# DETECTING ATTACKS AGAINST DEEP REINFORCEMENT LEARNING FOR AUTONOMOUS DRIVING

MAURA PINTOR$^1$, LUCA DEMETRIO$^2$, ANGELO SOTGIU$^1$, HSIAO-YING LIN$^3$, CHENGFANG FANG$^4$, AMBRA DEMONTIS$^1$, BATTISTA BIGGIO$^1$

$^1$University of Cagliari, Italy $^2$University of Genoa, Italy $^3$Huawei Technologies France $^4$Huawei International, Singapore
E-MAIL: {maura.pintor, angelo.sotgiu, ambra.demontis, battista.biggio}@unica.it, luca.demetrio@unige.it, lin.hsiao.ying@huawei.com, fang.chengfang@huawei.com

**Abstract:**

**With the advent of deep reinforcement learning, we witness the spread of novel autonomous driving agents that learn how to drive safely among humans. However, skilled attackers might steer the decision-making process of these agents through minimal perturbations applied to the readings of their hardware sensors. These force the behavior of the victim agent to change unexpectedly, increasing the likelihood of crashes by inhibiting its braking capability or coercing it into constantly changing lanes. To counter these phenomena, we propose a detector that can be mounted on autonomous driving cars to spot the presence of ongoing attacks. The detector first profiles the agent's behavior without attacks by looking at the representation learned during training. Once deployed, the detector discards all the decisions that deviate from the regular driving pattern. We empirically highlight the detection capabilities of our work by testing it against unseen attacks deployed with increasing strength.**

**Keywords:**

**reinforcement learning, autonomous driving**

## 1 Introduction

Deep Reinforcement Learning (DRL) is used to train deep-learning models to complete tasks by trial-and-error experience. In the context of self-driving cars, for example, a DRL system can be used to drive without direct human supervision. Unfortunately, DRL is subject to adversarial perturbations, i.e. carefully-crafted inputs that cause the system to behave unexpectedly [1, 2]. For example, previous works [1, 3] compute the Fast Sign Gradient Method (FGSM) against reinforcement learning policies trained to play ATARI games or to simulate simple physics environments. However, these strategies are studied in very simple cases and stand more as a proof of concept that needs to be adapted to the domain of choice. Detectors have been proposed to counter attacks against classification and showed promising results. They are usually designed as separate predictors trained to detect the perturbed samples by learning from a pre-computed dataset [4,5]. These detectors however have been shown to be less robust to unseen adversarial perturbations [6]. Furthermore, only a few approaches apply detectors to counter adversarial input attacks against DRL. Lin et al. use a visual foresight model to check if the agents' actions reflect the predictions [7]. Van Wyk et al. [8] leverage state transitions to detect the presence of hardware faults or non-adversarial cyber-attacks against the sensors. Similarly, Mustafa et al. [9] develop a reinforcement learning algorithm that is tuned to detect anomalies of wireless sensors and adversarial attacks, by again leveraging state-transition matrices. These approaches, however, require a fully-observable environment in which the states are perfectly known by the agents. In this paper, we propose an anomaly detector that promptly signals adversarial attacks against DRL policies. We first describe the DRL components (Sect. 2), then we detail the attack scenario (Sect. 3), and we apply anomaly detection algorithms to prevent DRL systems to cause damage when affected by adversarial attacks (Sect. 4). We show the effectiveness of our method by applying it to autonomous-driving simulation (Sect. 5), and finally, we comment on and summarize the results (Sect. 6).

## 2 DRL for Autonomous Driving

The goal of Reinforcement Learning (RL) is to train an agent to complete a task within an environment $\mathcal{E}$, within multiple time steps $t$. At each time step $t$, the agent receives representations of the state of the environment called *observations* $\mathbf{x}$. The *agent* can pick an action $y \in A$ to progress in the objective task. If the actions are taken from a finite set, i.e., $\mathcal{A} = \{y_1, \dots, y_A\}$, the action space is called a *discrete action space*. The agent learns a behavior function called *policy* that, depending on the observations and the state, picks an action $y$. After the action, the environment sends the agent an *immediate reward* $r_t$. This is a scalar that indicates how well the agent is doing at the current step. One episode is a sequence of time steps for which states, actions, and rewards are defined. Episodes end with a terminal state or when the maximum number of steps is reached. The agent's goal is to learn the optimal policy $\pi^\star$, which makes it pick the sequence of actions that maximize the expected *cumulative reward* received over episodes.

Proximal Policy Optimization (PPO) is an actor-critic, policy-gradient method that learns directly from what the agent encounters in the environment. More in detail, it learns by using two networks, the actor and the critic network. The actor directly implements the policy, whereas the critic estimates how good the policy is and feeds the information back to the algorithm for improving the actor network. In the case of the PPO, the *policy* $\pi$ can be parametrized through the weights $\theta = (\mathbf{w}_1, \dots, \mathbf{w}_l)$ of a deep neural network. The actor network directly parametrizes the policy through a score function:

$$\pi_\theta(y|\mathbf{x}) = f(y, \mathbf{x}; \theta), \tag{1}$$

that gives each action in $\mathcal{A}$ a score proportional to the expected reward of the subsequent states. Hence, the goal of a PPO algorithm is to find $\theta^\star$ that maximizes the expected cumulative reward. In the following, we will refer to the internal representations at each layer $i$ of the network as $\mathbf{z}_i = \sigma(\mathbf{w}^\star_{i-1}\mathbf{z}_{i-1})$, where $\sigma$ is the activation function of the layer.

## 3 Sensor Attacks against Deep Reinforcement Learning Agents

We consider attacks where the agent receives altered observations $\mathbf{x}' = \mathbf{x} + \delta$, where $\delta$ is an additive perturbation applied to the observations. In our settings, we assume that $y'$ is the target action, i.e., the action that the agent should perform to achieve the attacker's goal. Then, we formalize the attack as the following problem:

$$\delta^\star \in \arg\min_\delta L(\mathbf{x} + \delta, y'; \theta) \tag{2}$$

$$\text{s.t.} \quad ||\delta||_p \leq \epsilon \text{ and } \mathbf{x} + \delta \in [\mathbf{0}, \mathbf{1}]. \tag{3}$$

Where the attacker is interested in optimizing a loss function $L$ (e.g., the cross-entropy loss) to force the model to pick the selected action $y'$. By solving this problem, the attacker can find the optimal perturbation $\delta^\star$ that, added to the input observations $\mathbf{x}$, will force the agent to take the desired action $y'$ at each time step $t$. To launch more sophisticated attacks, the attacker can perform the attack multiple times to cause a sequence of actions that bring the agent into an unwanted state. For example, in the case of autonomous driving, the attacker can cause the self-driving agent to cause an accident.

The Projected Gradient Descent attack (PGD) [10] was proposed to find a solution to Equation 2 via iterative steps in the direction of the gradient $\nabla_\mathbf{x}$. PGD computes the gradient $\nabla_\mathbf{x} L(\mathbf{x}, y; \theta)$ and iteratively sums it to the current perturbation $\delta$ to improve the objective loss.

## 4 Defending Against Perturbed Observations

Anomaly detection algorithms can find out when our agent is being attacked by altered observations. To achieve this, we want to define a detector

$$D(\mathbf{x}; \theta) = g(\mathbf{z}_1, \dots, \mathbf{z}_l) \tag{4}$$

that will output a score proportional to how much the model considers each sample of the test data to be anomalous. Analysts can either analyze the top few anomalies, or they can select a cutoff threshold $\rho$ to select the number of anomalies to flag as ground truth. Specifically, we are interested in detecting unseen attacks. Thus, we leverage *semi-supervised novelty detection* (SSND).

In SSND, the training data is not polluted by outliers, and we are interested in detecting whether a *new* observation is an outlier. In this context, an outlier is also called a *novelty*. When a new sample falls in low-density regions of the training data distribution, they are considered anomalies (or novelties). Anomaly detection algorithms can be grouped into:

- *Statistical methods (MCD [11], HBOS [12]).* These methods use high-breakdown estimators with good performance under the null hypothesis, stating that no outliers are present and labeling as anomalies the samples that fail the statistical tests.
- *Density-based techniques (kNN [13], Isolation Forests [14], LOF [15]), CBLOF [16].* Estimate the data distribution locally by counting how many data objects are present in the sample in some local volume;
- *Geometric-based techniques (ABOD [17], OCSVM [18], PCA [19]).* Use a geometric model and tricks to define a frontier between the normal data and the outliers;
- *Ensemble techniques (Feature Bagging [20], AverageKNN [21]).* Group two or more learners and combine their scores or their decisions.

Anomaly detectors are usually evaluated in terms of their correct detections through the True Positive Rate (TPRs), and false alarms through the False Positive Rate (FPRs). When evaluating detectors, the desired characteristic is to have high TPR, meaning that they can correctly detect anomalies, and low FPR, hence avoiding false alarms. By observing the TPR and FPR over different values of the detection threshold, it is possible to build the Receiver Operating Characteristic (ROC) curves, which describe how the classifier can distinguish between the two classes. In the absence of an analyst able to inspect the anomalies, it is common practice to set the threshold $\rho$ to a value that fixes a specific FPR as the number of false positives that are supposed to contaminate the dataset (the contamination factor $c$ ). As previously discussed, we use the concatenation of all the activations $\{\mathbf{z}_1, \dots \mathbf{z}_l\}$ of the deep neural network as features for our anomaly detectors. Note that the detector uses representations already computed for predicting the next action. Thus the computational cost of the detector will only be the prediction from the anomaly detector model. To use the detector, we combine the output layer of the actor network with an additional score given by the detector. Hence, we add one more action in the action space, corresponding to the "failsafe" action, i.e., $\mathcal{A}^\star = \{a_1, \dots, a_A, a_D\}$. Compared to the DRL system without the detector, our model will output the action corresponding to the maximum score in $\mathcal{A}^\star$, thus selecting the failsafe action if the detection score is greater than a specific detection threshold $\rho$.

## 5 Experiments

We run all our experiments on a 32-cores Intel® Xeon® Gold 5217 with 188 GB of RAM, with two Nvidia RTX A6000 with 48 GB of VRAM. For testing our approach, we use the Python-based `Hiwhay-Env` simulator [22], which gathers a collection of environments for decision-making in autonomous driving.

**Agent.** We train an agent that uses Kinematic Observations, i.e. a $V \times F$ array that describes a list of $V$ nearby vehicles by a set of features of size $F$. We select as features the two coordinates describing the position of each vehicle (offset on the horizontal axis, offset on the vertical axis) and the two vectors that describe their speed in the two directions (velocity on the horizontal and vertical axis). We use a discrete action space that contains the actions {"lane left", "idle", "lane right", "faster", "slower" }. We train a PPO policy network composed of two layers of 256 weights. We train the model for $1,000,000$ episodes, with a maximum number of steps per episode of 40.

**Attacks.** We design three attacks that force the choice of specific target actions. We divide them into denial-of-service attacks (DOS) and dangerous-driving attacks (DD). DOS attacks are meant to make the agent useless for the task. DD attacks are attacks designed to cause crashes or dangerous situations.

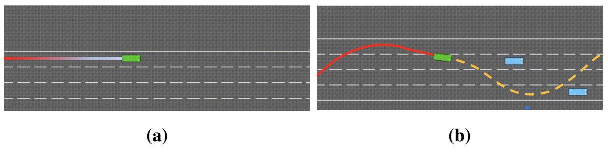
**FIGURE 1.** The Brake and Lane-changing attacks. The red line stands for steady and continuous velocity. In blue, we represent the deceleration, while the yellow dashed line is the future trajectory.

*Brake (DOS).* This attack causes the agent to stop in the middle of the lane by forcing it to select always the action "slower" (Figure 1a). This attack increases the likelihood of causing accidents as the other vehicles might impact it.

*Speed (DD).* This attack forces the driving agent to always pick the action "faster", coercing it to drive at sustained speed. This is the most dangerous scenario, as the agent will likely impact other vehicles on the road.

*Lane-changing (DD).* This attack forces the agent to pick alternately the actions "lane left" and "lane right" (Figure 1b). This scenario is unsafe and can also lead to causing the other vehicles to crash to avoid the agent.

We implemented these attacks by using the targeted implementation of PGD available in the `Foolbox` library [23]. We use the $\ell_2$ version of PGD, for which we set the number of iterations for the attack to $n = 20$, the step size to $\alpha = 0.1$, and we use different perturbation sizes $\epsilon \in \{0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. For each attack and perturbation size, we measure the crash rate $CR$ as the average number of times the agent crashes over the total number of 1000 episodes.

**Detector.** For implementing our detector, we collect a dataset of 1500 episodes for the normal driving scenario and 1000 episodes for each driving scenario. We collect the inner states of the DNN, i.e., $\mathbf{z}_i$ for $i \in \{1, 2\}$, and we use these as a feature vector for our anomaly detection.

To avoid high correlation due to taking samples from the same episodes, we select only one random step from each episode and use the data collected in that step for training and evaluating our detector. This is consistent with how we plan to use our detectors, as they will detect the attack from a single step of the RL episode (hence it can evaluate the detector at each step and block potential malicious actions). After this data collection step, we preprocess the data with a Standard Scaler, which rescales the data to fit in a Normal Distribution, i.e. a Gaussian with zero mean and unit variance.

We take the implemented anomaly detectors available

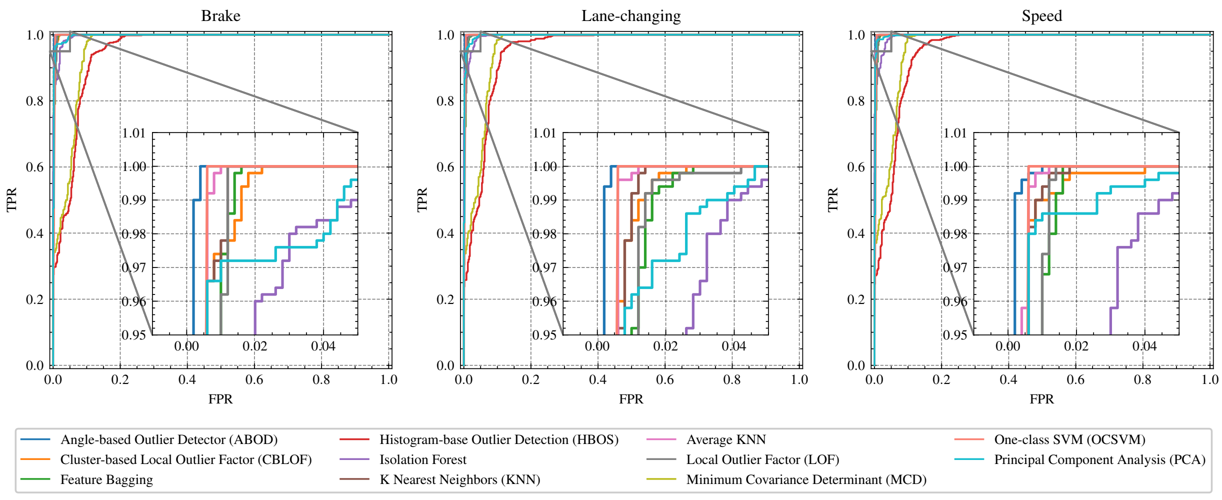

**FIGURE 2.** ROC curves of our detectors. The closer the curve is to the point TPR=1.0, FPR=0.0, the better the detector is able to distinguish between anomalies and normal situations. The zoomed area focuses on the difference between the different models.

in the `PyOD` library [24]. We use the following detectors, with the default hyperparameters: Angle-based Outlier Detector (ABOD) [17], Cluster-based Local Outlier Factor (CBLOF) [16], Feature Bagging [20], Histogram-base Outlier Detection (HBOS) [12], Isolation Forest [14], K Nearest Neighbors (KNN) [13], Average KNN [21], Local Outlier Factor (LOF) [15], Minimum Covariance Determinant (MCD) [11], One-class SVM (OCSVM) [18], and Principal Component Analysis (PCA) [19].

To simulate a real-case scenario where the attacks are not known at the training phase, we train our anomaly detectors with 500 samples belonging to the episodes of the normal class (the datasets containing anomalies are used only at the inference phase). Then, we test our detectors with the remaining 1000 samples from the normal scenario and the 1000 samples taken from the attacks we implemented.

We show the ROC curves of the detectors in Figure 2. As specified before, the best classifiers are the ones for which the curve is close to having TPR = 1.0 and FPR = 0.0. As displayed in the figure, the majority of the anomaly detectors that we trained are able to achieve good performances on the tasks. It is worth noting here that the detectors are the same for the three attacks, as they are only trained with normal data. For this reason, the choice should prioritize detectors that achieve good results in all the analyzed scenarios.

To give an overview of the performances of our detectors, we report the Area Under the Curve (AUC) values for the computed ROCs in Table 1. Again, the higher the value of the AUC, the closer the curve is to the perfect classifier, and we are interested in models able to achieve good results in all three cases. We observe that most models detect attacks in the three scenarios very well, demonstrating the usefulness and descriptive quality of our features. Some classifiers, mostly the statistical methods, are slightly less efficient overall, suggesting that the distributions might be skewed and require further preprocessing to identify outliers better. We emphasize, however, that even without applying additional preprocessing these features are still sufficient to identify and prevent the threats efficiently.

**TABLE 1.** Performances of the outlier detectors on the tested environments, for $||\delta||_2 = 3.0$.

| **Detector** | Brake | Lane-changing | Speed |
|---|---|---|---|
| Angle-based Outlier Detector (ABOD) [17] | **1.00** | **1.00** | **1.00** |
| Cluster-based Local Outlier Factor (CBLOF) [16] | **1.00** | **1.00** | **1.00** |
| Feature Bagging [20] | **1.00** | **1.00** | **1.00** |
| Histogram-base Outlier Detection (HBOS) [12] | 0.90 | 0.90 | 0.91 |
| Isolation Forest [14] | 0.99 | 0.99 | 0.99 |
| K Nearest Neighbors (KNN) [13] | **1.00** | **1.00** | **1.00** |
| Average KNN [21] | **1.00** | **1.00** | **1.00** |
| Local Outlier Factor (LOF) [15] | **1.00** | **1.00** | **1.00** |
| Minimum Covariance Determinant (MCD) [11] | 0.94 | 0.94 | 0.94 |
| One-class SVM (OCSVM) [18] | **1.00** | **1.00** | **1.00** |
| Principal Component Analysis (PCA) [19] | **1.00** | **1.00** | **1.00** |

To demonstrate the usefulness of the detectors, we show in Figure 3 the crash rate (CR) before and after applying the detector. For drawing these curves, we select one of the best-performing models, the OCSVM, and we fix the FPR threshold

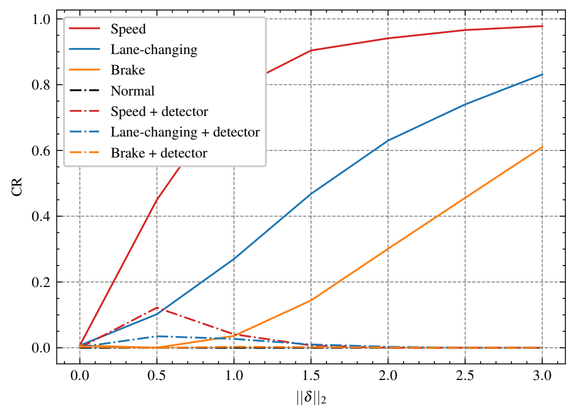

**FIGURE 3.** Crash rates of our attacks (continuous lines —), and crash rates with our OCSVM detector (dashed lines — · —). From the plot, it is evident how a detector can prevent the crashes caused by the attacks on the sensors.

to have 10 % FPs. When the detector triggers, we block the execution of the scenario, and we recompute $CR$ after detection. This simulates a scenario in which the agent stops in the emergency lane (failsafe action) as soon as the attack is detected.[1]

## 6 Conclusions

In this paper, we analyze attacks against hardware sensors of autonomous-driving agents, and we propose a detector that promptly stops the agent before causing damage. Our results highlight that the detector avoids crashes against other vehicles by looking at one single attacker-tainted observation.

We point out, however, that our method has been tested only on sensor attacks, whereas it could also prove useful in more complex attack scenarios, such as adversarial policies [25]. We believe our method can be further improved, by considering other contextual information as input features, like the sequence of actions picked by the agent. Moreover, the detector could consider ensembles of multiple models to promote diversity and improve its detection capability, and better handle unseen-and-complex attack scenarios.

We believe that our work can serve as a starting point for future research in DRL and trustworthy autonomous driving.

## Acknowledgment

This work has been supported by Huawei Technologies Co. Ltd. under the project “Deep Reinforcement Learning Key Security Technologies” (Agreement No: TC20201118006); by project SERICS (PE00000014), under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU; by Fondazione di Sardegna under the project “TrustML: Towards Machine Learning that Humans Can Trust”, CUP: F73C22001320007; and by the PRIN 2017 project RexLearn (grant no. 2017TWNMH2), funded by the Italian Ministry of Education, University and Research.

[1]Note that this differs from the scenario in the Braking attack, where the agent stops in the driving lane.

## References

[1] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, 2017.

[2] Ambra Demontis, Maura Pintor, Luca Demetrio, Kathrin Grosse, Hsiao-Ying Lin, Chengfang Fang, Battista Biggio, and Fabio Roli. A survey on reinforcement learning security with application to autonomous driving. *arXiv preprint arXiv:2212.06123*, 2022.

[3] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939, 2017.

[4] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[5] Angelo Sotgiu, Ambra Demontis, Marco Melis, Battista Biggio, Giorgio Fumera, Xiaoyi Feng, and Fabio Roli. Deep neural rejection against adversarial examples. *EURASIP Journal on Information Security*, 2020:1–10, 2020.

[6] Nicholas Carlini and David A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In Bhavani Thuraisingham, Battista Biggio, David Mandell Freeman, Brad Miller, and Arunesh Sinha, editors, *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, pages 3–14. ACM, 2017.

[7] Yen-Chen Lin, Ming-Yu Liu, Min Sun, and Jia-Bin Huang. Detecting adversarial attacks on neural network policies with visual foresight. *arXiv preprint arXiv:1710.00814*, 2017.

[8] Franco Van Wyk, Yiyang Wang, Anahita Khojandi, and Neda Masoud. Real-time sensor anomaly detection and identification in automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):1264–1276, 2019.

[9] Iqra Mustafa, Sheraz Aslam, Muhammad Bilal Qureshi, Nouman Ashraf, Shahzad Aslam, Syed Muhammad Mohsin, and Hasnain Mustafa. Rl-madp: Reinforcement learning-based misdirection attack prevention technique for wsn. In *2020 international wireless communications and mobile computing (IWCMC)*, pages 721–726. IEEE, 2020.

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[11] Johanna Hardin and David M Rocke. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics & Data Analysis*, 44(4):625–638, 2004.

[12] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track*, 1:59–63, 2012.

[13] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438, 2000.

[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[15] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[16] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Outlier detection in axis-parallel subspaces of high dimensional data. In *Advances in Knowledge Discovery and Data Mining: 13th Pacific-Asia Conference, PAKDD 2009 Bangkok, Thailand, April 27-30, 2009 Proceedings 13*, pages 831–838. Springer, 2009.

[17] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452, 2008.

[18] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[19] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering, 2003.

[20] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166, 2005.

[21] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *Principles of Data Mining and Knowledge Discovery: 6th European Conference, PKDD 2002 Helsinki, Finland, August 19–23, 2002 Proceedings 6*, pages 15–27. Springer, 2002.

[22] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[23] Jonas Rauber, Roland Zimmermann, Matthias Bethge, and Wieland Brendel. Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax. *Journal of Open Source Software*, 5(53):2607, 2020.

[24] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[25] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *International Conference on Learning Representations*.