Reduced Robust Random Cut Forest for Out-of-Distribution Detection in Machine Learning Models

Harsh Vardhan and Janos Sztipanovits

Abstract—**Most machine learning based regressors extract in-formation from data collected via past observations to make predictions in the future. Consequently, when input to these trained models is data with significantly different statistical properties from data used for training, there is no guarantee of accurate prediction. Consequently, using these models on out of distribution input data may result in a completely different predicted outcome from the desired one, which is not only erroneous but can also be hazardous in some cases. Successful deployment of these machine learning models in any system requires a detection system, which should be able to distinguish between out-of-distribution and in-distribution data (i.e. similar to training data). In this paper, we introduce a novel approach for this detection process using Reduced Robust Random Cut Forest (RRRCF) data-structure, which can be used on both small and large datasets. Similarly, to the Robust Random Cut Forest (RRCF), RRRCF is a structured, but reduced representation of the training data sub-space in form of cut-trees. Empirical results of this method on both low and high dimensional data showed that inference about data being in/out of training distribution can be made efficiently and the model is easy to train with no difficult hyper-parameter tuning. The paper discusses two different use-cases for testing and validating results.**

Index Terms—**Random Cut Forest, Robust Random Cut Forest, interpretable intelligence, out-of-Distribution detection, machine learning, cyber physical system.**

I. Introduction

Machine Learning (ML) models are increasingly deployed in design and operation of engineering system [1], [2], medical science [3], financial sector [4] etc. The training process of these models involve collection of data and then training on these collected data. During operation or prediction, we provide input to these trained ML models and get an output. As machine learning models are trained and validated against collected training data, their performance with new input that is not consistent with the statistical properties of the training data cannot be relied on. Therefore, it is imperative to have some mechanism to verify that a given input data is sampled from training data distribution. An Out-of-training-distribution (OOD) data point is one which is significantly different from the training data i.e., a data point which is an anomaly relative to the training data so that it may stir speculation that it was generated by a different mechanism [5]. There are three major categories of approaches to detect out of training distribution data: *statistical detection techniques*, *Deviation based techniques*, *proximity-based techniques* [6]. *Statistical detection techniques* attempt to fit the training data in a parametric/non-parametric probability distribution. The goal of learning is to find distribution model and parameters that can fit training data. Using these learned densities distribution, inference about new data point can be made based on probability of it being generated from the trained densities. A datapoint is defined as an OOD if the probability of it being generated is very low. The major limitation of this approach is difficulty to find a good probability fit, even when dimension of the problem is very small. *Deviation based techniques* are based on one or another flavor of encoder-decoder setting. The encoder is trained to embed the data to latent space and decoder is trained to reconstruct the data from latent space. During training of this model, goal is to reconstruct the input data as output of decoder. Once trained, inference about a datapoint can be made by passing it through encoder and decoder setting and estimating the reconstruction error/probability gap. The underlying principle of deviation-based methods are 'data from out-of-distribution will have high reconstruction error' as the encoder-decoder parameters are not trained for it.

Proximity based techniques assume that anomalous data is isolated from most of the data, and it use various approaches to measure density or cluster representation of training data and derive a mechanism to measure relationship of datapoint with the cluster. Our approach will fall into category of proximity-based clustering approach. In clustering-based approaches, an algorithm is deployed to represent the cluster of data from metric space to some data structure. Selection of algorithm and data-structure should be such that relationships of the data points in metric space must be preserved in this data-structure. The early work in this context is done by using randomized cut forests/isolation forests [7]. The isolation forest approach has several drawbacks, such as not compatible with streaming data and missing crucial OODs in the presence of irrelevant dimensions etc [8]. To address these challenges, [8] proposed Robust Random Cut Forest (RRCF) as data structure, which is very promising in terms of very small false alarm rate (high accuracy), and can work on streaming data, but it is not scalable on large data [9]. In this paper, we address this problem by developing a method that can use RRCF on large data sets. The main idea of our

Manuscript received July 2, 2021; revised April 10, 2022. This work is supported by DARPA's Symbiotic Design for CPS project and by the Air Force Research Laboratory (FA8750-20-C-0537). Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or AFRL.

The authors are with the Institute of Software and Integrated System, Vanderbilt University, Nashville, TN, USA (e-mail: harsh.vardhan@vanderbilt.edu, janos.sztipanovits@vanderbilt.edu).

doi: 10.18178/ijmlc.2022.12.6.1104

approach is to make an outline sketch of whole training data. This method capitalizes on the fact that for sketching the data space, we only need few featured datapoints that can outline the data and can ignore all other irrelevant datapoints which has less/no relevance in creating the outline sketch. Here relevancy of a datapoint can be described as if inclusion of datapoint increase the data space coverage to a significant level then this datapoint is relevant for OOD detection. An elementary example is if two points are at same location in metric space then we can keep one as relevant datapoint and throw off other as irrelevant data because inclusion of it will not contribute towards making decision about OOD. The main contributions of this paper are the followings:

1) Developing a method which uses RRCF for OOD detection in computationally efficient manner on large data.

2) This proposed method is domain independent and can be used for any ML-based regressor model. For empirical evaluation and validation of the proposed method, we tested it on two different test cases in Cyber Physical System (CPS) domain. First, we tested this method on a reinforcement learning car braking controller with a three-dimensional input data stream and second, we tested this method on a high dimensional image data stream generated by an open-source simulator CARLA [10] on a car emergency braking system.

3) We conducted some empirical sensitivity analysis of this approach on the image dataset and found that this method can capture different level of changes in distribution.

4) In contrast to deviation-based method, which uses black box artificial neural network, The proposed method follows a white-box model which is understandable and interpretable (unlike neural networks) and can be used by the user to better understand the results. This approach has various benefits over other methods. First, it can make inference about the data being OOD/non-OOD on a stream of incoming data, which is compatible with the continuous operation of detection system. Second, empirical result reflects that it can capture presence of a single out-of-distribution data point in a stream of non-OOD data. Third, for training this model there is no such difficult hyper-parameter tuning required. Hyper-parameter tuning in machine learning relies on experimental results, and general approach to determine the optimal settings is to try many different combinations or do random/guided search in hyper-parameter space and evaluate the performance of each model. This is an iterative process and can take plenty of training time. For training this model, we have only two hyper parameters. These are: number of trees and size of tree. Finding a right hyperparameter can be decided easily in comparison to other detectors (statistical/deviation based) which need to tune their hyper-parameter on various aspects. Fourth, this approach is highly parallelizable so, by increasing allocation of resources the time complexity of the process can be improved.

II. Background

Our approach for anomaly detection relies on learning the data cluster (T)'s shape from metric space to a data-structure(S). The motivation behind learning the cluster's shape into a data-structure is to abstract the information from metric space in a structured manner such that computer and related algorithms can be efficiently deployed for inference. Let $D=\left\{d_1, d_2, \ldots, d_n\right\}$ are set of datapoints such that $d_i \in R^m$. For the purpose of Out-of-Distribution detection, following requirements are imposed on this data-structure(S):

1) S should represent the cluster of data in structured way.
2) Relationships (ψ) between the data points in metric space must be preserved in this data-structure. i.e.

$$\psi\left\{T\left(d_k, d_l\right)\right\} \approx \psi\left\{S\left(d_k, d_l\right)\right\}$$

3) Relationship(φ) of a data-point with the cluster can been coded in simple quantitative measure. i.e. $\phi\left(T, d_k\right)$ can be measured as a scalar value in the data-structure(S).

We chose the RRCF [8] as our data-structure to represent our cluster in structured way. Robust Random Cut Forest can be formally defined as:

Definition 1: Robust Random Cut Tree on set of data point $D=\left\{d_1, d_2, \ldots, d_n\right\}$ can be generated by following procedure:

1) $r_i = max_{X\in D}\left(X_i\right) - min_{X\in D}\left(X_i\right),\ \forall i \in m$

2) $p_i = \dfrac{r_i}{\sum_{i=1}^{i=m} r_i},\ \forall i$

3) Select a random dimension $\boldsymbol{i}$ with probability proportional to p_i

4) Choose $x_i \mid x_i \sim Uniform\left(max\left(X_i\right) - min\left(X_i\right)\right)$

5) $D_1 = \left\{X \mid X \in D, X_i \leq x_i\right\}$

6) $D_2 = D \setminus D_1$

Recurse on D_1 and D_2 until $D_i \geq 1$

Robust Random cut Forest is an ensemble of various RRCT. The selected relationship(ψ) between datapoints in metric space is captured as L_p distance between datapoints, then we require a distance preserving embedding of this relationship in the data structure. For this purpose, the tree distance between two datapoints d_k and d_l in data structure(S) is defined as the weight of the least common ancestor of d_k and d_l [8], then according to Johnson-Landatrauss lemma [11] the tree distance can be bounded from at least $L_1\left(d_k, d_l\right)$ to maximum $O\left(d \cdot \frac{\log|k|}{L_1\left(d_k, d_l\right)}\right)$Accordingly, a point which is far from other points in metric space will continue to be at least as far in a random cut tree. Relationship(Φ) of a data-point with the cluster can be encoded in simple quantitative measure by displacement which is an estimate of change in model complexity (summation of leave's depth) before and after inserting a given point $\boldsymbol{x}$ in tree data-structure.

III. Problem Formulation and Approach

For problem formulation, we first introduce some notations. Let X represents an input data point given to a machine learning model during training. This data can be an observation made by sensors attached to CPS system, like image data from camera or distance data from LIDAR etc. During training, we collect all such *Xs*. From ML model's perspective, this collection of all observed set of *X* represents the environment in which model has been trained, we collectively call this set as E_t. The OOD detection goal is to find whether an input data X′ given during prediction is sampled from E_t or not. If $X' \sim E_t$, then we call this observed state as non-OOD or else we call it as OOD, i.e., the trained agent has not seen this kind of input during training and the trained agent may behave unexpectedly to this OOD input data. Our approach to solve this problem is to learn the data cluster's shape from metric space to a Robust Random Cut Forest data-structure. The basic element of RRCF is a Robust Random Cut Tree (RRCT) that is a binary search tree constructed by recursively partitioning from given data points(Y) until each point is isolated. RRCF data structure contain sufficient information about the given data set(Y) and approximately preserve distances in metric space i.e., if a point is far from others that it will continue to be at least as far in a random cut tree in expectation and vice-versa (proof can be found in Guha et al [8]).

Algorithm 1: Reduced Robust Random Cut Forest (offline)

```
Input: training data (Y)
Output: reduced RRCF, threshold
Parameter: number of trees
Initialization: Randomly select Z | Z ⊂ Y
  rrcf_init = createForest(Z)
  DispVal_Z = DispValue(Z)
  DispVal_threshold = mean (DispValue_Z)
  L = Y \ Z
  for i=0; i< len(L) ; i=i+1 do
      for j=0; j < number of trees; j=i+1 do
          rrcf_new (j) = insertpoint(rrcf_init(j), L(i))
          mci(j) = Dispvalue(L(i))
      end
      point_dispValue = mean(mci)
      if point_dispValue ≥ DispVal_threshold then
          # include L(i) in featured Datapoint set
          DispVal_Z.append(point_dispValue)
          DispVal_threshold = mean(DispValue_Z)
      end
      else
          # Donot include L(i) in featured Datapoint set
          for j = 0 ; j< number of trees; j= i+1 do
              rrcf_new(j) = Deletepoint (rrcf_init(j) , L(i))
          end
      end
  end
  threshold = max(DispVal_Z )
  return rrcf_new , threshold
```

Anomaly score (also called DispValue) of a data-point measures the change in model complexity incurred by inserting a given data-point *x* in RRCF. Model complexity of a binary tree (RRCT) is defined as sum of depth of all datapoints in the tree. During insertion of a data-point that is far off the cluster in metric space, there is high probability to be partitioned in initial stage of RRCT construction. This will increase the depth of all leaves below it and consequently increase the model complexity by large number. On the other hand, if inserted point is inside the cluster, it will be partitioned in lower part of tree, and consequently it will have a smaller number of leaves below it and this will reduce the model complexity by small amount. Given a training dataset Y, we first create these random cut trees and find the maximum DispValue of all inserted data-points. We define OOD as datapoint that is significantly different from data used in training i.e., it is away from the training data cluster(Y) in the normed vector space. These off-the-cluster data-point has high probability to be isolated in initial stage of RRCT construction. Insertion of this point in tree will significantly increase the model complexity. Applying this approach on large, high dimensional training data results into creation of large forest and consequently any inference will have high prediction inaccuracy, computationally costly and even sometimes computationally infeasible. Underlying reason is as current learning models and processes are not very sample efficient, and a well-trained agent needs big training data. Research efforts are being made to make these process more sample efficient [12] but it has few practical generalities. [13] showed that on large data sub-sampling may improve random forest's performance but these forests are sensitive to extent of sub-sampling and become inconsistent with either no sub-sampling or too severe sub-sampling [14].

To address this issue, we attempted to find only those featured datapoints which will create an outline of data cluster space and has significance in making decision about OOD and drop all others datapoints for construction of RRCF. The underlying intuition is in a dense data space, we need few data points to gain information about the data space they represent, and we can drop most of the other datapoints. However, in sparser region, we select most of the datapoint to represent the data space they acquire. The sparsity and density in data space is correlated with the average model complexity increment by inclusion of datapoint in data-structure. Using this approach, we can reduce the number of training data-points significantly that needs to be stored for construction of RRCF. RRCF created using these featured data-points is called Reduced Robust Random Cut Forest (RRRCF). For reduction of entire data to the featured data, we run a process of insertion and conditional deletion on each data point in training data. After one sweep of this process on whole training data, we collect all featured datapoints which represent identical data subspace as training data space. For initialization, we first create RRCF using very small dataset(Z), where Z⊂Y, this will give us an initial small forest. Once we have created the RRCF using Z, we calculate the DispValue of all points in Z. For making decision on whether a given point can be included in the RRCF or not, we choose the mean of all DispValue calculated over Z as threshold. This threshold represents average complexity of data structure. For rest of the datapoints (Y\Z), we insert each point in forest and we calculate the DispValue for this point. If DispValue is more than the threshold DispValue of initial forest, then we will keep this point in the forest or else we will reject this point and delete it from the RRCF. We recursively apply this on all left-over datapoints(Y\Z). The final forest created from this process would be our reduced robust random cut forest and can be used for making inference during prediction for making decision about OOD for a given datapoint. This is an offline method and need to run only once. We can store the reduced RRCF(RRRCF) and threshold, which is maximum model complexity from already included featured training datapoints of selected featured datapoints to be used for prediction (refer algorithm 1). This reduced forest is a structured representation of our training data sub-space.

Algorithm 2: OOD detection (online)

Input: RRRCF(T), Data-point(x), threshold
Output: Inference about x being OOD
T' = Insertpoint (T, x)
mci = calculatemodelcomplexity
T = Delete point (T', x)
mcd = calculatemodelcomplexity
DispValue = mci -mcd
If** DispValue ≥ threshold **then
  x is OOD
else:
  x is not OOD
end

During prediction, we insert newly observed input datapoint(x) into the stored RRRCF model (obtained from algorithm 1) and check whether inclusion of this point increases the model complexity to an extent higher than threshold value. Every new observation can be passed to this detector and prediction made by the machine learning based model can be accepted only if the scalar measure of datapoint by OOD detector is lower than the threshold generated by the algorithm1. The setup for deploying this OOD detector is shown in Fig. 1.

If DispValue of new point is greater than the threshold obtained from algorithm 1 then we declare this datapoint as OOD and vice versa (refer algorithm 2). For making inference on a datapoint, we do insertion step, where we include new data point to RRRCF, and measure increased complexity and then we do deletion step to remove the inserted point. So, during prediction, we do not extend our forest, we just do one insertion and one deletion step per prediction and our reduced robust random cut forest remain intact.

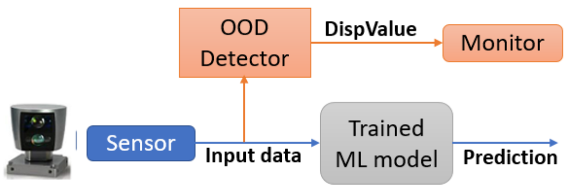

Fig. 1. Deployment of OOD detector.

IV. Experiments

To empirically evaluate above mentioned approach, we setup two experiments. In both cases, the machine learning model is a reinforcement learning based controller for car braking system where in first experiment, observations are in low dimension (3 dimensions) while in second experiment, our observation is in high dimension image data. In the first experiment, an approaching car detects a stationary obstacle at distance of 100 meters and the learning goal is to self-train a controller for braking system to stop the car without crashing (refer Fig. 2). We used reinforcement learning algorithm called DDPG (Deep Deterministic Policy Gradient) [15] based learning model to design the braking system of a car. The static and kinetic friction coefficient of road is constant throughout the experiment. The random variable in this scenario is the speed of the car when it detects the obstacle, which is drawn from a uniform distribution between 40 to 70 miles/hour. InitialSpeed(v)~U(40,70).The reward setting is done in such a way that vehicle when brakes around region of 5-10 meters from obstacle, gets the maximum reward. During training, we observe three variables, d (distance form obstacle), v (velocity of car) and mu(friction coefficient). $X=\{d,v,mu\}$ provides state information of vehicle in the environment. During training, X becomes the input of the neural network and brake value b is the output.

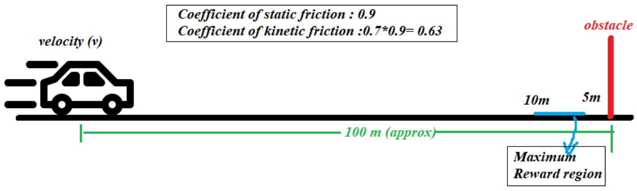

Fig. 2. Training Scenario setup.

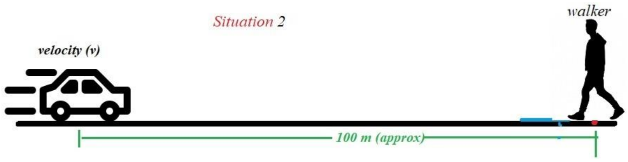

Fig. 3. Prediction Scenario 1.

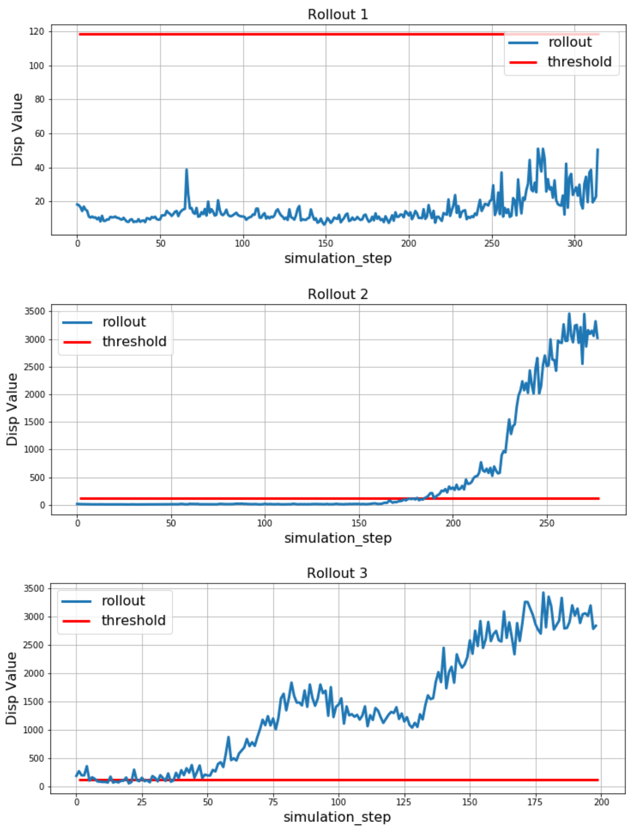

Fig. 4. Three rollouts and respective output Disp va1ue and threshold. Rollout I: Environment is same as training scenario(stationary obstacle with initial speed ~U(40,70)); Rollout2: Moving obstacle(walker) with initial speed ~U(40,70) ; Rollout3: Stationary obstacle with initial speed(v)∤U(40,70).

The braking system is trained and tested in this environmental setting. During prediction, we make some changes in the environment, which was never observed during training process. We created two such scenarios: first, in place of stationary obstacle we used an obstacle that is moving toward the car at some small velocity (for this experiment it is drawn from a uniform random distribution between 0.1-2 meter/second, refer Fig. 3, moving obstacle is represented by a walker). This situation was never observed during training, and result of it the braking system do not respond to this changed scenario appropriately and it leads to crash. In the second scenario, we keep obstacle stationary but spawned the vehicle at the velocity 75 miles/hr. It means the vehicle when detect the obstacle and invokes the braking system has velocity out of the training range (40-70 miles/hr). In this scenario also, the braking system fails to brake and leads to a crash because during training we never observed this speed. In both cases, our RRRCF based detection scheme detect these evolved situations and raised a flag for data being out of distribution. Fig. 4 shows result of above three different test rollouts. Dispvalue which is a scalar measure for each datapoint given to this RRRCF detector provides inference about possibility of data being in or out of training distribution. The threshold maximum Dispvalue derived for this set of experiment by running algorithm 1 is approximately '*118.58*', which is shown by the red horizontal line in Fig. 4. During test rollouts, at each step of simulation, we estimated DispValue for datapoint observed i.e. $X=\{d,v,mu\}$ as simulation progress. In the first roll-out(Rollout 1), we did not make any change in the environment and environment was similar to the training scenarios.

The observed Dispvalue for each datapoint during rollout is always less than the threshold value, which implies there was not any state observed during this rollout which is OOD. In case of second roll-out (Rollout2), we made the obstacle move towards the car at the speed of 2 meter/second. It was observed that in the initial stage of simulation, the Disp value is less than threshold value. It is because as obstacle is moving slowly, at initial stage very small change in observed state. Once car approaches the obstacle, the new state information X is slowly shifting from (i.e., getting far off) from training data cluster resulting into higher and higher Disp value as car reaches towards the end of rollout. It is also observed that, the change is Disp value is quite significant which facilitate easy detection of such scenario and less false alarms. In case of third roll-out (Rollout3), we keep the obstacle stationary, but we changed the initial spawning velocity of car to 75 miles/hour. We observed Dispvalue higher than threshold almost through-out the simulation step, the underlying reason initial spawned velocity of car is out of training range and when OOD detector observe this first initial state, it finds it as out-of-distribution. It can be also seen that initially measured Dispvalue fluctuates around the threshold, which reflects very proximal OODs but with the progress of simulation, this gap increases and results into higher value of measured DispValue.

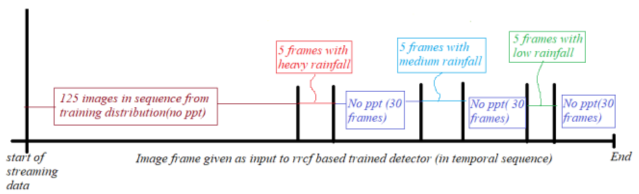

Fig. 5. Image data stream given as an input to OOD detector (stream I).

In the second experiment, we used the work done by Cai *et al*. [16] for training a braking controller, but we trained it on wider range of speed (30-75 m/s) with different reward policy. Input in this case is the image scene from the car's camera and predicted output is the brake value. We trained and tested this model on image data generated with no precipitation condition in simulator CARLA. We collected all these images with no precipitation in several episodic rollout and label it as non-OOD training scenario data. We used all these image data to build our RRRCF based OOD detector by running algorithm 1. Threshold scalar Disp Value in this experiment is calculated and it has a value of '*23.26*'. For creating OOD scenarios, we changed the no-precipitation condition to three different participation conditions, *Case1:* Heavy precipitation, *Case2*: Medium precipitation, and *Case3*: Low precipitation. (Refer fig 9) and collected image data captured by the camera.

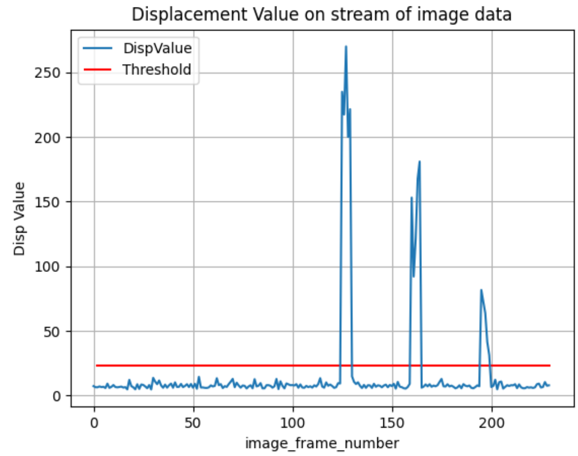

Fig.6 . Output Disp value of stream I

Our goal is to evaluate the performance of our OOD detector on stream of image data. For this purpose we created a stream of images called *stream I*, which has first 125 image frames with no precipitation scenario, then 5 frames of each three different precipitation cases (heavy, medium and low) with 30 intermittent frames of no precipitation in-between as shown in Fig. 5.

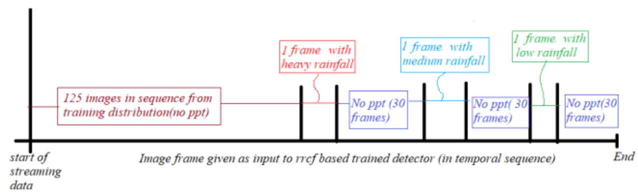

Fig. 7. Image data stream given as input to OOD detector (stream II).

With a given input image, if DispValue is greater than the threshold value then we declare it as OOD. Fig. 6 shows the DispValue generated by the OOD detector for the data stream I. The red horizonal line represent the threshold value of detector. It can be easily observed that the detector output is significantly higher than threshold value in all the three OOD case (heavy, medium and low precipitation) and lower for all non-OOD case. We tried to find the performance of detector in context of point OOD i.e., is the detector able to detect even single OOD in stream of non-OOD data?

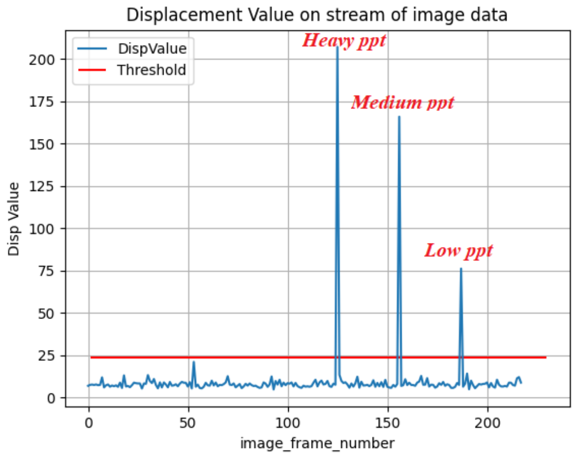

Fig. 8. Output Disp value of stream II.

To find an answer to this question, we created another stream of image frames called, *stream II*. In *stream II*, we inserted single frame of different precipitation cases (heavy, medium, and low) in a stream of image frame similar to training data i.e., no precipitation (refer 7). We observed that OOD detector can even detect a single OOD data in a stream of no-OOD data (for result, refer Fig. 8). Source code of experiments can be found here: https://github.com/vardhah/finalcodes/tree/master/ODD.

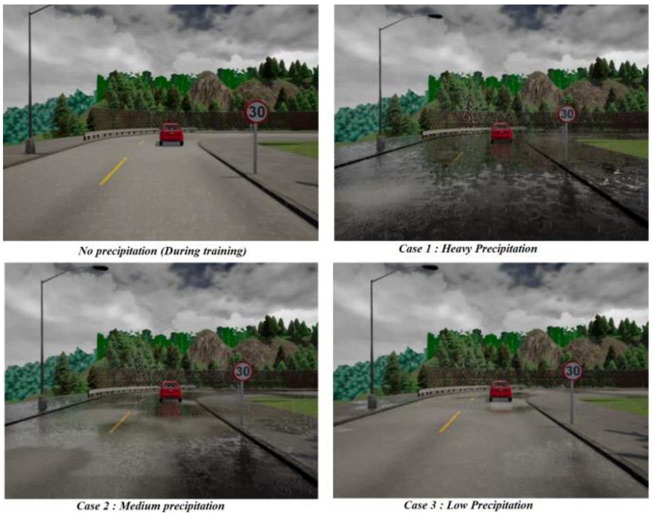

Fig. 9. Different precipitation conditions generated during the training and prediction (no ppt, heavy ppt, medium ppt, low ppt).

V. Discussion

A. Time Complexity Analysis

Algorithmic complexity of ODD detection process should be low during prediction time. Its time complexity depends upon the process required to make inference on a data-point by detector i.e.," Forest maintenance on stream". Forest Maintenance on stream of data involves two processes per data-point first, insertion of a point in tree data-structure and second, deletion of a point from tree data-structure. If we store each data-point i.e. the leaf of a tree in a hash table, then search process for locating a leaf will have time complexity O(1). Given a set of points Y and a point $x \in Y$, we construct a tree T on data Y. Consider if we want to delete a leaf x from the tree and produces tree $T(Y-x)$, we just need to remove the parent of x and make another child's parent pointer to its grandparent accordingly (refer fig 10). This deletion process has time complexity of $O(1)$. Total time complexity of deletion process of a point (search + deletion) from a tree is $O(1)$. We can efficiently insert and delete data-points into a random cut tree data-structure. Insertion of a point in tree is a process when we have a tree T and we want to insert a leaf x to the tree and produces tree $T(Y \cup x)$, where $x \notin Y$. In the worst case, the time complexity of this process would be $O(n)$, while in the average and best case it would be O(logn) and $O(1)$ respectively, where n is the size of tree i.e. number of data-points used to create the tree. If we have m trees in a forest, then total time complexity of the process is O(mn), $O(m\text{logn})$ and $O(m)$ in worst, average, and best case respectively. The effect of m on RRRCF creation, insertion and deletion process can be made $O(1)$ for time complexity analysis purpose as the process is completely parallelizable i.e. we can do insertion and deletion on each tree on different thread/core in parallel if GPU/multi-core based parallel implementation of RRRCF is used.

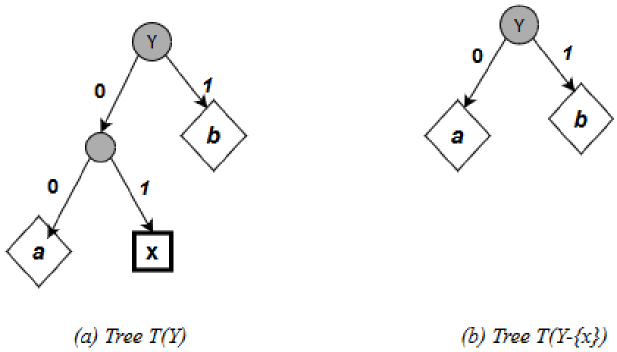

Fig. 10. Deletion of a datapoint from a tree.

Consequently, the insertion process on forest will have time complexity $O(n)$, $O(\log n)$ and $O(1)$ respectively for worst, average and best case scenario. Here a reasonable question is how big the variable n can be? As machine learning process is not very sample efficient, a well-trained model results into large training data. For example, training a braking system perception LEC for speed range variance of 10Km/hr (90km/h to 100km/h) in Cai *et al* [16] experimental setup took approximately 8160 images in a 100-meter distance. Similarly in experiment set 1, our training data-point was approx 0.36 million. But for ODD detection, it is not required to use whole training data and it is enough if we can make a sketch/outline of data i.e. we can ignore many datapoints that are densely clustered and use only those featured data points which represent the outline of data subspace.

TABLE I: Total Datapoints and Selected Featured Datapoints for RRRCF and Percentage Reduction

Scenario	Training points	Featured points	%age reduction
1	367035	11134	96.97
2	8197	1050	87.2

We used our offline algorithm to select the number of data points which may represent the whole data. For first set of experiment, our algorithm gave 11134 featured data-points out of total 367035 datapoints. For second set of experiment, our algorithm gave 1050 featured data-points out of total 8197 image datapoints. Selection on n also depends upon the resources available. With availability of large number of parallel cores in modern GPU, it is also possible to increase m and reduce n and randomly choose a subset of element out of *n* for creating a tree. With a very large training data, it is possible to extract only a small segment of data to sketch the RRRCF outline.

B. Sensitivity of OOD Detector on Change in Distribution

For testing the sensitivity of the OOD detector, in experiment set 2, we used image frames generated from different weather conditions (different level of precipitation: heavy, medium, and low. refer Fig. 9). Higher the precipitation level, the more deviation of data from training distribution. Empirical result suggests that this RRRCF based detector can detect all three different level of change in distribution. Empirical result also suggests that as test data-point go away from the training data cluster in metric space, the anomaly score i.e., DispValue progressively increases in expectation.

C. Sensitivity of OOD Detector on Single OOD Data in Stream of non-OOD Data

OOD detection problem in machine learning can generally classified in two parts- contextual OOD and point OOD. In case of contextual OOD, goal is to detect the anomaly in observed input data in environmental context. For example, [16] proposed a method for OOD detection for CPS system but this method needs several OOD data before making conclusion about the environment. These methods require multiple OODs in sequence before they conclude anything about data and can completely miss single OODs or intermittent OODs and it is not desirable for safety critical nature of CPS. With rrrcf based approach, we tried to fill this gap, and empirical result reflects that it can detect even a single OOD data in the stream of non-OOD data.

VI. Training Details

A. RRRCF Training Details

In case of experiment set 1, we collected approximately 367035 training datapoints(Y), where each data point is a tuple of distance, velocity and friction coefficient ({*d*, *v*,*mu*}). For constructing initial RRCF, we selected 1000 random data from pool of total data i.e., *Z*=1000. After construction of initial rrcf data structure, mean DispValue is calculated over all points in *Z*. It is the initial threshold value, and its value was approximately '*6.4'*. This threshold was used for making decision about inclusion/discard of other points in training data (*Y\Z*). After every 5000 data-points, we calculated the mean DispValue as our new threshold. The reason for repeatedly updating our threshold is to accommodate the changes made in tree structure by inclusion of datapoints. We recursively applied this process to rest of training data. After running algorithm 1, 10134 featured datapoints were selected and rest 355574 datapoints were rejected. These total 11134 (10134+1000) datapoints were used to represent our whole data (367035) in the reduced RRCF data structure, which is approximately 3.03%percent of whole data. In case of experiment set 2, same experimental setting as by Cai *et al* [16] was used and 8197 images were collected for training scenario i.e., with no precipitation condition. For creation of initial RRCF, we selected 100 images i.e., *Z*=100. Once initial RRCF was created, the average threshold for Z was calculated and it was approximately *'5.62'*. This threshold was used for making decision about inclusion/discard of other points in training data(*Y\Z*). After every 100 data-points, we recalculated the mean DispValue as our new threshold. After running algorithm 1,950 datapoints were selected and rest 7147 datapoints were rejected. These featured 1050 datapoints (950+100) were used to represent our whole data (8197 images) in the reduced RRCF data structure, which is approximately *12.8%* of whole data. For construction of random cut tree, modified version of implementation of tree written by Barto *et al* [17] was used.

B. Braking system Training Details:

The braking system in experiment set 1 is trained using Actor-Critic based Deep Deterministic Policy Gradient algorithm with both actor and critic network are three-layer neural network, with 50(layer1) and 30(layer2) neurons in the hidden layer and 1 neuron in output layer. For hidden layers in both cases, *relu* activation function was used and for output layer *sigmoid* activation layer was used in actor and *linear* activation was used in critic. The model was trained using Adam [18] optimization method by tuning different learning rate for 5000 episodes. In case of experiment set 2, we used same setting as done by Cai *et al.* [16].

VII. Related Work

It is a well-known issue that machine learning models fails on both in-distribution training data [19] as well out-of-distribution training data, and they often do this even after providing high confidence predictions on training data [20]. Out of distribution detection can be seen as attempt towards making a verified and safe artificial intelligence. As direct verification suffers from a scalability problem due to computational complexity and size of networks, till now work has been done for smaller scale networks or with approximate methods that provide some convergence guarantees on the bounds [21]. The alternative is to detect datapoints those are OOD and do not use predicted output from the trained model on these kind of input data. [22] attempted to detect OOD based on the *softmax* probability estimated by the predictor. Empirically they showed that OOD data generally have lower *softmax* probabilities than the correctly classified datapoints, as this can be used to detect the data being OOD. [6], [16], [23] used the deviation-based approach using one or another flavor of Variational Auto Encoder (VAE). VAEs are directed probabilistic graphical model, whose posterior and likelihood are approximated by a neural network, forming an encoder and decoder like structure. In VAE based OOD detector, people capitalize on either latent space representation or its error in reconstruction to find the OOD data from normal data. In proximity-based method, early work on OOD detection is done by using collection of random cut decision tree and called it isolation forest [7]. Improving this work [8] proposed Robust Random cut forest, which addressed most of the challenge posed during practical use of isolation forest like working on stream of data, false detection etc.

VIII. Conclusion and Future Work

In this work, we demonstrated a white-box interpretable method for out-of-distribution detection. We also showed that this method can detect even a single OOD data in a stream of non-OOD data. We demonstrated the effectiveness of this approach for both low and high dimensional input data space. We also discussed that a GPU based parallel implementation of reduced RRCF can significantly reduce the execution time. Parallel implementation of reduced rrcf may be used in real-time system for real-time OOD detection. Writing an opensource GPU and multi-core implementation of reduced RRCF and its evaluation for real time performance is the part of future work. Evaluation on real world data and comparison with other methods for OOD like variational autoencoder (VAE) based, Gaussian mixture models based are also part of our future work.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Janos Sztipanovits and Harsh Vardhan outlined the problem and discussed different approaches to solve the problem and outlined this approach in its current form. Harsh Vardhan conducted the experiment, collected, and analyzed the data. Both authors contributed to writing the paper. All authors had approved the final version.

References

[1] H. Vardhan, P. Volgyesi, and J. Sztipanovits, "Machine learning assisted propeller design," in *Proc. the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, 2021, pp. 227–228.

[2] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning, " *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[3] A. Al Bataineh, "A comparative analysis of nonlinear machine learning algorithms for breast cancer detection," *International Journal of Machine Learning and Computing*, vol. 9, no. 3, pp. 248–254, 2019.

[4] M. A. Ghazanfar, S. A. Alahmari, Y. F. Aldhafiri, A. Mustaqeem, M. Maqsood, and M. A. Azam, "Using machine learning classifiers to predict stock exchange index," *International Journal of Machine Learning and Computing*, vol. 7, no. 2, pp. 24–29, 2017.

[5] D. M. Hawkins, *Identification of Outliers*, Springer, 1980, vol. 11.

[6] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[7] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[8] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, "Robust random cut forest based anomaly detection on streams," in *Proc. International Conference on Machine Learning*, 2016, pp. 2712–2721.

[9] R. A. A. Habeeb, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A survey," *International Journal of Information Management*, vol. 45, pp. 289–307, 2019.

[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," arXiv preprint arXiv: 1711.03938, 2017.

[11] W. J. J. Lindenstrauss, "Extensions of Lipschitz maps into a Hilbert space," *Contemp. Math*, vol. 26, pp. 189–206, 1984.

[12] K. Asadi and J. D. Williams, "Sample-efficient deep reinforcement learning for dialog control," arXiv preprint arXiv: 1612.06000, 2016.

[13] R. Genuer, J.-M. Poggi, C. Tuleau-Malot, and N. Villa-Vialaneix, "Random forests for big data," *Big Data Research*, vol. 9, pp. 28–46, 2017.

[14] C. Tang, D. Garreau, and U. von Luxburg, "When do random forests fail?" *Advances in Neural Information Processing Systems*, 2018, pp. 2983–2993.

[15] 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," arXiv preprint arXiv: 1509.02971, 2015.

[16] F. Cai and X. Koutsoukos, "Real-time out-of-distribution detection in learning-enabled cyber-physical systems," in *Proc. 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, 2020, pp. 174–183.

[17] M. D. Bartos, A. Mullapudi, and S. C. Troutman, "rrcf: Implementation of the robust random cut forest algorithm for anomaly detection on streams," *Journal of Open-Source Software*, vol. 4, no. 35, p. 1336, 2019.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv: 1412.6980, 2014.

[19] H. Vardhan and J. Sztipanovits, "Rare event failure test case generation in learning-enabled-controllers," in *Proc. 2021 6th International Conference on Machine Learning Technologies*, New York, NY, USA, 2021, p. 34–40.

[20] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Man´e, "Concrete problems in AI safety," arXiv preprintarXiv: 1606.06565, 2016.

[21] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[22] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," arXiv preprintarXiv:1610.02136, 2016.

[23] K. Sundar, S. Ramakrishna, Z. Rahiminasab, A. Easwaran, and A. Dubey, "Out-of-distribution detection in multi-label datasets using latent space of beta-vae," arXiv preprint arXiv: 2003.08740, 2020.

Harsh Vardhan received his bachelor's in technology in electrical and electronics engineering from SASTRA University in India from 2005 to 2009). He is currently pursuing his Ph D. from 2018 in computer science at Vanderbilt University, Nashville, TN, USA. His research interest is data-driven modeling, data-driven system identification, machine learning based system design and operation. He worked from 2010- 2017 with Government of India as a project executive in Electrical and Electronics Engineering Department. He has won multiple national competition in field of autonomous robotics and integrated circuit designs.

Janos Sztipanovits graduated from the Technical University of Budapest, Hungary, in 1970 and received the Ph.D. degree from the Hungarian Academy of Sciences, Budapest, Hungary, in 1980. He is currently the E. Bronson Ingram Distinguished Professor of Engineering at Vanderbilt University, Nashville, TN, USA, and founding director of the Institute for Software Integrated Systems. His current research interest includes the foundation and applications of model and component-based design of cyber-physical systems, design automation methods and systems-security co-design technology. Dr. Sztipanovits is an external member of the Hungarian Academy of Sciences and holds the title of John von Neumann Professor of the Technical University of Budapest. He was founding chair of the ACM Special Interest Group on Embedded Software (SIGBED).