or foggy) OOD trace. Similarly, we gradually increase the darkness parameter to generate night time OOD traces. Examples of these OOD traces are shown in Appendix.

6.1.1 Results on OOD detection for heavy rain. We define frames with precipitation level greater than 20 as OODs due to heavy rain. There are 4488 in-distribution images with precipitation parameter from 0 to 10. The test dataset contains 26 in-distribution and 74 out-of-distribution traces. Our approach involves a few hyperparameters like window length W, threshold count τ, probability threshold α, and distance threshold d. In the current experiments, we choose the hyperparameters empirically. In practice, the desired sensitivity of the system would dictate the parameter choice. We report some of the top performers in Table 1. A more exhaustive study has been reported in Figure 10 of the Appendix.

(W, τ, α, d)	Mem	FP	FN	Avg Delay	Exec Time (ms)
(5,5,0.92,0.2)	145	**0/26**	2/74	0.42	21.39
(10,5,0.92,0.2)	145	**0/26**	2/74	0.04	19.80
(5,5,0.78,0.3)	**36**	1/26	1/74	**0**	6.26
(10,5,0.78,0.3)	**36**	1/26	**0/74**	**0**	**5.88**

Table 1: Results on heavy rain traces as OODs. *Mem*: **Total number of memories,** *Exec time*: **the time for OOD detection on each frame (calculated on 3 random traces)**

The experimental results shows that Algorithm 2 efficiently compressed the training data (4488→ 145/36) into memories and can perform OOD detection under autonomous driving setting. With the parameters as shown in the table, the results in Table 1 indicate that our OOD detector could successfully detect traces with heavy rain precipitation and without delay. Performance of the VAE based OOD detector [9] is comparable to ours in terms of false positive and false negative rates but their average detection delay (as reported) is higher than ours. We also report the average execution

	Night traces		Foggy traces	
(W, τ, α, d)	**FN**	**Avg Delay**	**FN**	**Avg Delay**
(5,5,0.92,0.2)	**0/27**	**0.15**	**0/27**	**5**
(10,5,0.92,0.2)	**0/27**	**0.15**	**0/27**	10
(5,5,0.78,0.3)	**0/27**	1.89	**0/27**	9
(10,5,0.78,0.3)	**0/27**	**0.15**	**0/27**	11.15

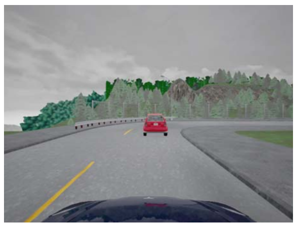
Table 2: Results on night and foggy OOD traces.

times for detecting an OOD in Table 1. We observe that it is well within the the sampling period of the system. Implying that Algorithm 3 is amenable to real-time OOD detection. Also, as expected, reducing d in Algorithm 1 results in higher memories. But results in slower execution times with better false positive rates.

6.1.2 Results on detection for foggy and night OODs. Table 2 shows results of OOD detection on foggy and night OODs. Here we consider 27 OOD traces for both settings. With the same hyperparameters as in the heavy rain OOD traces, our detector is able to detect all OOD traces.

6.2 OOD-ness due to change in front obstacles

One of the motivations for building an OOD framework is that it is often the case that unobserved data during training may lead to *crash*. The perception LEC only saw cars as front obstacles during its training. At test time, ego vehicle is able to stop at a safe distance from the front obstacle if the obstacle is a car (Figure 3(a)). But if we change the front obstacle from car to bike then it leads to a crash (Figure 3(b)). We generated 27 OOD traces with different positions and types of bikes as front obstacles and all of these traces lead to a crash with the biker.


(a) Ego vehicle stopping at a safe distance from the lead car at test time

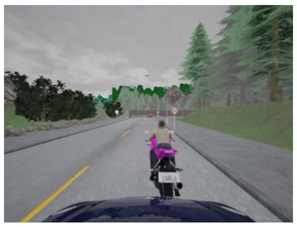
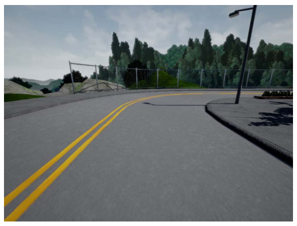

(b) Shift from training distribution with a biker as front obstacle leads to a crash at test time

Figure 3: Illustration of safety hazard, i.e. collision due to shift in the training distribution

6.2.1 Results. We define the OOD frame starting from time-step 20 in the traces (when the biker becomes visible to human). We use the same set of hyperparameters in Section 6.1. As shown in Table 3, our OOD detector could successfully alarm the system before a collision happens for all the 27 OOD traces for two hyperparameter settings. For the other two settings, we could not detect 2 out of 27 OOD traces.

(W, τ, α, d)	FP	FN	Avg Delay
(5,5,0.92,0.2)	**0/26**	**0/27**	**0**
(10,5,0.92,0.2)	**0/26**	**0/27**	**0**
(5,5,0.78,0.3)	1/26	2/27	0.96
(10,5,0.78,0.3)	1/26	2/27	**0**

Table 3: Results on OOD traces with bikers.


(a) Input image (clean)

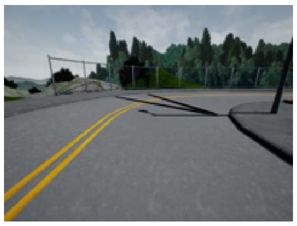

(b) Test image (adversarial sticker on the road)

Figure 4: OOD-ness due to adversarial road perturbations [9]

6.3 OOD-ness due to perturbations by adversarial attack

In these experiments, we evaluate our approach for an adversarial attack detection. Again, we consider the same attack of painting lines on the roads as considered in [9]. This attack was introduced by Boloor et al. [7] and shown that it causes the car to follow the painted lines leading to a crash.

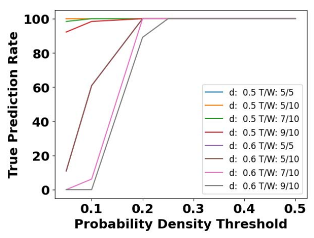
We use the same attacked dataset from [9] which focuses on Right Corner Driving case. We run our OOD detector to check whether our detector could predict crash beforehand. There are total 105 traces for tests and 69 out of them ends with a crash. Note that an attack prediction is only useful as long as it happens before the actual crash. We forecast a *crash* when a shift in distribution occurs. Let us call this t_{pc}, time when crash prediction is set to $True$. Also, let us denote the time of actual crash by t_{ac}. A crash is successfully predicted when $t_{pc} < t_{ac}$. We report our performance on the following metrics in the context of crash:

$$\begin{aligned} \text{True Prediction Rate (TPR)} &= \frac{\text{\# crash predicted successfully}}{\text{\# crash happens}} \\ \text{False Prediction Rate (FPR)} &= \frac{\text{\# no crash happens}}{\text{\# crash predicted}} \\ \text{Missed Prediction Rate (MPR)} &= \frac{\text{\# crash happens without forecast}}{\text{\# crash happens}} \end{aligned} \quad (10)$$

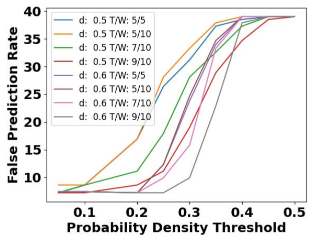
In addition, we record the average forecast time as the average value of $t_{ac} - t_{pc}$, for the correctly predicted cases, and it is reported in the number of frames. We report these numbers in Table 4

Here we also report the top performance in Table 4 using selected hyperparameters according to the Figure 5. These results show that our methodology is also successful in adversarial trace detection at least 5 frames before the crash.

6.3.1 OOD detection reasoning using SSIM. As mentioned before, an advantage of our framework is that it is interpretable to a human. For in-distribution data, it is simply the closest memory the test



(a) True prediction rate



(b) False prediction rate

Figure 5: Out-of-distribution traces detection results for detecting adversarial attack on the road with different hyperparameters

(W, τ, α, d)	Mem	TPR	FPR	MPR	Avg Forecast
(5,5,0.05,0.5)	243	**100.0**	**7.2**	**0.0**	5.08
(5,5,0.1,0.5)	243	**100.0**	8.6	**0.0**	14.78
(5,5,0.2,0.6)	**114**	**100.0**	**7.2**	**0.0**	4.89
(5,5,0.25,0.6)	**114**	**100.0**	12.3	**0.0**	**16.2**

Table 4: Results on adversarial sticker detection.

frame matches to. This happens by design due to the choice of the distance metric $\mathcal{D}$. A more interesting case arises when a test frame is recognized as an OOD. Note that a simple way to frame the reason for an OOD would be to say - it is not *similar* enough to anything in the memory system. But, here we go a step further and try to provide pixel level reasoning. This can be mined from the closest memory to an OOD sample, using modified $\mathcal{D}$ to attribute pixels responsible for the dissimilarity. Additionally, in case of scattered highlighted pixels (indicating that the test input is drawn from a distribution that is very different from the training distribution), our framework refrains from pixel attribution, and simply raises an alarm.

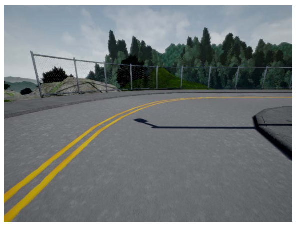
Heatmap generation: Notice in Section 4.3, the SSIM value is a mean of the dissimilarity scores for all pixels. For some pairs of images it is possible that difference is high due to a high concentration of dissimilarity scores on a few pixels. Thus it is possible to filter out these pixels if the distance is above a certain threshold in its window. We attribute these pixels responsible for higher SSIM value and highlight them in red for providing reasoning about OOD detection. The details about the heatmap generation algorithm are provided in A.4.

We demonstrate this in Figure 1b and Figure 6b. In 1b, the unrecognized biker is highlighted in this OOD frame. In the adversarial sticker experiment (7b), we can also notice that the highlighted area contains the adversarial stickers on the road.

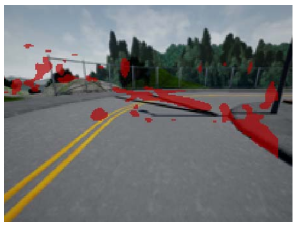
7 CASE STUDY 2 - DRIVING WITH LIDAR

7.1 System Description

LiDAR forms a fundamental component for a large section of self-driving car hardware, and is a reliable fall back option when it comes to situations where camera is not enough. LiDAR simply computes the distance of the closest obstacle in specific angles



(a) Match the input test image with memories in training data



(b) Highlight the least similar part compared to the memory

Figure 6: OOD detection reasoning for sticker detection (The test image is Figure 4b)

for a fixed range. Even though the nature of the input is of much simpler nature compared to a camera, NNs with LiDAR inputs can suffer from similar behavior when exposed to OOD scenarios. In this section we introduce the case study involving an autonomous car, discussed in [17]. Figure 8 illustrates the arrangement of the functional blocks. The system involves a car from the F1/10 Autonomous Racing Competition [4], navigating square tracks using only LiDAR measurements to judge obstacles and make general orientation decisions. The LiDAR measurements are sent to a neural network (NN) controller, which issues steering controls. It operates under a constant throttle setting for reasons discussed in [17]. The system state such as position and orientation, along with the surrounding environment determines the nature of scan that the LiDAR receives. The NN controller is trained using standard deep reinforcement learning techniques like deep-deterministic policy gradient (DDPG), and Twin Delay DDPG (TD3). The LiDAR scan obtained from the system has 1081 rays ranging from −135 degrees to 135 degrees, with 0 degrees being the heading of the car. Most of the controllers trained in [17] acted on a sub-sampled set of 21 LiDAR rays, which produced satisfactory performance in simulation. This sets the number of LiDAR rays for the experiments in this paper as well.

7.2 Simulation vs Reality

As we saw before, one of the challenges when it comes to deploying learning-enabled cyber-physical systems in the real world is the unexpected behavior caused by the sim2real gap. Even though the recent literature [12, 18] has seen an explosion of interest in verifying closed-loop systems with NN controllers, verification results make sense as long as the assumptions on the environment hold. The NN controllers for this benchmark were trained in a virtual environment with exactly the same racing track, and obstacle setting. Simulations are a useful and rich source of training data when it comes to deep reinforcement learning approaches. However, the downside is that the aberrations arising in the real world can cause the system to go berserk. In the current setting this aberration comes from the presence of reflective surfaces as shown in Figure 7. This introduces a large source of uncertainty. A LiDAR ray reflected away from a highly reflective surface, takes longer time to return to the on board detectors. Which ends up giving a false impression of no obstacle in that angle. This is hard to model since surface

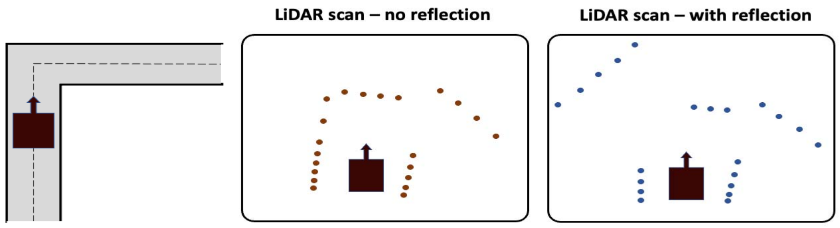


Figure 7: Left : We show a setting where the car should take a right turn on an L-shaped track. Middle : The dots show the distance estimates as provided by the sensor. It matches well with the position of the obstacles. Right : Due to reflection from the left wall, it gives a false impression of no obstacle to the left of the car when deployed in the real world.

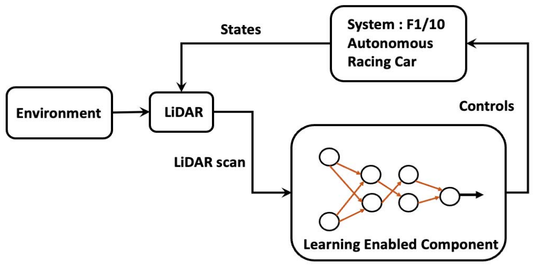

States
System : F1/10 Autonomous Racing Car
Environment
LiDAR
Controls
LiDAR scan
Learning Enabled Component

Figure 8: Functional blocks in the F1/10 Autonomous Car.

reflectivity is largely unknown. In Figure 7, this happens at the left corner of L-shaped track. This creates a false impression of no obstacle to the left of the car. The ground-truth is hard to guess just from the LiDAR inputs. But a crash could have been avoided if the car had switched to a safe mode, or raised an alarm ahead of time. In this case study, we focus on the ability to detect such OOD scenarios. The right course of action after detecting such a shift is context dependent and is beyond the scope of this work.

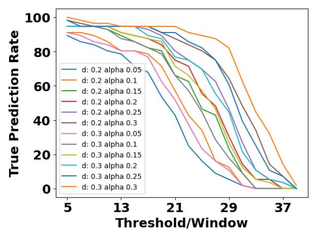
7.3 Predicting Crash

The authors in [17] report the presence of reflections as being correlated with an actual crash. Additionally, they show that getting rid of the reflections artificially can lead to safer outcomes. Hence, our hypothesis here is that, crash could be due to the potential deviation from in-distribution data, which is from simulations and does not contain any reflections. The intuition being, if an NN controller has experienced reflections during training time, then it would have known the right course of action. The LiDAR scans with reflected rays could therefore be treated as out-of-distribution data. Notice that in this case study, there is no clear distinction when things start becoming OOD. What we have instead is real crash data. We have access to the time-stamped LiDAR scan log for each such run of the system along the L-shaped trajectory in Figure 7. The data set $\mathcal{S}$ here, is a set of trajectories $\{T_1, T_2, \ldots, T_n\}$. Each trajectory is a list of time-stamped LiDAR scans, $T_i = \{(x_1, p_1), (x_2, p_2), \ldots\}$, where $x_i \in \mathbb{R}^q$ is the LiDAR scan at time i, and p_i is a flag variable for crash. What we wish to test here is whether a detector for distribution shift is a good predictor for a future crash.

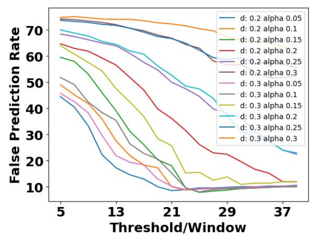
Results In order to simulate a crash prediction setting, we run our OOD detector for each LiDAR scan in the trace T_i starting from $i = 0$. The in-distribution data here is obtained by running the simulator for the 12 different controllers. These include LiDAR scans over the length of 70 time steps. Which is approximately the number of time-steps the system takes to reach from one end of the track to the other. Note that the controllers were trained well enough during simulation that none of the traces show a crash. In this experiment, we create a 2-dimensional data array by repeating the 1-dimensional measurement. The distance metric for OOD detection is the same SSIM metric applied to a LiDAR scan. The detection of distribution shift is implemented using Algorithm 3. We report our performance on the same metrics as mentioned Equation 10, in Table 5 for different choices of the parameters. In the best case we were able to predict 82.1% of the crashes with 22.7% false positive rate and ≈ 9 time steps ahead. The missed predictions rate ≈ 10%.

(W, τ, α, d)	TPR	FPR	MPR	Avg Forecast
(40,15,0.05,0.3)	80.36	**19.35**	10.71	9.69
(40,17,0.1,0.3)	**82.14**	22.73	**8.93**	9.8
(40,11,0.05,0.2)	80.36	22.22	12.5	**12.67**
(40,15,0.1,0.2)	80.36	21.88	10.71	10.4

Table 5: Results for LiDAR data



(a) True prediction rate vs threshold ($W = 40$)



(b) False prediction rate vs threshold ($W = 40$)

Figure 9: OOD detection results for detecting LiDAR crash with different hyperparameters

8 CONCLUSION

OOD detection can be of utmost importance in ensuring safety of cyber-physical systems equipped with learning enabled components. What we have achieved to demonstrate in this paper, is that state of the art results in OOD detection for self-driving car applications, can go hand in hand with overall interpretablity, without compromising on execution times. In the future, we would like to extend this technique on applications beyond self-driving cars where anomalous inputs are challenging to handle.

9 ACKNOWLEDGEMENT

This work was supported in part by ARO W911NF-20-1-0080, AFRL and DARPA FA8750-18-C-0090, NSF-1915398, NSF-2125561 and SRC Task 2894.001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Air Force Research Laboratory (AFRL), the Army Research Office (ARO), the Defense Advanced Research Projects Agency (DARPA), or the Department of Defense, or the United States Government.

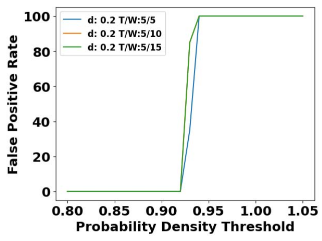
Additionally, we would like to thank Radoslav Ivanov, postdoctoral scholar from the University of Pennsylvania for discussions on the LiDAR experiments and sharing the data. We also especially appreciate Feiyang Cai, PhD candidate from Vanderbilt University for sharing the CARLA dataset and simulation code.

REFERENCES

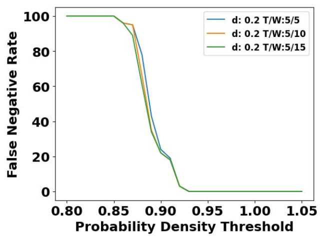
[1] [n. d.]. *pytorch-msssim.* https://pypi.org/project/pytorch-msssim/
[2] [n. d.]. *Toyota Safety Sense.* https://www.toyota.com/safety-sense/
[3] 1990. *Partitioning Around Medoids (Program PAM).* Chapter 2, 68–125. https://doi.org/10.1002/9780470316801.ch2 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470316801.ch2
[4] 2021. F1TENTH. https://f1tenth.org
[5] Alexander A. Alemi, Ben Poole, Ian Fischer, Joshua V. Dillon, Rif A. Saurous, and Kevin Murphy. 2017. An Information-Theoretic Analysis of Deep Latent-Variable Models. *CoRR* abs/1711.00464 (2017). arXiv:1711.00464 http://arxiv.org/abs/1711.00464
[6] Michele Basseville, Igor V Nikiforov, et al. 1993. *Detection of abrupt changes: theory and application.* Vol. 104. prentice Hall Englewood Cliffs.
[7] Adith Boloor, Karthik Garimella, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. 2020. Attacking vision-based perception in end-to-end autonomous driving models. *Journal of Systems Architecture* 110 (2020), 101766. https://doi.org/10.1016/j.sysarc.2020.101766
[8] Dominique Brunet, Edward R. Vrscay, and Zhou Wang. 2012. On the Mathematical Properties of the Structural Similarity Index. *IEEE Transactions on Image Processing* 21, 4 (2012), 1488–1499. https://doi.org/10.1109/TIP.2011.2173206
[9] Feiyang Cai and Xenofon Koutsoukos. 2020. Real-time out-of-distribution detection in learning-enabled cyber-physical systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS).* IEEE, 174–183.
[10] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan Su. 2019. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Edward A. Fox, and Roman Garnett (Eds.). 8928–8939. http://papers.nips.cc/paper/9095-this-looks-like-that-deep-learning-for-interpretable-image-recognition
[11] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning.* 1–16.
[12] Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2019. Sherlock - A Tool for Verification of Neural Network Feedback Systems: Demo Abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control* (Montreal, Quebec, Canada) *(HSCC '19).* Association for Computing Machinery, New York, NY, USA, 262–263. https://doi.org/10.1145/3302504.3313351
[13] Yeli Feng, Daniel Jun Xian Ng, and Arvind Easwaran. 2021. Improving Variational Autoencoder based Out-of-Distribution Detection for Embedded Real-time Applications. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 5s (2021), 1–26.
[14] K. Fukunaga and P.M. Narendra. 1975. A Branch and Bound Algorithm for Computing k-Nearest Neighbors. *IEEE Trans. Comput.* C-24, 7 (1975), 750–753. https://doi.org/10.1109/T-C.1975.224297
[15] Francisco José Gisbert. 2003. Weighted samples, kernel density estimators and convergence. *Empirical Economics* 28 (02 2003), 335–351. https://doi.org/10.1007/s001810200134
[16] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).
[17] Radoslav Ivanov, Taylor J. Carpenter, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2020. Case Study: Verifying the Safety of an Autonomous Racing Car with a Neural Network Controller. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control* (Sydney, New South Wales, Australia) *(HSCC '20).* Association for Computing Machinery, New York, NY, USA, Article 28, 7 pages. https://doi.org/10.1145/3365365.3382216
[18] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig: Verifying Safety Properties of Hybrid Systems with Neural Network Controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control* (Montreal, Quebec, Canada) *(HSCC '19).* Association for Computing Machinery, New York, NY, USA, 169–178. https://doi.org/10.1145/3302504.3311806
[19] Ramneet Kaur, Susmit Jha, Anirban Roy, Sangdon Park, Edgar Dobriban, Oleg Sokolsky, and Insup Lee. 2022. iDECODe: In-distribution Equivariance for Conformal Out-of-distribution Detection, Association for the Advancement of Artificial Intelligence. arXiv:2201.02331 [cs.LG]
[20] Ramneet Kaur, Susmit Jha, Anirban Roy, Sangdon Park, Oleg Sokolsky, and Insup Lee. 2021. Detecting OODs as datapoints with High Uncertainty. *arXiv preprint arXiv:2108.06380* (2021).
[21] Rikard Laxhammar and Göran Falkman. 2015. Inductive conformal anomaly detection for sequential detection of anomalous sub-trajectories. *Annals of Mathematics and Artificial Intelligence* 74, 1 (2015), 67–94.
[22] Juncheng Li, Frank R. Schmidt, and J. Zico Kolter. 2019. Adversarial camera stickers: A physical camera-based attack on deep learning systems. *CoRR* abs/1904.00759 (2019). arXiv:1904.00759 http://arxiv.org/abs/1904.00759
[23] David Macêdo, Tsang Ing Ren, Cleber Zanchettin, Adriano LI Oliveira, and Teresa Ludermir. 2021. Entropic out-of-distribution detection. In *2021 International Joint Conference on Neural Networks (IJCNN).* IEEE, 1–8.
[24] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2015. DeepFool: a simple and accurate method to fool deep neural networks. *CoRR* abs/1511.04599 (2015). arXiv:1511.04599 http://arxiv.org/abs/1511.04599
[25] R.T. Ng and Jiawei Han. 2002. CLARANS: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering* 14, 5 (2002), 1003–1016. https://doi.org/10.1109/TKDE.2002.1033770
[26] Xiao Qin, Lei Cao, Elke A. Rundensteiner, and Samuel Madden. 2019. Scalable Kernel Density Estimation-based Local Outlier Detection over Large Data Streams. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019.* 421–432.
[27] Shreyas Ramakrishna, Zahra Rahiminasab, Gabor Karsai, Arvind Easwaran, and Abhishek Dubey. 2021. Efficient Out-of-Distribution Detection Using Latent Space of β-VAE for Cyber-Physical Systems. *arXiv preprint arXiv:2108.11800* (2021).
[28] Erich Schubert and Peter J. Rousseeuw. 2018. Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. *CoRR* abs/1810.05691 (2018). arXiv:1810.05691 http://arxiv.org/abs/1810.05691
[29] B.W. Silverman. 2018. *Density Estimation for Statistics and Data Analysis.* 1–175 pages. https://doi.org/10.1201/9781315140919
[30] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR* abs/1312.6034 (2014).
[31] Vijaya Kumar Sundar, Shreyas Ramakrishna, Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. 2020. Out-of-distribution detection in multi-label datasets using latent space of β-vae. In *2020 IEEE Security and Privacy Workshops (SPW).* IEEE, 250–255.
[32] Jihoon Tack, Sangwoo Mo, Jongheon Jeong, and Jinwoo Shin. 2020. Csi: Novelty detection via contrastive learning on distributionally shifted instances. *arXiv preprint arXiv:2007.08176* (2020).
[33] Ashish Tiwari, Bruno Dutertre, Dejan Jovanović, Thomas de Candia, Patrick D Lincoln, John Rushby, Dorsa Sadigh, and Sanjit Seshia. 2014. Safety envelope for security. In *Proceedings of the 3rd international conference on High confidence networked systems.* 85–94.
[34] Vladimir Vovk, Ilia Nouretdinov, and Alexander Gammerman. 2003. Testing exchangeability on-line. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03).* 768–775.
[35] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612. https://doi.org/10.1109/TIP.2003.819861
[36] Ev Zisselman and Aviv Tamar. 2020. Deep residual flow for out of distribution detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 13994–14003.

A APPENDIX

A.1 Hyperparameter experiments



(a) False Positive Rate



(b) False Negative Rate

Figure 10: Out-of-distribution episode detection results for detecting OODs due to heavy rain

A.2 Overview of the AEBS system

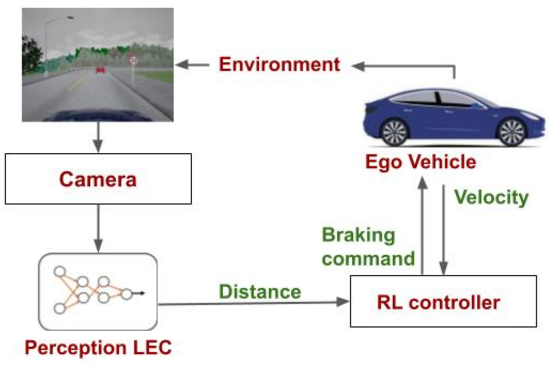


Figure 11: Closed loop of the AEBS from [9]

A.3 OODs Data Set Case Study 1

Fog dataset: the OOD frame starts from the 1st frame. The average length of foggy episodes is 123 frames.

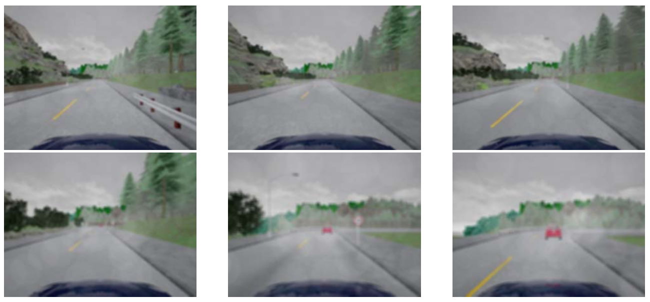


Figure 12: Example sequence in Fog Dataset (we gradually increase the level of fog)

Night dataset: the OOD frame starts from the 10th frame. The average length of night episodes is 123 frames.

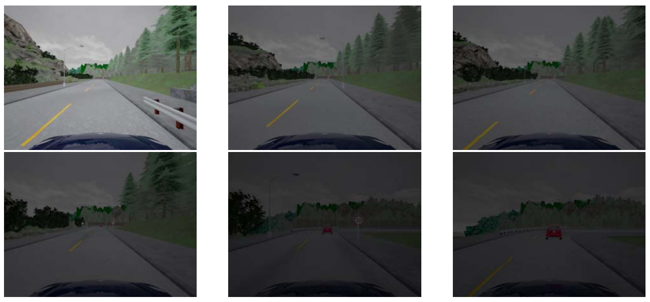


Figure 13: Example sequence in Night Dataset (we gradually increase the darkness parameter)

A.4 Heatmap Generation Algorithm

As mentioned in section 6.3.1, in addition to use the SSIM value to indicate whether the test frame is similar to each memory, we can also compute the contribution of each corresponding pixel to the SSIM distances using *ComputeFullSSIM*. *ComputeFullSSIM* computes and returns the local differences for individual pixels between two images. By highlighting the pixels with high contribution in the heatmap, we can visualize the most dissimilar parts between the test frame and its closest memory.

Algorithm 4 Heatmap Generation

```
Input: Time Series Data $x_t \in \mathbb{R}^{m \times n}$, Closest Memory $m_c$
Output: Heatmap $x'_t \in \mathbb{R}^{m \times n}$
Parameter : Color Distance Threshold $d_{color}$
1: Instantiate $x'_t \leftarrow x_t$
2: $D_x \in \mathbb{R}^{m \times n} \leftarrow$ ComputeFullSSIM($x_t$,$m_c$)
3: for $1 \leq m' \leq m$ do
4:     for $1 \leq n' \leq n$ do
5:         if $D_x[m', n'] > d_{color}$ then
6:             $x'_t[m', n'] \leftarrow$ PaintPixel($x_t[m', n']$)
7: return $x'_t$
```