Received October 1, 2021, accepted October 28, 2021, date of publication November 8, 2021, date of current version November 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3125920

Detect Adversarial Attacks Against Deep Neural Networks With GPU Monitoring

TOMMASO ZOPPI AND ANDREA CECCARELLI
Dipartimento di Matematica e Informatica, Università degli Studi di Firenze, 50134 Firenze, Italy
Corresponding author: Andrea Ceccarelli (andrea.ceccarelli@unifi.it)

This work was supported in part by the Tuscany Region through the Programma operativo regionale (Por) del Fondo europeo di sviluppo regionale (Fesr) Toscana 2014–2020 Project SPaCe—Smart Passenger Center.

ABSTRACT Deep Neural Networks (DNNs) are the preferred choice for image-based machine learning applications in several domains. However, DNNs are vulnerable to adversarial attacks, that are carefully-crafted perturbations introduced on input images to fool a DNN model. Adversarial attacks may prevent the application of DNNs in security-critical tasks: consequently, relevant research effort is put in securing DNNs. Typical approaches either increase model robustness, or add detection capabilities in the model, or operate on the input data. Instead, in this paper we propose to detect ongoing attacks through monitoring performance indicators of the underlying Graphics Processing Unit (GPU). In fact, adversarial attacks generate images that activate neurons of DNNs in a different way than legitimate images. This also causes an alteration of GPU activities, that can be observed through software monitors and anomaly detectors. This paper presents our monitoring and detection system, and an extensive experimental analysis that includes a total of 14 adversarial attacks, 3 datasets, and 12 models. Results show that, despite limitations on the monitoring resolution, adversarial attacks can be detected in most cases, with peaks of detection accuracy above 90%.

INDEX TERMS Attack detection, anomaly detection, graphics processing unit, deep Neural Networks, adversarial attacks, image classification.

I. INTRODUCTION

Machine Learning (ML) is the enabling technology for a multitude of services, with increasing applications in security-critical domains, including financial market [74], healthcare [72], industrial automation [75], transportation systems [73], and energy management [76]. Most recent machine learning systems exploit Deep Neural Networks (DNNs), which notably outperform other machine learners especially for image classification [71], [75] and image processing applications [80].

Unfortunately, recent works have repeatedly shown that DNNs can be fooled by adversarial attacks [52], [58]. These attacks are primarily crafted against object classification where, given an input x of class y, the goal of the attack is to generate a new input x' that is similar to x but it is not classified as y [22]. In addition to classification tasks, adversarial attacks have been applied to other image-based ML applications such as self-driving agents [1], [55], [56]. Further, it has been recently shown that these attacks can be directly exercised against physical sensors (e.g., cameras). For example, the work in [3] showed that a classifier can be attacked through images displayed on cameras. Similarly, the works in [53] and [55] deceive autonomous cars by placing respectively tags on traffic signs, and stickers at intersections. Last, authors of [54] tricked a facial biometric system using an eyeglass frame with a colored pattern.

Consequently, it is crucial to understand how adversarial attacks can be perpetrated, and consequently build security defenses. Typical defenses aim at either making the DNN robust against adversarial attacks, or at detecting that the input is an adversarial one [52]. These approaches operate either on the training dataset, enriching the training phase with additional adversarial data to improve model robustness [25], or on the model itself e.g., appropriately building the model to identify adversarial inputs [37]. However, there is plenty of experimental evidence that shows the problem is still open: attacks can still go undetected even when applying the most recent security defenses [86], [87].

In this paper, we present an approach to the detection of adversarial attacks which is based on a new perspective, and that is complementary to existing approaches. Our

The associate editor coordinating the review of this manuscript and approving it for publication was Hossein Rahmani.

approach monitors performance indicators of the Graphics Processing Unit (GPU) to discover possible fluctuations of the monitored values when adversarial inputs are processed by the DNN model instead of legitimate inputs. The overall idea relies on the common knowledge that activation patterns of DNN neurons are different when classifying a legitimate input or an adversarial input [50]–[52]. We first hypothesize that i) the different activation patterns alter the computational load on the GPU which executes the DNN model, and ii) such alterations can be detected thanks to a software monitor and an anomaly detector. Then, we confirm our hypothesis with extensive experiments, to show the ability to detect adversarial attacks under different conditions. More precisely, to confirm our hypotheses we operate as follows:

- First, we present the software monitor to observe and log GPU performance indicators, and we identify the indicators more relevant for our purpose.
- Then, using the library in [2], we compute adversarial images for 14 adversarial attacks, on 3 image datasets and for 12 DNN models.
- Next, we feed the images to the DNN models, and we observe the GPU activity with a software monitor. Overall, we collect approximately 10 GB of data that describe the GPU activity when the DNN models are processing either the image datasets or adversarial images.
- We then process such data with a supervised anomaly detector to provide evidence that adversarial attacks can be detected through the analysis of GPU performance indicators. In the majority of cases, it is possible to detect ongoing attacks, despite detection accuracy ranges from above 90% to less than 60% depending on the DNN model, the attack, and the image dataset.
- Additionally, we observe how probing resolution often ends up being too coarse-grained. Consequently, we foresee that the availability of dedicated hardware/software probes with increased resolution could drastically improve detection accuracy.

As additional contribution, we believe that the data logged from GPU performance indicators, released at [5], can be used as reference for attack detection tasks: in fact, it is the first publicly available dataset that reports on GPU activity during image classification. We release ∼130 GB of adversarial images, and ∼10 GB of tabular data logged from the GPU. Generating all the data required multiple weeks of 24/7 execution: therefore, having it readily available can represent a valid accelerator and a benchmark for the work of researchers which aim to face the same or similar challenges. In addition, we release a software suite [4] to reproduce all the data of this paper, from generation of attacks datasets to GPU logging and analysis.

The rest of the paper is organized as follows. Section II presents background notions and positions our paper with respect to existing approaches. Section III describes our monitoring system, letting Section IV to present the experimental approach. Section V discusses results, whereas Section VI concludes the paper and debates on existing limitations and future works.

II. BACKGROUND, MOTIVATIONS AND STATE OF THE ART

We present background notions on adversarial attacks and defenses (Section II.A and Section II.B), and on the detection of attacks achieved via anomaly detection (Section II.C). Then, at the light of the presented notions, we review motivations and novelty of our paper (Section II.D), and we identify how it differentiate from related works on this subject (Section II.E).

A. ADVERSARIAL ATTACKS

DNNs are vulnerable to adversarial attacks, in which inputs (images, texts, tabular data, etc.) are deliberately modified to produce a desired response by the DNN model, while being perceptually indistinguishable from the legitimate inputs [3], with few exceptions e.g., adversarial patches [20] where colored patches are sticked on the input image. If adversarial attacks act without knowledge of the DNN model, they are named as black-box attacks; otherwise, if they exploit specific weaknesses of a given DNN model, they are white-box attacks.

Especially for image classification, adversarial attacks can deceive a target model to produce completely wrong predictions by adding small perturbations to the legitimate image [1], [38], [52]. Adversarial attacks to images usually target classification tasks, and are organized in three categories: poisoning, extraction and evasion attacks [2], [38], [52]. In poisoning attacks, attackers deliberately manipulate the training data to significantly decrease the performance of the DNN model, trigger targeted misclassification, and insert backdoors and neural trojans. Extraction attacks instead aim to develop a new model, starting from a proprietary black-box model; the new model emulates the behavior of the original model. In this paper, we focus on evasion attacks only, which target DNNs (typically, classifiers) while they are performing their task.

Evasion attacks modify the input to a DNN model such that it is misclassified, while keeping the modification as small as possible [38]. Evasion attacks can be black-box or white-box. Black-box evasion attacks are mostly based on the possibility to input any image at will and acquire classification results. An attacker can then leverage the acquired results to design more effective adversarial examples to fool the target DNN model [27]. White-box evasion attacks assume that the attacker has full access to the model, including the architecture and its parameters [22].

B. DEFENCES AGAINST ADVERSARIAL ATTACK

Several security defenses against adversarial attacks were proposed in recent years and they are usually categorized in the three groups [2] below.

Model hardening generates a new DNN model with better robustness properties than the original one; typically, the

DNN is trained using a dataset enriched with adversarial examples [25].

Data preprocessing aims to increase robustness by applying transformations of the inputs and labels at test and/or training time, before feeding the input to the DNN model. For example, in [37] the input image is transformed before it is processed by the classifier.

Runtime detection of adversarial samples extends the original classifier with a detector to check whether a given input is adversarial or legitimate. For example, it could determine whether the model is under attack by tracking the activations of the neurons in hidden layers (the called deep features) of the target DNN [39], or by observing the output distributions of the hidden neurons in a classifier [40].

C. ANOMALY DETECTION

Anomaly detection refers to the problem of finding patterns in data that do not conform to the expected (normal) behavior [59]. The underlying assumption is that pattern changes are caused by specific and non-random factors, such as the activation of ongoing attacks. Consequently, in the last decade, the detection of attacks, intrusions and failures using anomaly detection has been largely explored especially in case of trace logs [77], [78], or traffic data [79]. Anomaly detectors are often applied to detect intrusions [65], [67] or to predict failures [60], based on the hypothesis that the activation of a fault or an ongoing attack generates increasingly an anomalous performance-related behavior.

Additionally, anomaly detectors have been recently used to detect attacks to DNNs [62], [64] or other classifiers [63], by observing anomalies in the input data distribution or in the activation of hidden layers.

D. MOTIVATION AND NOVELTY OF OUR CONTRIBUTION

Adversarial attacks introduce small perturbation in images, so that the DNN model activates alternative neurons which are typically not activated by legitimate inputs [40], [50]. The neuron activation patterns of DNNs which process an adversarial input are quite different from those given by legitimate inputs of the same class [40]. From this observation, we hypothesize and experimentally show that i) given a DNN model that runs on a GPU, adversarial attacks to the DNN model generate a GPU load which is different than the GPU load obtained with legitimate inputs; ii) this leads to different values collected from the performance indicators of the GPU, and iii) such different values can be detected by an anomaly detector. This allows building a runtime detector of adversarial samples based on GPU performance indicators. Our approach is complementary to existing defense mechanisms reviewed in Section II.B. In fact, it collects values of GPU performance indicators with no access to the model and to the input data, and it executes independently from the DNNs executing on the GPU.

Further, to be aligned with the vast majority of works on adversarial attacks [18], [20], [22], [23], [25], we consider image classification models.

E. RELATED WORKS

To the best of our knowledge, our work is the first attempt to propose a runtime detector of adversarial images by observing anomalies from GPU performance indicators. The only work which we deem close to our approach is the recent work [1] which, amongst other contributions, observes spikes in GPU memory overhead and GPU utilization overhead, and shows that adversarial attacks can be detected observing the increased usage of GPU resources.

The most relevant differences from our work are three. First, to our understanding the models in [1] are trained on datasets augmented with the adversarial examples for robustness, and attacks are launched while running defensive methods. This may be the reason for the visible spikes of the monitored GPU performance indicators in [1], which we do not confirm in our experiments; for example, differently from [1] we believe that observing the GPU memory overhead is not useful. Second, the detection approach in [1] is based on static thresholds: instead, we use ML algorithms for anomaly detection, which is clearly a more powerful approach. Third, the study in [1] is limited to 3 DNN models and 5 attacks on a single dataset, while we use 12 DNN models, 14 attacks and 3 datasets; this is particularly relevant as we record significant differences depending on the model, attack, and dataset in use.

III. MONITORING FOR ADVERSARIAL ATTACKS

A. OVERALL ARCHITECTURE

To reach our goal, we need to: i) build a GPU monitor which exploits drivers and software probes to periodically collect performance indicators from the GPU, and ii) design and train a runtime anomaly detector which detects anomalies caused by adversarial attacks. We notice that the detection of an anomaly should trigger a response: however, response strategies are outside the scope of this work.

We present our *GPU Monitor and Runtime Anomaly Detector* in Figure 1. First, we observe that it does not interact with the existing components of DNN-based applications, which we depicted in the upper-left side of Figure 1; for example, applications can rely on PyTorch or Tensorflow. When performing a classification, the predicted class of the input image should be paired with the output of the anomaly detector. The *GPU Monitor* provides information from the GPU, whereas the *Runtime Anomaly Detector* uses such information to decide on the image legitimacy, i.e., if it is an adversarial image or a legitimate image.

B. THE GPU MONITOR

The GPU Monitor relies on specific drivers or probes that can periodically query the GPU and retrieve values of performance indicators. Those drivers clearly depend on the GPU hardware and on the operating system. In our approach, we use a GPU NVIDIA Quadro RTX 5000 and related drivers and tool suites on Linux OS. However, our approach fits all

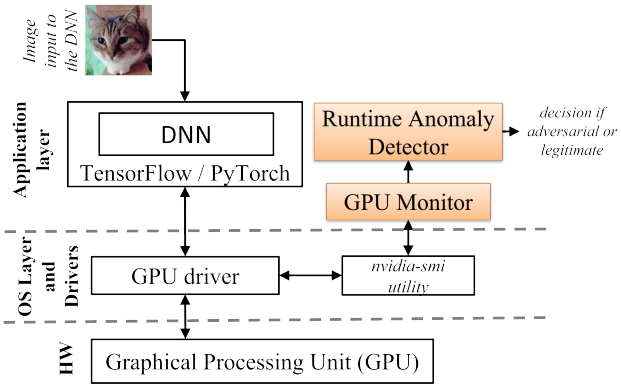


FIGURE 1. The GPU monitor and runtime anomaly detector.

recent NVIDIA GPUs, with differences only on the proper selection of performance indicators.

The monitoring probes are provided by the utility NVIDIA System Management Interface (NVIDIA-smi, [46]). We configure NVIDIA-smi to collect all the available indicators for our GPU. Further, we enable on the GPU: i) the Error Correcting Code (ECC) mode, which allow gathering error counts for various types of ECC errors, ii) the persistence mode, so that the NVIDIA driver remains loaded even when no active clients exist: this minimizes the driver load latency associated with running the dependent applications (i.e., CUDA programs) and increases repeatability of experiments, and iii) the accounting flag, which computes statistics on individual processes running on the GPU.

The GPU Monitor can acquire periodically performance indicators from five probes with NVIDIA-smi, as follows:

- *query-gpu (70 indicators):* indicators about current GPU status and utilization,
- *supported-clocks (7 indicators)*: the list of clocks combinations that the GPU can operate on, which indicates the frequency at which the different parts of the GPU are running,
- *compute-apps (8 indicators)*: information on the processes which are currently running on the GPU,
- *accounted-apps (11 indicators)*: details on the process which previously accessed the GPU for computation,
- *retired-pages (8 indicators)*: the memory pages that have been retired. Pages are retired in case of double bit errors or multiple single bit errors which could not be corrected by the available SECDED (Single Error Correction-Double Error Detection) mechanism.

We rework the GPU Monitor to i) minimize the number of performance indicators to be processed, and ii) create a unique output out of the different probes. The following four steps are valid for our setup, and it is recommendable to cross-check them for different systems.

First, we observe that accounted-apps and retired-pages return *null* values for most of the indicators in most of our experiments. Therefore, we discard those two probes.

Second, we observe that query-gpu has an average sampling period of 30 milliseconds (it outputs 33 times per seconds), while supported-clocks and compute-apps have a sampling period of approximately one second. Additionally, query-gpu returns 1 csv (comma-separated value) line for each individual output i.e., it produces an average of 33 lines per second. Instead, in the same timeframe supported-clocks and compute-apps return hundreds of csv lines. To tackle these issues, our GPU Monitor aggregates the outputs of the probes as follows. Each output of query-gpu matches an output from the GPU Monitor. The GPU Monitor enriches the output with aggregated information from all the csv lines produced by supported-clocks and compute-apps in their most recent sampling period. The aggregated information is the maximum, minimum, mean and the standard deviation of each numerical column in the csv lines of supported-clocks and compute-apps.

Third, the GPU monitor discards all timestamps, because our study does not focus on the analysis on time series; moreover, using the timestamp as a feature to train an anomaly detector would be a trivial mistake [41]: the detector should not use the timestamp to decide on the image, because the values learned from such feature have no relevance when it is deployed after training.

Fourth, we observe that most of the performance indicators do not carry informative content that is useful for the detection of anomalies. In fact, values of many indicators are constant throughout all experiments and do not fluctuate at all: those can be dropped by the GPU Monitor.

We summarize the remaining 12 performance indicators in Table 1. Noteworthy, Table 1 does not include any performance indicator from compute-apps. However, it is recommendable to repeat the selection of monitored features in case of novel experiments in different settings. For example, the GPU memory in use is a relevant indicator in [1] to perform detection, whereas *used_gpu_memory* from compute-apps [46] is constant through the execution of a DNN model in our experiments.

C. RUNTIME ANOMALY DETECTOR

The Runtime Anomaly Detector elaborates the latest output of the GPU monitor, to decide if a DNN was fed with adversarial images or legitimate images in the last sampling period (30 milliseconds, from Section III.B).

For the purpose of this paper, we run the Runtime Anomaly Detector offline i.e., we first save data from the GPU Monitor and then we apply anomaly detection on such data. While the architectural structure of Figure 1 does not change, this allows storing data and comparing multiple anomaly detection algorithms.

We rely on supervised anomaly detection, which is more consolidated in the literature and offers better detection performance than unsupervised alternatives [81], [82]. Supervised algorithms take advantage of labels in training data: in our case, labels refer to the legitimacy of images that are processed by the DNN model. The data from the GPU Monitor (and then processed by the Runtime Anomaly Detector) is

TABLE 1. Our selection of GPU performance indicators, collected from the GPU monitor.

GPU Performance Indicator (Feature)	Description
Indicators from probe query-gpu	
utilization.gpu	Percent of time during which kernels (applications) were executing in the last sampling period
utilization.memory	Percent of time during which the device memory was being read or written in the last sampling period
temperature.gpu	GPU temperature
clocks.current.graphics	Current frequency of graphics (shader) clock
clocks.current.sm	Current clock frequency of the Streaming Multiprocessor (the computing module of the GPU)
clocks_throttle_reasons.active	Information about factors that are reducing the frequency of clocks, reported in a bitmask
clocks_throttle_reasons.hw_slowdown	It indicates that, to reduce the frequency of core clocks, hardware slowdown is engaged e.g., because of high temperature.
clocks_throttle_reasons.hw_thermal_slowdown	
clocks_throttle_reasons.sw_power_cap	The frequency of clocks may be reduced because the GPU is consuming too much power
clocks.current.video	Current frequency of video clock
power.draw	Last measured power draw for the entire board
fan.speed	Speed the device's fan is intended to run at
Indicators from probe supported-clocks	
mem_clock_min	Minimum, average and standard deviation of supported memory clocks combinations
mem_clock_avg	
mem_clock_std	
graphics_clock_avg	Average frequency and standard deviation of supported graphics (shader) clocks combinations
graphics_clock_std	

labelled as *legitimate* if no adversarial images are processed in the related probing interval, and *adversarial* otherwise.

Selecting the most suitable anomaly detection algorithm is difficult and time-consuming, because it usually requires comparison studies and empirical evaluation of multiple alternatives [65], [68]. Therefore, we run preliminary tests by using a wide variety of supervised anomaly detectors [83], [84] as Gradient Boosting, k-NN, Random Forests, ADABoost, LDA, Naïve Bayes, Logistic Regression, Support Vector Machines, Multi-Layer Perceptron, and neural networks enhanced with entity embedding. Results of preliminary tests suggested to rely on neural networks enhanced with entity embedding [47], because they achieved significantly better results in almost all our preliminary tests. Consequently, we will proceed with neural network with entity embedding from now on, while results with other approaches are available for reference at [4]. Entity embedding maps similar values in the embedding space to reveal the intrinsic properties of the categorical (i.e., discrete) variables [47]. This improves fitting of neural networks on tabular data, especially when the data is sparse and/or statistics are unknown. The Python code to execute the Runtime Anomaly Detector was crafted starting from [48] and it is available at [4].

IV. GPU PERFORMANCE INDICATORS: DATASET GENERATION

We describe the experimental process that generates the labelled datasets of GPU performance indicators (on such datasets, we apply anomaly detection as from Section III.C). First, we present our general strategy (Section IV.A). Then we present the 3 image datasets (Section IV.B), the 12 DNNs models (Section IV.C), and the 14 adversarial attacks (Section IV.D). The source code to reproduce these steps is available at [4], while data and trained models can be retrieved from [5].

A. METHODOLOGY

1) SELECTION OF IMAGE DATASETS AND CLASSIFIERS

We start by selecting 3 image datasets, detailed in Section IV.B. Each dataset is partitioned into a train set and a test set. We train 4 DNNs on each of the 3 train sets separately, obtaining $4 \cdot 3 = 12$ trained DNN models i.e., 12 classifiers. Obviously, these classifiers are ready to be exercised on the 3 test sets. From now on, we call the 3 test sets as *Legitimate Image Sets* (LISs).

2) CREATION OF ADVERSARIAL IMAGE SETS

Next, we select 14 adversarial attacks. We apply each adversarial attack on each DNN model and its corresponding LIS. This generates, for each adversarial attack and each of the 12 models, an entire copy of the LIS, but composed exclusively of adversarial images. Last, we create the Black and White datasets: they are equivalent to the LISs in images shape and number, but they contain respectively entirely black images and entirely white images. This will turn useful as it will show the different behavior of the GPU in outlier cases. Sets composed either by black, white or adversarial images are called from now on *Original Adversarial Image Sets* (OAISs).

Often, adversarial attacks fail and produce the same classification output of the legitimate image; instead, we are especially interested in detecting adversarial attacks that alter such *classification* output. For this reason, we select adversarial images from the OAISs for which the classifier decides differently and erroneously than on the LISs. More precisely, given the classifier c, any legitimate image x of class y, and the corresponding adversarial image x', we select all x' such that $c(x) \neq c(x')$ and $c(x') \neq y$. For each adversarial attack, these images are then replicated to reach datasets of the same size of the LISs. From now on, we call the datasets created in this way as *Synthetic Adversarial Image Sets* (SAISs).

3) DATA COLLECTION GOALS

At this point, we just have to exercise the 12 DNN models on LISs, OAIS, and SAISs while the GPU Monitor logs data. From past experience with tabular data and anomaly detection, we target roughly 800·000 rows of data logged from the GPU Monitor for each of the 12 DNN models. We target a 50-50 split between rows labelled as legitimate and adversarial: in other words, for each model, we aim to collect 400·000 rows from the GPU Monitor while images from LISs are being processed. The remaining ∼400·000 rows are logged while images from the OAISs and SAISs are

processed (∼200·000 rows each). Since we are considering 14 attacks plus the Black and White cases, each attack occurs roughly in 400·000/16 =∼25·000 rows. To reach these goals, the total execution time of the GPU Monitor is above 150 hours.

4) DATA COLLECTION APPROACH

We introduce strategies to eliminate possible bias in the measurements. For example, running first the LISs, and then the OAISs and SAIs, would be questionable: realistically, we should instead expect that attacks are mixed with legitimate data. Further, we need to avoid that attacks are recognized thanks to specific conditions of the GPU that are not related to their occurrence. As simple example, the GPU temperature typically starts at ∼30 °C and raises through time exceeding 60 °C. If we test on LISs at the beginning of the experiment, and only successively on OAISs and SAISs, the GPU temperature could be used to distinguish between the execution of legitimate or adversarial images with high accuracy but no actual value in the results.

The strategies to remove possible bias originate the procedure for data collection reported in Algorithm I and described below. Algorithm I exemplifies the operations for a single DNN model; this is repeated for all the models, and for each dataset in the LISs, OAISs, and SAISs.

Algorithm 1 GPU Data Collection Procedure, Repeated for Each DNN Model

```
def classify_on_LIS:
 load LIS
 start logging from GPU Monitor
 repeat classification for approx. 15 seconds
 stop GPU Monitor
 drop first and last 50 lines of GPU Monitor
 pause for 2 second

def classify_Adversarial(attack n):
# executed for each attack in the OAIS and SAIS. Works
# the same of classify_on_LIS, but uses images
# from OAISs and SAISs as inputs

main:
 load trained DNN model
 repeat 30 times:
    classify_on_LIS
    classify_Adversarial(attack 1)
    classify_on_LIS
    classify_Adversarial(attack 2)
    classify_on_LIS
    classify_Adversarial(attack 3)
    classify_on_LIS
    ...
```

Algorithm I alternates classification on LISs, OAISs and SAISs. This way, the DNN model continuously alternates processing of legitimate images and adversarial images. For each image set (either from LISs, OAISs and SAISs) provided to the DNN model, the processing period must be sufficiently long to acquire enough data from the GPU monitor: we set 15 seconds of continuous execution of the classifier on the same image set, in which ∼500 lines are collected from the GPU Monitor.

The whole process is iterated multiple times (25 to 30). Additionally, image sets are always shuffled so that a different order is provided to the model. Further, when the 15 seconds slot completes, we discard the first and last 50 rows logged by the GPU Monitor. This cuts out possible transient behaviors of the GPU and the system. Last, once the 15 seconds of execution expire, the entire computation is paused for approximately 2 seconds to reduce the risk of any residual transient condition of the system.

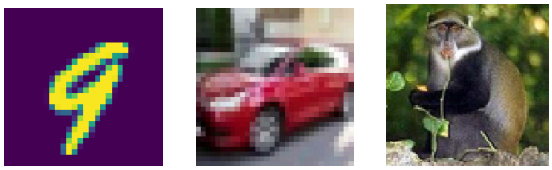
In the pseudo-code in Algorithm I, the function *classify_on_LIS* loads a Legitimate Image Set and performs classification while the GPU Monitor is logging. The same procedure is executed on the other image sets, by the function *classify_Adversarial*, with the difference that each time it is invoked, it works on a different attack i.e., a different image set amongst those in OAISs and SAISs.

5) HARDWARE AND SOFTWARE PLATFORM

All the computations are performed on a Dell Precision 5820 Tower with an Intel I9-9920X, GPU NVIDIA Quadro RTX5000 with 24GB VRAM, 128GB RAM and Ubuntu 18.04 with kernel 5.4.0, NVIDIA driver 450.119.03 and runtime CUDA 11.0.

B. LEGITIMATE IMAGE SETS (LISs)

Our Legitimate Image Sets are the test sets of the three datasets MNIST [13], CIFAR-10 [14], and STL-10 [15]. They are commonly used for image classification tasks and are highly appreciated because of the relatively low size of the images, which makes training time reasonably short. This aspect comes particularly handy for our study in which we need to train 12 models and apply several attacks. Example images from the three datasets are in Figure 2, while the datasets are described below.



a) MNIST b) CIFAR-10 c) STL-10

FIGURE 2. Sample images from the three LISs.

The MNIST database of handwritten digits has a training set of 60·000 examples and a test set of 10·000 examples, all of 28 × 28 pixels and in black/white only. All digits have been size-normalized and centered in a fixed-size image [13]. In general, this dataset is considered easy to classify, even by DNNs with only few convolutional layers.

The CIFAR-10 dataset consists of 60·000 32 × 32 colored images from 10 classes, with 6·000 images per class. There are 50·000 training images and 10·000 test images [14].

STL-10 consists of 5·000 training images and 8·000 test images. Images are colored and of 96 × 96 size [15]. The training set of STL-10 contains far less images than

MNIST and CIFAR-10; this makes supervised classification on STL-10 harder than in the previous two cases.

C. DNN MODELS

We select four DNNs implemented in PyTorch [12], [21] for each of the three LISs, according to the following criteria: i) one simple DNN, from tutorial examples available; ii) three state-of-the-art DNNs that are deeper and show higher accuracy. The outcome of our selection is summarized in Table 2.

TABLE 2. Details of the DNN models trained on the image sets MNIST, CIFAR-10, STL-10.

Acronym	Dataset	Accuracy	DNN	Trained from	From
MNIST-1	MNIST	0.979	2 conv. layers	scratch, using the settings proposed by the authors	[19]
MNIST-2		0.983	ResNet		[31]
MNIST-3		0.990	2 conv. layers		[32]
MNIST-4		0.992	9 conv. layers		[33]
CIFAR-1	CIFAR-10	0.572	2 conv. layers		[34]
CIFAR-2		0.955	ResNet		[35]
CIFAR-3		0.954	DenseNet		[35]
CIFAR-4		0.926	MobileNetV2		[35]
STL-1	STL-10	0.707	6 conv. Layers		[36]
STL-2		0.782	ResNet	transfer learning from ImageNet	[42]
STL-3		0.821	DenseNet		[42]
STL-4		0.814	MobileNetV2		[42]

For the MNIST dataset, we select the following DNN models of incremental accuracy:

- MNIST-1 is provided at [19], and it is composed of just 2 convolutional layers and 2 linear layers.
- MNIST-2 instead uses a user-defined ResNet [43] for MNIST, available at [31].
- MNIST-3 from [32] and MNIST-4 from [33] are custom networks, with respectively 2 and 9 convolutional layers.

For CIFAR-10, we select:

- CIFAR-1: the DNN provided as example on the PyTorch website [34], with just 2 convolutional layers.
- CIFAR-2, CIFAR-3 and CIFAR-4 are implementations, available at [35], of the three DNNs ResNet18 [43], DenseNet121 [45] and MobileNetV2 [44]. A normalization factor is applied to the images, and training is augmented with random cropping and random horizontal flipping [35].

For STL-10, we select:

- STL-1: a classifier for STL-10 that we found online at [36] with 6 convolutional layers.
- STL-2, STL-3 and STL-4 are ResNet18, DenseNet161 and MobileNetV2, obtained with transfer learning from the ResNet18, DenseNet161 and MobileNetV2 pretrained on the ImageNet dataset and available on the PyTorch model zoo [42]. We transfer-learned from pretrained models, because of the difficulties in retrieving optimized implementations of DNNs for STL-10, as instead we did for CIFAR-10.

D. ATTACKS LIST

We select 14 evasion attacks from the Adversarial Robustness Toolbox (ART, [2]), a Python library originally developed by IBM and now maintained by the Linux Foundation. ART implements tools to craft black-box and white-box adversarial attacks against DNNs, alongside with suggesting defenses. Attacks implemented in ART can be invoked by providing as input the DNN and other parameters e.g., the loss/optimization functions and the size of the input images. In our study, we use ART 1.5.1 and PyTorch 1.7.1.

Each of the 14 attacks has modifiable parameters [19] to tweak in order to be more effective against the target DNN model or to reduce the computational time needed to create the adversarial sample. Especially, all attacks except the Adversarial Patch [20] aim at forging an adversarial image that is as close to the legitimate image as possible, so we tune attacks to find a compromise between attack efficacy and image alteration. Table 3 reports the attacks parameters we used in this study. When attacks or parameters are not specified in the table, default values from [19] are used.

TABLE 3. Attack parameters.

Attack	Selected configuration of attack parameters
Adversarial Patch	*size*=0.4
Basic Iterative Method	MNIST: $\varepsilon = 0.1$, $\varepsilon_{step} = 0.02$, *max_iter* = 200 CIFAR-10, STL-10: $\varepsilon = 0.01$, $\varepsilon_{step} = 0.01$
ElasticNet	$iter_{max}$= 2
Fast Gradient Method	ε =0.09, ε_{step}=0.0001, *minimal*=True
HopSkipJump	$iter_{max}$ = 5, $eval_{max}$ =500
NewtonFool	$iter_{max}$ = 10, $\eta = 0.01$
Projected Gradient Descent	MNIST: ε =0.1, ε_{step} =0.1 CIFAR-10, STL-10: ε =0.01, ε_{step} =0.01
Spatial Transformation	MNIST: max(δu, δv)=10, translations=5, max(θ)=5°, rotations=5 STL-10, CIFAR-10: max(δu, δv) =5, translations=1, max(θ)=5°, rotations=1
Square	$\varepsilon = 0.1$, *restarts* = 10
Zoo	$\sigma = 0.8$, $l_{rate} = 0.5$, $iter_{max} = 1$, c = 0.6, *use_resize* = False, $v_h = 0.8$

We briefly review the 14 attacks, distinguishing between black-box and white-box. We introduce an acronym for each attack, to facilitate the discussion in the rest of the paper.

1) BLACK-BOX ATTACKS

The *Adversarial Patch* (APatch, [20]) consists in generating a colored patch that, when printed out and inserted into a natural scene, brings to misclassification of pictures taken of that scene. While various alternatives exist for creating patches, we use the algorithm in [20]. Figure 3 shows examples of the patched images and gives a clear understanding of this attack and the patch size. MNIST does not contain colored images and therefore in this case the usage of a colored patch is not meaningful.

HopSkipJump (HSJ, [17]) starts from a big perturbation and aims to reduce it to a minimum that still causes misclassification. The perturbation is reduced iteratively through binary searches. For computational reasons, we set $iter_{\max} = 5$ the maximum number of algorithm iterations, and $eval_{\max} = 500$ the maximum number of evaluations that estimate the current perturbation (Table 3).

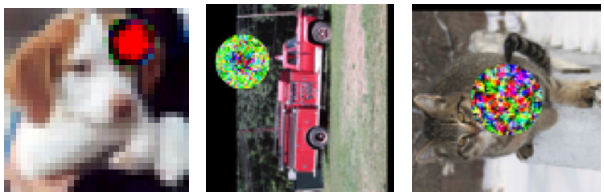

a) b) c)

FIGURE 3. Samples of the adversarial patch attack, computed for the models a) CIFAR-1, b) STL-1, c) STL-2.

Simple Black-box Attack (SimBA, [28]) randomly samples a vector from a predefined orthonormal basis and either adds or subtracts it to the target image.

Spatial Transformation (ST, [16]) aims to find the minimum spatial transformation that causes misclassification of the image. Repeatedly, the image is rotated by θ angle and shifted of (δu, δv) pixels, calculated as a percentage of the image size.

Square (SQ, [29]) is based on a randomized search scheme which, iteratively, performs localized square-shaped updates of the image. We set $\varepsilon = 0.1$ the maximum perturbation to be introduced during an iteration, and *restarts* $= 10$ the maximum number of restarts from random seed for each image.

Zeroth-Order-Optimization (ZOO, [27]) attack is a black-box version of Carlini and Wagner L_2 which relies on queries of the output confidence scores [2] and has several configuration parameters [19]. We do not apply ZOO in STL-1 to STL-4, because an overflow error was generated using ART.

2) WHITE-BOX ATTACKS

Carlini & Wagner L_2 (CWL_2, [22]) finds the adversarial instance by finding the smallest noise added to an image that will change the classification. The noise level is measured in terms of L_2 distance.

Carlini & Wagner L_∞ (CWL_∞, [22]) is similar to CWL_2 but it uses the L_∞ norm.

DeepFool (DFool, [23]) aims, for a given input, to perform an iterative linearization (exploiting the L_2 norm) of the classifier to generate minimal perturbations that are sufficient to change the classification label.

Elastic Net (ENet, [24]) is a variant of CWL_2 which aims at controlling, as distortion metric, the total variation in the perturbation (measured using the L_1 norm). We set $iter_{\max} = 2$ to limit the maximum number of iterations.

Fast Gradient Method (FGM) aims at controlling the gradient of the cost function with respect to the data, measured using the L_1 and L_2 norms and following the approach in [25]. We configure it to compute the minimal perturbation using input variation $\varepsilon = 0.09$ and descent step size $\varepsilon_{\text{step}} = 0.0001$.

Basic Iterative Method (BIM, [3]) is an iterative version of the Fast Gradient Sign Method [25], which produces an adversarial example by calculating the perturbation that maximizes the loss (with respect to the loss of the input image). The Basic Iterative Method extends the FGM attack by applying it multiple times until *max_iter* limit, with small step of size $\varepsilon_{\text{step}}$. Each intermediate result is cropped to ensure that it stays within the limits established by a hyper parameter ε, that sets the amount of perturbation allowed in the target image.

NewtonFool (NFool, [18]) uses the gradient-descent algorithm to find the perturbation that minimizes the probability of the original class. The tuning parameter η determines how aggressively the gradient descent attempts to minimize the probability of the original class.

Projected Gradient Descent (PGD, [26]) is an iterative extension of FGSM; its main strategy is similar to *BIM*.

3) ATTACK ACCURACY

As already said, by exercising the 14 attacks to the LISs, the OAISs is generated, exception made for the white and black cases. In Table 4 we report the classification accuracy (given *TN* True Negatives, *TP* True Positives, *FP* False Positives, *FN* False Negatives, we define accuracy $Acc = TP + TN/(TP + TN + FP + FN)$ [70]) of the DNN models when they elaborate image sets from the LISs and the OAISs. Noticeably, accuracy on the SAISs is zero by construction of such image set, and it is not reported in Table 4. The Black and White cases have 10% accuracy, which is random guessing in our classifiers with 10 classes, and are also not reported in Table 4.

We observe that some attacks as CWL_2, CWL_∞, SimBA and HSJ have more distinguishable impact on classification than others, dropping accuracy to below 10%. Other attacks as ST and, to a lesser extent, APatch have minimal effects. Last, BIM and PGD have significant impact on MNIST-1 and on STL-1 to STL-4, but are far less effective in the other cases, with accuracy values very close to the "no attacks" values. Noteworthy, these results strongly depends on: i) the attack and its configuration, ii) the DNN model, and iii) the image itself. However, we remark that rating adversarial attacks is not the contribution of this paper, and Table 4 is presented only because it is functional to the analysis of results in the successive Section.

V. ANOMALY DETECTION: EXECUTION AND RESULTS

A. SET-UP AND EXECUTION

Once the data logged from the GPU Monitor is available, it is possible to exercise the anomaly detector based on neural network with entity embedding as described in Section III.C.

We analyze the detection performance of the anomaly detector when the LISs, SAISs and the OAISs for each DNN model are processed.

We use a balanced dataset with 50% legitimate data and 50% adversarial data, where legitimate data is produced when legitimate images are being processed by the GPU, and adversarial data is produced when adversarial images are being processed by the GPU. Whenever needed, we select random samples from the legitimate data to maintain such balance: this happens for example when we differentiate the analysis

TABLE 4. Classification accuracy of the 12 DNN models on the LISs (row "no attacks") and on the OAISs (14 attacks). In grey cells, the attack efficacy is limited: accuracy drops less than 50% with respect to the "no attacks" case.

	MNIST-1	MNIST-2	MNIST-3	MNIST-4	CIFAR-1	CIFAR-2	CIFAR-3	CIFAR-4	STL-1	STL-2	STL-3	STL-4	
no attacks	0.98	0.98	0.99	0.99	0.57	0.95	0.95	0.93	0.71	0.78	0.82	0.81	
													average
APatch	*APatch not applicable on MNIST*				0.46	0.87	0.88	0.82	0.58	0.57	0.57	0.54	0.66
BIM	0.16	0.96	0.99	0.99	0.44	0.94	0.93	0.89	0.17	0.13	0.11	0.11	0.57
CWL_2	0.13	0.05	0.08	0.05	0.2	0.17	0.05	0.25	0.15	0.13	0.11	0.12	0.12
CWL_∞	0.29	0.13	0.39	0.45	0	0.1	0.09	0.09	0	0	0	0	0.13
DFool	0.39	0.05	0.01	0.01	0.28	0.1	0.1	0.11	0.18	0.34	0.39	0.47	0.20
ENet	0.02	0.01	0.01	0.01	0.19	0.4	0.41	0.41	0.15	0.13	0.11	0.12	0.16
FGM	0.69	0.4	0.91	0.94	0.19	0.4	0.4	0.36	0.16	0.15	0.13	0.14	0.41
HSJ	0.01	0.01	0.01	0.02	0.16	0.02	0.03	0.02	0.16	0.1	0.14	0.08	0.06
NFool	0.7	0.1	0.87	0.95	0.13	0.41	0.4	0.42	0.16	0.1	0.09	0.08	0.37
PGD	0.34	0.96	0.99	0.99	0.44	0.94	0.93	0.89	0.1	0.13	0.11	0.11	0.58
SimBA	0.01	0.01	0.49	0.64	0.02	0.08	0.09	0.11	0.01	0	0	0	0.12
ST	0.64	0.96	0.98	0.98	0.53	0.52	0.54	0.49	0.67	0.64	0.69	0.71	0.70
SQ	0.73	0.4	0.94	0.96	0.01	0.22	0.21	0.15	0.02	0	0	0	0.30
ZOO	0.65	0.18	0.67	0.84	0.2	0.54	0.56	0.49	*ZOO not applicable on STL*				0.51
Average	0.37	0.32	0.56	0.60	0.23	0.41	0.40	0.39	0.19	0.18	0.19	0.19	

by attacks type, because the legitimate data is much more than the data of an individual adversarial attack. Data is always shuffled when creating the train/validation/test split with proportions of [0.5, 0.2, 0.3]. Our datasets are balanced: therefore, we use the metrics accuracy Acc previously defined, the False Positive Rate $FPR = FP/(TN + FP)$ and the coverage $Cov = TP\ /\ (FN + FP)$ [70].

The entire source code for the repetition of our analyses is available at [4].

B. DISCUSSION OF RESULTS

First, we elaborate on detection of attacks without differentiating by attacks type. We describe results with the help of Figure 4. For MNIST-1 to MNIST-4 (Figure 4a), the anomaly detector can distinguish between legitimate or adversarial data with accuracy above 0.7, reaching $Acc = 0.81$ for SAISs in MNIST-3. This shows that our detector is able to understand when adversarial images or legitimate images are being processed by the GPU.

Instead, we observe a clear drop in detection accuracy when considering CIFAR-1 to CIFAR-4 (Figure 4b) and STL-1 to STL-4 (Figure 4c). To some extent, the anomaly detector can still distinguish when legitimate or adversarial images are presented to the DNN e.g., we have $Acc = 0.67$ on CIFAR-2, and $Acc = 0.65$ on CIFAR-3 for both OAISs and SAISs. However, the evidence is much smaller than with the MNIST. We believe the reason is the increased dimension of the trained DNN models, in terms of extracted features and number of weights. This complicates the detection of anomalies using GPU performance indicators with our monitoring resolution. Instead, we believe the model depth does not contribute to such accuracy drop: for example, the number of convolutional layer in CIFAR-1 and STL-1 is similar or smaller than in MNIST-1 to MNIST-4.

Importantly, we observe that there is no significant difference in results obtained with the SAISs and with the OAISs. This can be confirmed visually in the three graphs of Figure 4, by looking at the lines describing accuracy for SAIS and OAIS, which almost overlap. This means that the altered behavior of the GPU is observable whenever an attack is in progress, and not only when the attack is successful: if images are adversarial, the GPU is solicited in a different way than with legitimate images, independently of the outcome of the classifier. This behavior will also be confirmed in the next results we discuss.

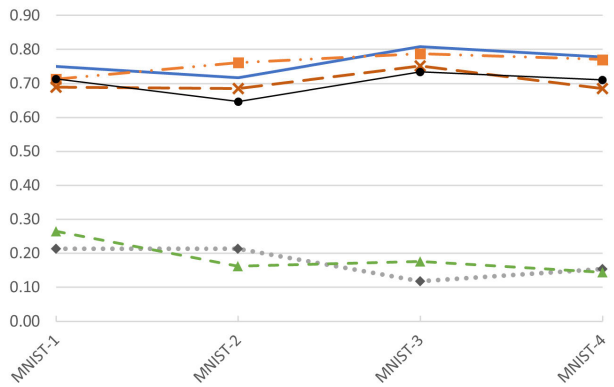
We now comment on the detection performance on each individual attack. Table 5 reports accuracy for the OAISs; the highlighted cells show detection accuracy above 80%. It is immediate to note that the black and white cases (images entirely black or entirely white) are usually outlier cases, obtaining very high detection accuracy. As expected, those images are so different from legitimate images that they solicit the GPU in a very distinguishable manner.

In general, detecting an individual attack is easier i.e., understanding the altered behavior on the target GPU in case of an individual attack is generally possible. Accuracy to detect attacks on MNIST dataset is above 80% in the majority of cases, with peaks above 90% (DFool, HSJ, NFool, SimBA). Accuracy drops in CIFAR and STL, but not as much as in Figure 4. There are few special cases, for example the accuracy achieved by the anomaly detector exceeds 90% when detecting CWL_∞ attack on CIFAR-2 to CIFAR-4. This suggests that the perturbation introduced by CWL_∞ into the image has also a significant impact on the GPU behavior, far different from the behavior under legitimate images.

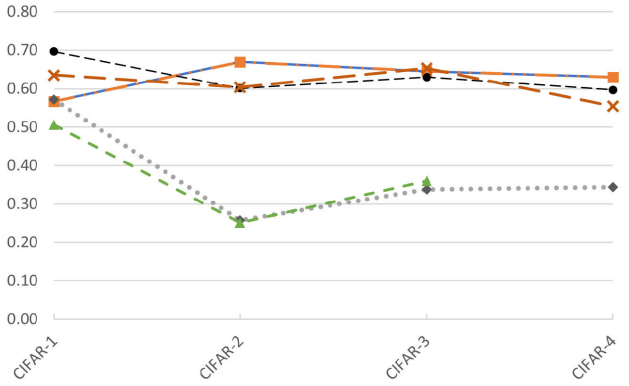
Last, in Figure 5 we show that accuracy scores are very close even when detecting individual attacks from the SAISs and OAISs. Given a model, we compute the difference between accuracy on SAISs and OAIS for each individual attack, and we report the highest and lowest values, respectively represented by a triangle and a square in Figure 5. The small difference is in the range $[-0.1, +0.1]$ with rare exceptions, as ZOO on MNIST-3 and MNIST-4.

C. RELEVANCE RANKING OF THE PERFORMANCE INDICATORS

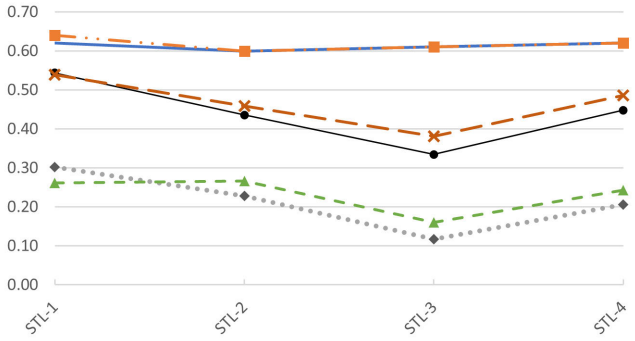
Not even the best anomaly detector out of the many alternatives in Section III.C can always detect attacks with



a) Accuracy, FPR and coverage of MNIST-1 to MNIST-4



b) Accuracy, FPR and coverage of CIFAR-1 to CIFAR-4



c) Accuracy, FPR and coverage of STL-1 to STL-4

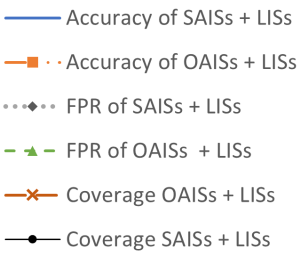


FIGURE 4. Detection performance on the entire SAISs and OAISs, for the entire set of attacks including white and black cases.

TABLE 5. Detection accuracy for each individual attacks, computed on the OAISs. Colored cells show accuracy above 80%.

	MNIST-1	MNIST-2	MNIST-3	MNIST-4	CIFAR-1	CIFAR-2	CIFAR-3	CIFAR-4	STL-1	STL-2	STL-3	STL-4
BLACK	0.92	0.77	1.00	1.00	0.75	0.97	0.96	0.94	0.89	0.84	0.94	0.84
WHITE	0.82	0.60	1.00	0.98	0.73	0.97	0.95	0.93	0.87	0.81	0.90	0.86
APatch					0.69	0.65	0.68	0.63	0.75	0.68	0.68	0.72
BIM	0.81	0.85	0.82	0.81	0.67	0.64	0.67	0.62	0.76	0.68	0.67	0.72
CWL_2	0.73	0.67	0.63	0.65	0.67	0.71	0.69	0.66	0.75	0.65	0.66	0.72
CWL_∞	0.75	0.71	0.60	0.66	0.65	0.97	0.95	0.91	0.72	0.66	0.69	0.71
DFool	0.79	0.91	0.84	0.82	0.65	0.71	0.70	0.67	0.74	0.67	0.67	0.71
ENet	0.75	0.87	0.84	0.81	0.67	0.72	0.71	0.67	0.76	0.66	0.69	0.69
FGM	0.74	0.86	0.85	0.84	0.68	0.72	0.70	0.67	0.76	0.66	0.67	0.70
HSJ	0.83	0.91	0.90	0.85	0.68	0.71	0.72	0.69	0.74	0.67	0.67	0.70
NFool	0.80	0.87	0.91	0.86	0.68	0.73	0.70	0.67	0.75	0.64	0.65	0.71
PGD	0.83	0.83	0.83	0.79	0.67	0.65	0.67	0.64	0.75	0.67	0.65	0.72
SimBA	0.85	0.90	0.78	0.69	0.71	0.68	0.69	0.63	0.75	0.65	0.64	0.72
ST	0.82	0.80	0.75	0.71	0.67	0.71	0.70	0.65	0.75	0.67	0.64	0.70
SQ	0.82	0.83	0.89	0.86	0.67	0.71	0.71	0.64	0.74	0.65	0.64	0.69
ZOO	0.73	0.75	0.77	0.70	0.67	0.72	0.71	0.66				
Average Acc	**0.79**	**0.83**	**0.80**	**0.77**	**0.67**	**0.72**	**0.71**	**0.67**	**0.75**	**0.66**	**0.66**	**0.71**
Standard Dev.	0.04	0.07	0.09	0.08	0.01	0.08	0.07	0.07	0.01	0.01	0.02	0.01
Average FPR	0.25	0.17	0.20	0.23	0.38	0.28	0.37	0.37	0.32	0.39	0.37	0.35
Average Cov	0.83	0.82	0.80	0.78	0.73	0.71	0.80	0.71	0.82	0.71	0.70	0.77

high accuracy. We motivate this result with the aid of Table 6, which reports the information gain [69] carried by the most relevant GPU performance indicators. Information gain can measure the relevance of individual features for classification [69], and ranges from 0 (no relevance at all) to 1 (the feature perfectly describes the label to be predicted).

Table 6 shows that *power.draw* is the most relevant performance indicator for the purpose of detection, while the others have a much lower information gain score. In general, all scores are quite low. This remarks that performance indicators of the GPU certainly have the potential to detect attacks, but there is a need of i) a finer-grained resolution of monitoring probes, and ii) a wider pool of non-constant indicators to

TABLE 6. Information gain of most relevant performance indicators. For brevity, we report only the top six.

performance indicators	MNIST-1	MNIST-2	MNIST-3	MNIST-4	CIFAR-1	CIFAR-2	CIFAR-3	CIFAR-4	STL-1	STL-2	STL-3	STL-4
power.draw	3.64E-02	1.23E-01	1.61E-01	1.45E-01	1.28E-03	6.64E-02	4.01E-02	3.14E-02	2.18E-02	2.41E-02	1.08E-02	2.00E-02
clocks.current.graphics	4.49E-03	0	4.28E-04	4.35E-03	2.19E-03	6.79E-04	2.51E-03	3.23E-03	4.21E-04	4.85E-04	7.19E-04	6.97E-04
clocks.current.sm	4.49E-03	0	4.28E-04	4.35E-03	2.19E-03	6.79E-04	2.51E-03	3.23E-03	4.21E-04	4.85E-04	7.19E-04	6.98E-04
clocks.current.video	3.42E-03	0	4.28E-04	7.31E-04	2.19E-03	5.69E-04	1.23E-03	6.07E-04	3.27E-04	1.57E-04	4.47E-04	6.66E-04
temperature.gpu	3.26E-03	1.08E-03	3.91E-03	1.36E-02	4.60E-03	2.32E-03	4.10E-03	4.61E-03	7.93E-04	9.67E-04	3.54E-04	1.07E-03
utilization.memory	3.93E-04	6.22E-04	1.55E-03	6.18E-05	2.56E-05	3.18E-04	1.48E-04	2.92E-04	4.60E-04	8.29E-05	1.35E-04	5.43E-04
utilization.gpu	3.86E-05	7.75E-04	6.11E-04	1.37E-04	9.54E-05	2.15E-04	0	3.03E-05	1.23E-04	2.87E-04	6.52E-04	8.15E-04

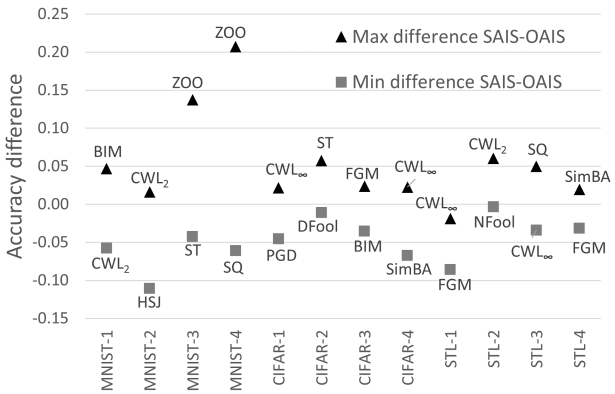


FIGURE 5. Difference in accuracy scores with SAISs and OAISs.

increase the number of features that the anomaly detector can process.

VI. CONCLUDING REMARKS

A. CONCLUSION

The objective of this work is to explore a new perspective for the detection of adversarial attacks towards ML-based applications, and especially against image classifiers. Our approach consists in building a GPU Monitor and Runtime Anomaly Detector: the GPU Monitor observes the GPU performance indicators, which are analyzed by the Runtime Anomaly Detector. This last component discovers possible fluctuations when adversarial inputs are processed by the DNN model instead of legitimate inputs. Relying on a large experimental campaign, with 12 DNN models, 3 datasets and 14 attacks, we showed the feasibility of our approach and we confirmed that it is possible to detect attacks.

B. THREATS TO VALIDITY

Monitoring probes offered by NVIDIA-smi retrieve fresh data approximately 33 times per second. Given the image size of MNIST, CIFAR-10 and STL-10, this frequency is appropriate only when multiple subsequent images are adversarial. In our experiments, it would not be possible to detect few adversarial images scattered amongst several legitimate images. While this may restrict the practical application of our work, we argue the following. First, this is still appropriate in some cases, for example with adversarial patches in the autonomous driving domain, where the patch may be applied to objects (e.g., traffic signs) and captured by the camera in successive frames [55]. Second, in case of larger images as in [30], [85], the processing time for a single image increases significantly, up to 10-30 images processed per seconds with an average GPU.

Another possible limitation is that the number and quality of GPU performance indicators that can be monitored is not adequate. Those could be improved for example observing subcomponents and consequently better characterize the GPU behavior; however, having the required probes is still bound to the available hardware and software. In addition, the monitoring probes we deployed prevent further analysis on the GPU behavior, for example to identify which GPU components are operating when the different parts of the DNN are solicited. We remark that these issues could be mitigated, as there is room to solve them with an appropriate hardware/software development, as discussed in our plan for future works in Section VI.C.

C. FUTURE WORKS

We are undergoing future works prioritized as follows. First, we are planning to repeat experiments with alternative GPU monitors, that may offer alternatives indicators to be monitored that can complement those explored with nvidia-smi. Candidate monitoring tools are the recent NVIDIA Nsight Systems [61] and NVIDIA NvBit [57]. Additionally, we plan to repeat the analysis by implementing alternative detection measures against evasion attacks on the target model, and in particular those already available in ART. This will allow measuring the combined effects of anomaly detection and state-of-the-art detection approaches.

On a longer term, we observe that increasing attention is given to bugs in neural network software [6], [7] and to GPUs faults [8], [9]. Consequently, we aim to study the impact of faults in the DNN model [10], [11] instead than attacks, to understand the extent faults can be detected by the GPU Monitor and Runtime Anomaly Detector.

REFERENCES

[1] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim, "An analysis of adversarial attacks and defenses on autonomous driving models," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, Mar. 2020, pp. 1–10.

[2] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. M. Molloy, and B. Edwards, "Adversarial robustness toolbox v1.0.0," 2018, *arXiv:1807.01069*.

[3] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 2016, *arXiv:1607.02533*.

[4] *Code and Detailed Instrutions to Reproduce the Experiments*. Accessed: Nov. 6, 2021. [Online]. Available: https://github.com/AndreaCeccarelli/gpu-monitor
[5] *All Datasets and Logs (Approximately 140 GBs)*. Accessed: Nov. 6, 2021. [Online]. Available: https://drive.google.com/drive/folders/15QIz6g2Meun_t_nbwoPf0t5v_PdzEaR5?usp=sharing
[6] X. Du, G. Xiao, and Y. Sui, "Fault triggers in the TensorFlow framework: An experience report," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2020, pp. 1–12.
[7] S. Jha, S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "AVFI: Fault injection for autonomous vehicles," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2018, pp. 55–56.
[8] J. E. R. Condia, B. Du, M. S. Reorda, and L. Sterpone, "FlexGripPlus: An improved GPGPU model to support reliability analysis," *Microelectron. Rel.*, vol. 109, Jun. 2020, Art. no. 113660.
[9] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2017, pp. 1–12.
[10] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: A flexible fault injection framework for TensorFlow applications," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2020, pp. 426–435.
[11] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "PyTorchFI: A runtime perturbation tool for DNNs," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2020, pp. 25–31.
[12] *PyTorch*. Accessed: Nov. 6, 2021. [Online]. Available: https://pytorch.org/
[13] Y. LeCun. (1998). *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[14] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009, pp. 1–60.
[15] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 215–223.
[16] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1802–1811.
[17] J. Chen, M. I. Jordan, and M. J. Wainwright, "HopSkipJumpAttack: A query-efficient decision-based attack," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1277–1294.
[18] U. Jang, X. Wu, and S. Jha, "Objective metrics and gradient descent algorithms for adversarial examples in machine learning," in *Proc. 33rd Annu. Comput. Secur. Appl. Conf.*, Dec. 2017, pp. 262–277.
[19] *ART Documentation V1.5.1*. Accessed: Nov. 6, 2021. [Online]. Available: https://adversarial-robustness-toolbox.readthedocs.io/en/latest/
[20] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," 2017, *arXiv:1712.09665*.
[21] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," 2019, *arXiv:1912.01703*.
[22] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.
[23] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2574–2582.
[24] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "EAD: Elastic-net attacks to deep neural networks via adversarial examples," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 10–17.
[25] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*.
[26] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017, *arXiv:1706.06083*.
[27] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," *CoRR*, vol. abs/1708.03999, pp. 15–26, Nov. 2017.
[28] C. Guo, J. Gardner, Y. You, A. G. Wilson, and K. Weinberger, "Simple black-box adversarial attacks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2484–2493.
[29] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein, "Square attack: A query-efficient black-box adversarial attack via random search," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, pp. 484–501, 2020.
[30] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, 2013.
[31] *ResNET in PyTorch*. Accessed: Nov. 6, 2021. [Online]. Available: https://github.com/LukasMut/ResNet-in-PyTorch/
[32] *PyTorch Examples*. Accessed: Nov. 6, 2021. [Online]. Available: https://github.com/pytorch/examples/
[33] *Handwritten Digit Recognition Using PyTorch*. Accessed: Nov. 6, 2021. [Online]. Available: https://medium.com/@ravivaishnav20/handwritten-digit-recognition-using-pytorch-get-99-5-accuracy-in-20-k-parameters-bcb0a2bdfa09
[34] *Welcome to PyTorch Tutorials*. Accessed: Nov. 6, 2021. [Online]. Available: https://pytorch.org/tutorials/
[35] *Train CIFAR-10 With PyTorch*. Accessed: Nov. 6, 2021. [Online]. Available: https://github.com/kuangliu/pytorch-cifar/
[36] *PyTorch Palyground*. Accessed: Nov. 6, 2021. [Online]. Available: https://github.com/ aaron-xichen/pytorch-playground/
[37] C. Guo, M. Rana, M. Cisse, and L. van der Maaten, "Countering adversarial images using input transformations," 2017, *arXiv:1711.00117*.
[38] D. Warde-Farley and I. Goodfellow, "Adversarial perturbations of deep neural networks," in *Perturbation, Optimization, and Statistics*, T. Hazan, G. Papandreou, and D. Tarlow, Eds. Cambridge, MA, USA: MIT Press, 2016.
[39] F. Carrara, F. Falchi, R. Caldelli, G. Amato, and R. Becarelli, "Adversarial image detection in deep neural networks," *Multimedia Tools Appl.*, vol. 78, no. 3, pp. 2815–2835, 2019.
[40] Z. Zheng and P. Hong, "Robust detection of adversarial attacks by modeling the intrinsic properties of deep neural networks," in *Proc. NeurIPS*, vol. 31. Red Hook, NY, USA: Curran Associates, 2018, pp. 7924–7933.
[41] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Unsupervised algorithms to detect zero-day attacks: Strategy and application," *IEEE Access*, vol. 9, pp. 90603–90615, 2021, doi: 10.1109/ACCESS.2021.3090957.
[42] *PyTorch Model Zoo*. Accessed: Nov. 6, 2021. [Online]. Available: https://pytorch.org/serve/model_zoo.html
[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[45] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[46] NVIDIA. *NVIDIA System Management Interface Program*. Accessed: Nov. 6, 2021. [Online]. Available: https://developer.download.NVIDIA.com
[47] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," 2016, *arXiv:1604.06737*.
[48] B. Lau. *The Right Way to Use Deep Learning for Tabular Data*. Accessed: Nov. 6, 2021. [Online]. Available: https://towardsdatascience.com/the-right-way-to-use-deep-learning-for-tabular-data-entity-embedding-b5c4aaf1423a
[49] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," 2017, *arXiv:1705.07874*.
[50] N. Das, H. Park, Z. J. Wang, F. Hohman, R. Firstman, E. Rogers, and D. H. Chau, "Bluff: Interactively deciphering adversarial attacks on deep neural networks," 2020, *arXiv:2009.02608*.
[51] S. Carter, Z. Armstrong, L. Schubert, I. Johnson, and C. Olah, "Activation atlas," *Distill*, vol. 4, no. 3, p. e15, 2019.
[52] M. Xue, C. Yuan, H. Wu, Y. Zhang, and W. Liu, "Machine learning security: Threats, countermeasures, and evaluations," *IEEE Access*, vol. 8, pp. 74720–74742, 2020.
[53] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1625–1634.
[54] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1528–1540.
[55] E. Ackerman, "Three small stickers in intersection can cause Tesla autopilot to swerve into wrong lane," *IEEE Spectr.*, vol. 1, 2019. [Online]. Available: https://spectrum.ieee.org/three-small-stickers-on-road-can-steer-tesla-autopilot-intooncoming-lane

[56] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Softw.*, vol. 34, no. 3, pp. 56–63, May/Jun. 2017.
[57] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "NVBit: A dynamic binary instrumentation framework for NVIDIA GPUs," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, New York, NY, USA, 2019, pp. 372–383, doi: 10.1145/3352460.3358307.
[58] K. Ren, T. Zheng, Z. Qin, and X. Liu, "Adversarial attacks and defenses in deep learning," *Engineering*, vol. 6, no. 3, pp. 346–360, 2020.
[59] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, 2009.
[60] R. Baldoni, L. Montanari, and M. Rizzuto, "On-line failure prediction in safety-critical systems," *Future Gener. Comput. Syst.*, vol. 45, pp. 123–132, Apr. 2015.
[61] *NSight Systems User Manual V1.3.1*, NVIDIA, Santa Clara, CA, USA, Jul. 2021.
[62] A. Paudice, L. Muñoz-González, A. Gyorgy, and E. C. Lupu, "Detection of adversarial training examples in poisoning attacks through anomaly detection," 2018, *arXiv:1802.03041*.
[63] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 1807–1814.
[64] D. Miller, Y. Wang, and G. Kesidis, "When not to classify: Anomaly detection of attacks (ADA) on DNN classifiers at test time," *Neural Comput.*, vol. 31, no. 8, pp. 1624–1670, 2019.
[65] T. Zoppi, T. Capecchi, A. Ceccarelli, and A. Bondavalli, "Unsupervised anomaly detectors to detect intrusions in the current threat landscape," *ACM/IMS Trans. Data Sci.*, vol. 2, no. 2, pp. 1–26, 2021.
[66] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *J. Artif. Intell. Res.*, vol. 46, pp. 235–262, Jan. 2013.
[67] N. B. Aissa and M. Guerroumi, "Semi-supervised statistical approach for network anomaly detection," *Proc. Comput. Sci.*, vol. 83, pp. 1090–1095, Jan. 2016.
[68] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS ONE*, vol. 11, no. 4, 2016, Art. no. e0152173.
[69] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the Gini index and information gain criteria," *Ann. Math. Artif. Intell.*, vol. 41, no. 1, pp. 77–93, 2004.
[70] D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," 2020, *arXiv:2010.16061*.
[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[72] L. M. Prevedello, S. S. Halabi, G. Shih, C. C. Wu, M. D. Kohli, F. H. Chokshi, B. J. Erickson, J. Kalpathy-Cramer, K. P. Andriole, and A. E. Flanders, "Challenges related to artificial intelligence research in medical imaging and the importance of image analysis competitions," *Radiol., Artif. Intell.*, vol. 1, no. 1, 2019, Art. no. e180031.
[73] A. E. L. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electron. Imag.*, vol. 19, pp. 70–76, Jan. 2017.
[74] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *Eur. J. Oper. Res.*, vol. 270, no. 2, pp. 654–669, Oct. 2018.
[75] M. Miškuf and I. Zolotová, "Comparison between multi-class classifiers and deep learning with focus on industry 4.0," in *Proc. Cybern. Informat. (KI)*, Feb. 2016, pp. 1–5.
[76] Y. Liu, C. Yang, L. Jiang, S. Xie, and Y. Zhang, "Intelligent edge computing for IoT-based energy management in smart cities," *IEEE Netw.*, vol. 33, no. 2, pp. 111–117, Mar./Apr. 2019.
[77] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "SwissLog: Robust and unified deep learning based log anomaly detection for diverse faults," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2020, pp. 92–103, doi: 10.1109/ISSRE5003.2020.00018.
[78] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "LogTransfer: Cross-system log anomaly detection for software systems with transfer learning," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2020, pp. 37–47, doi: 10.1109/ISSRE5003.2020.00013.
[79] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Raheem, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33789–33795, 2018, doi: 10.1109/ACCESS.2018.2841987.
[80] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Proc. Sci. Inf. Conf.* Cham, Switzerland: Springer, 2019, pp. 128–144.
[81] N. S. Arunraj, R. Hable, M. Fernandes, K. Leidl, and M. Heigl, "Comparison of supervised, semi-supervised and unsupervised learning methods in network intrusion detection system (NIDS) application," *Anwendungen Konzepte Wirtschaftsinformatik*, vol. 6, pp. 10–19, 2017.
[82] K. Lee, D. Booth, and P. Alam, "A comparison of supervised and unsupervised neural networks in predicting bankruptcy of Korean firms," *Expert Syst. Appl.*, vol. 29, no. 1, pp. 1–16, 2005.
[83] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerg. Artif. Intell. Appl. Comput. Eng.*, vol. 160, pp. 3–24, Jun. 2007.
[84] Y. C. Chang, K. H. Chang, and G. J. Wu, "Application of eXtreme gradient boosting trees in the construction of credit risk assessment models for financial institutions," *Appl. Soft Comput.*, vol. 73, pp. 914–920, Dec. 2018.
[85] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11621–11631.
[86] F. Tramer, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," 2020, *arXiv:2002.08347*.
[87] O. Bryniarski, N. Hingun, P. Pachuca, V. Wang, and N. Carlini, "Evading adversarial example detection defenses with orthogonal projected gradient descent," 2021, *arXiv:2106.15023*.

TOMMASO ZOPPI is currently a Research Associate with the Università degli Studi di Firenze. He is involved in several European/national funded and even industrial projects. His research interests include anomaly detection, security and safety, often applying standards to plan, design, develop, and implement appropriate architectures or software in the domain of critical systems. He currently serves as a member of a program committee of several international conferences.

ANDREA CECCARELLI received the Ph.D. degree in informatics and automation engineering from the Università degli Studi di Firenze, Florence, Italy, in 2012. He is currently a Tenured Assistant Professor in computer science with the University of Florence. His scientific activities originated more than 100 articles, which appeared in international conferences, workshops, and journals. His research interests include the design, monitoring, and experimental evaluation of dependable and secure systems, and systems-of-systems.