CuRTAIL: ChaRacterizing and Thwarting AdversarIal Deep Learning

Mojan Javaheripi, *Student Member, IEEE*, Mohammad Samragh, *Student Member, IEEE*, Bita Darvish Rouhani, *Student Member, IEEE*, Tara Javidi, and Farinaz Koushanfar, *Fellow, IEEE*

Abstract—Recent advances in adversarial Deep Learning (DL) have opened up a new and largely unexplored surface for malicious attacks jeopardizing the integrity of autonomous DL systems. This article introduces CuRTAIL, a novel end-to-end computing framework to characterize and thwart potential adversarial attacks and significantly improve the reliability (safety) of a victim DL model. We formalize the goal of preventing adversarial attacks as an optimization problem to minimize the rarely observed regions in the latent feature space spanned by a DL network. To solve the aforementioned minimization problem, a set of complementary but disjoint modular redundancies are trained to validate the legitimacy of the input samples. The proposed countermeasure is unsupervised, meaning that no adversarial sample is leveraged to train modular redundancies. This, in turn, ensures the effectiveness of the defense in the face of generic attacks. We evaluate the robustness of our proposed methodology against the state-of-the-art adaptive attacks in a white-box setting considering that the adversary knows everything about the victim model and its defenders. Extensive evaluations for analyzing MNIST, CIFAR10, and ImageNet data corroborate the effectiveness of CuRTAIL framework against adversarial samples. The computations in each modular redundancy can be performed independently of the other redundancy modules. As such, CuRTAIL detection algorithm can be completely parallelized among multiple hardware settings to achieve maximum throughput. We further provide an open-source Application Programming Interface (API) to facilitate the adoption of the proposed framework for various applications.

Index Terms—Deep learning, model reliability, adversarial samples, white-box attacks

1 Introduction

Security and safety consideration is a major obstacle to the wide-scale adoption of emerging learning algorithms in sensitive scenarios, such as intelligent transportation, healthcare, and video surveillance applications [1], [2]. While advanced learning techniques are essential for enabling coordination between autonomous agents and the environment, a careful analysis of their vulnerabilities and their reliability in face of adversarial attacks is still in its infancy.

Adversarial samples [3], [4], [5] are carefully crafted input instances which lead machine learning algorithms into misclassifying while the input changes are imperceptible to the human eye. For instance, in the case of traffic sign classifiers employed in self-driving cars, an adversary can add a specific imperceptible perturbation to a legitimate "stop" sign sample and fool the DL model to classify it as a "yield" sign, thus, jeopardizing the safety of the vehicle as shown in [1]. Thereby, it is highly important to identify and reject risky samples to ensure the integrity of DL models used in autonomous systems such as unmanned vehicles/drones.

This paper provides an end-to-end solution (called CuRTAIL) to characterize and thwart adversarial attacks for DL models in an online manner. Our proposed solution addresses three main challenges regarding the adversarial attacks in the context of deep learning.

• *The authors are with the University of California San Diego, La Jolla, CA 92093 USA. E-mail: {mojan, msamragh, bita, tjavidi, farinaz}@ucsd.edu.*

Manuscript received 7 June 2018; revised 28 Aug. 2020; accepted 2 Sept. 2020. Date of publication 15 Sept. 2020; date of current version 12 Mar. 2021. (Corresponding author: Bita Darvish Rouhani.)
Digital Object Identifier no. 10.1109/TDSC.2020.3024191

(i) Understanding the root cause of DL vulnerabilities to adversarial samples. Our hypothesis is that the vulnerability of Deep Neural Networks (DNNs) to adversarial samples originates from the existence of rarely explored sub-spaces in each feature map. This phenomenon is particularly caused by the limited access to labeled data and/or inefficiency of regularization algorithms [6], [7]. Fig. 1 provides a simple illustration of the partially explored space in a two-dimensional setup. We provide statistical analysis and empirically back up our hypothesis by extensive evaluations on various DL benchmarks and attacks.

(ii) Characterizing and thwarting the adversarial subspace for model assurance. A line of research has shown that there is a trade-off between the robustness of a model and its accuracy [8], [9]. Taking this into account, instead of making a single model that is both robust and accurate, we introduce a new defense mechanism called Modular Robust Redundancy (MRR). In MRR methodology, the victim model is kept *as is* while separate defender modules are trained to checkpoint the hidden features and assess the reliability of the victim's prediction. Each defender module characterizes the explored sub-space in the pertinent layer by learning the probability density function (PDF) of legitimate data points and marking the complement sub-spaces as rarely observed regions. Once such characterization is obtained, MRRs evaluate the input sample in parallel with the victim model and raise alarm flags for data points that lie within the rarely explored regions.

(iii) Just-in-time online defense against adversarial attacks. We propose CuRTAIL, the first end-to-end hardware-accelerated framework that enables robust and just-in-time defense against adversarial attacks on DNNs. CuRTAIL is devised

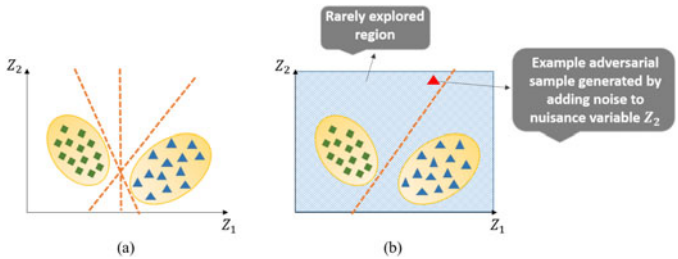

Fig. 1. (a) Data points (green and blue squares) can be easily separated in one-dimensional space. Extra dimensions add ambiguity in choosing the decision boundaries: all shown boundaries (dashed lines) result in the same classification accuracy but are not equally robust to noise. (b) The rarely explored space in a learning model leaves room for adversaries to manipulate the non-critical dimensions (Z_2 in this figure) and mislead the model by crossing the decision boundaries.

based on algorithm/hardware co-design to enable safe DL execution while customizing system performance in terms of latency, energy consumption, and/or memory footprint based on the available resource. CuRTAIL leverages FPGAs to provide fine-grained parallelism and just-in-time response by our defender modules. The customized data path for memory access on FPGA helps to improve the overall system energy efficiency.

We consider a *white-box* attack model where the attacker knows everything about the victim model including its architecture, learning algorithm, parameters, and defenders. This threat model represents the most powerful attack that can endanger real-world applications. We validate the security of our proposed approach for different DL benchmarks including MNIST, SVHN, CIFAR, and a subset of ImageNet data. The explicit contributions of this paper are as follows:

- Proposing CuRTAIL, the first algorithm/hardware co-design enabling online defense against adversarial samples for DL models. Our methodology is unsupervised and robust against the most challenging attack scenario to date.
- Introducing Modular Robust Redundancy as a viable countermeasure for adversarial attacks on DL. CuRTAIL uses dictionary learning and probability density functions to statistically detect abnormalities in the input data.
- Providing quantitative metrics to characterize the sensitivity of DL layers from a statistical point of view. Based on the result of our analysis, we provide new insights on the reason behind the existence of adversarial transferability.
- Implementing the first streaming-based DL defense using FPGAs. CuRTAIL devises an automated customization tool to adaptively maximize model robustness against adversaries while complying with the underlying hardware resource constraints, i.e., runtime, energy, and memory.
- Performing extensive evaluations in both black-box and adaptive white-box settings against various attack methodologies including Fast-Gradient-Sign [10], Jacobian Saliency Map attack [3], Deepfool [4], Basic Iterative Method [11], and Carlini&WagnerL2 [5], [12]. Thorough performance comparison on various hardware platforms including CPUs, GPUs, and FPGAs corroborates CuRTAIL's algorithmic practicality and system efficiency.
- Devising an automated accompanying API to facilitate adoption/integration of CuRTAIL for reliable realization of different DL applications by data scientists and engineers.[1]

An earlier version of CuRTAIL was presented in [13]. In this article, we extend CuRTAIL framework by: (i) Providing a thorough statistical analysis to elaborate on the root cause of DL model vulnerabilities to adversarial attacks (Section 3). The results of our analysis, in turn, sheds light on the transferability of adversarial samples in between different models (Section 6.6). (ii) Devising a sensitivity analysis module (Section 4.4). This added module, in turn, enables careful evaluation of layer-wise vulnerability to determine the most effective location for checkpointing purposes. (iii) Delineating the hardware architecture of latent (Section 5.1.1) and input checkpoints (Section 5.1.2). We further characterize the computation complexity and latency of each CuRTAIL's custom layer in Section 5.3. (iv) Extending our evaluations on new visual datasets with higher complexity in Section 6.3.

2 Background and Preliminaries

A machine learning model refers to a function f and its associated parameters θ that are specifically trained to infer/discover the relationship between input samples $x \in \{x_1, \ldots, x_N\}$ and the ground truth labels $y \in \{y_1, \ldots, y_N\}$. Each output observation y_i can be either continuous as in most regression tasks or discrete as in classification applications. Machine learning algorithms typically aim to find the optimal parameter set θ such that a loss function $\mathcal{L}$ that captures the difference between the output inference and ground-truth labeled data is minimized

$$\theta = \underset{\theta}{argmin} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i, \theta), y_i). \quad (1)$$

In this paper, we focus on state-of-the-art deep learning models due to their popularity in the realization of various autonomous learning systems. Consistent with the literature in this field, we specifically focus our discussions on the classification tasks using DL methodology. However, we emphasize that our proposed core concept is generic and can be used for reliable deployment of different learning techniques such as generalized linear models, regression methods (e.g., Lasso), and kernel support vector machines.

2.1 Deep Learning

Deep learning is an important class of machine learning algorithms that has provided a paradigm shift in our ability to comprehend raw data. A DL network is a hierarchical learning topology consisting of several processing layers stacked on top of one another. Each layer extracts features from its input and feeds it to the succeeding layer. This hierarchical structure gradually maps the input data samples to higher-level abstractions. The output from the last layer in the network is used for classification or regression purposes.

1. Implementation at https://github.com/Bitadr/DeepFense

The most common core processing layers used in DL models are*Convolution* and*Dense* layers. A Dense layer performs matrix-vector multiplication on its input. Convolution layers take channels of input images, convolve them with a set of 3-D kernels, and generate a stack of output channels. The output a linear layer is passed through an*Activation* layer which applies a non-linear function to the data, allowing the network to model non-linear and complex patterns. *Pooling* layers are implemented to reduce dimensionality as well as make the model invariant to local translation.

During training, model parameters are automatically learned in the service of the task. Training a DL network involves two main steps: (i) forward propagation, and (ii) backward propagation. In forward propagation, the raw values of the input data measurements are mapped to higher-level abstractions based on the current state of the DL parameters (θ). The acquired data abstractions are used to predict the inference label in the last layer of the DL network based on a Softmax regression. In backward propagation, an optimization is performed to update the DL parameter set θ such that the distance between network prediction (output of forward propagation) and the ground-truth label is minimzied. Once the DL network is fully trained, the model is employed as a classification oracle in the inference (execution) phase. During this phase, model parameters are fixed and prediction is performed through one round of forward propagation for each input sample. Attacks based on adversarial samples target the DL execution phase and do not involve any tampering with the training procedure.

2.2 Attack Models

Adversarial machine learning can be cast as a zero-sum Stackelberg game between the machine learning oracle (victim) and the attacker. Depending on the attacker's knowledge, the threat model can be categorized into three classes:

- *White-box attacks.* The attacker knows everything about the victim model including the learning algorithm, model topology and parameters, as well as the defense mechanism, the corresponding defender parameters.
- *Gray-box attacks.* The attacker knows the learning algorithm, model topology, and defense mechanism and has no access to the model/defender parameters.
- *Black-box attacks.* The attacker does not know anything about the pertinent machine learning algorithm, ML model, or defense mechanism. This attacker can merely obtain the outputs of the victim ML model by providing input samples. In this setting, the adversary can perform a differential attack by observing the output changes with respect to input variations.

A complete taxonomy of adversarial capabilities and goals are provided in [3], [14], [15]. In this paper, we consider the white-box threat model as the most powerful attacker that can appear in real-world machine learning applications.

In an adversarial setting, the attacker aims to find a perturbed adversarial sample (x^a) such that it incurs minimal distance from the source sample (x^s) while its corresponding output is sufficiently different to mislead the victim. Fig. 2 illustrates an example, where the image on the left is initially classified correctly as a dog by the victim model while adding a small amount of perturbation to the original image has misled the victim to infer it as a black swan (right image). Clearly, if the source instance is already misclassified by the victim model ($f(x^s, \theta) \neq y^*$), the adversarial problem becomes trivial. Therefore, we particularly focus on instances x^s that could have been classified correctly by the oracle before adding structured adversarial noises ($f(x^s, \theta) = y^*$).

Fig. 2. An example of (a) input data and (b) its corresponding adversarial sample. The added noise is imperceptible but can cause the victim model to misclassify.

Several attack mechanisms have been proposed in the literature to craft adversarial samples. We evaluate CuRTAIL performance against a variety of attacks to empirically confirm the generalizability of out unsupervised MRR methodology across a wide range of attacks. In particular, we have evaluated CuRTAIL against (i) Fast Gradient Sign (FGS) [10], (ii) Jacobian Saliency Map Attack (JSMA) [3], (iii) Deepfool [4], (iv) Basic Iterative Method (BIM) [11], and (v) Carlini&WagnerL2 adaptive attacks [5], [12] . In the following, we provide a brief explanation of the attack algorithms evaluated in this paper.

Fast-Gradient-Sign. FGS attack [10] crafts an adversarial sample as $x' = x + \epsilon \cdot Sign(\frac{\partial C}{\partial x})$, where C is the cost function of the neural network, and $Sign(\cdot)$ outputs the sign of its operand. The attack is parameterized by ϵ, which determines the amount of additive perturbation.

Basic Iterative Method. BIM [11] is an iterative version of FGS characterized by the number of iterative updates, n_{iters}, and the per-iteration perturbation coefficient, ϵ.

Deepfool. This algorithm iteratively modifies the input image based on a specific update rule to obtain an adversarial sample [4]. In each iteration, the perturbation vector $\frac{\partial C}{\partial x}$ is normalized and added to the sample. Deepfool is parameterized by the number of iterative updates n_{iters}.

Carlini&WagnerL2. This attack is formalized as a minimization problem where the objective is the L_2 norm of the perturbation vector. Carlini&WagnerL2[2] proposes an iterative method for solving this minimization objective. The detailed set of parameters for the attack is provided in [5].

3 Statistical Analysis of Adversarial Samples

Let us denote the output of the ith layer of a DL model given an input sample x by $f_i(x)$. One can construct a probabilistic density function $P_X(f_i(x))$ for each layer where X is a random variable drawn from the model input space. Our conjecture is that adversarial samples cannot lie in high-probability regions of the PDF function $P_X(.)$, which is learned using the samples drawn from legitimate training

2. For brevity we denote this attack as CarliniL2 in the paper.

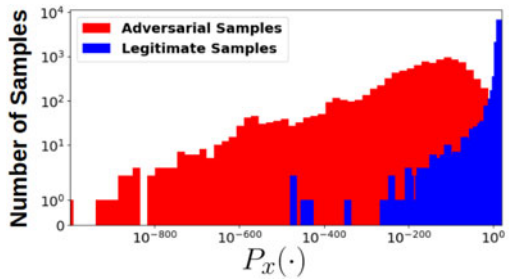

Fig. 3. Histogram of the estimated PDF for adversarial (red) and legitimate (blue) samples. Adversarial samples are generated using Deepfool for *Lenet* architecture. The PDF is learned in the second-to-last layer of the network.

data. More formally, the expected value of the probability corresponding to legitimate samples is higher than that of adversarial samples

$$E(P_X(f_i(x^s))) >> E(P_X(f_i(x^a))), \tag{2}$$

where $E(\cdot)$ is the expectation and x^s and x^a are safe and adversarial input samples, respectively.

This difference between the expected values of the probability distribution can be leveraged to characterize and thwart adversarial attacks. To do so, CuRTAIL approximates the PDF of each layer by training a set of defender modules, as will be explained in Section 4. Fig. 3 empirically validates the criterion outlined in Eq. (2). In this example, the PDF of benign samples in the second-to-last layer ($P_X(f_{N-1}(x))$, where N is the number of model layers) of the *LeNet* model is obtained by passing through legitimate MNIST data points and acquiring the corresponding activations. Once the PDF is learned, we generate adversarial samples x^a and compute $P_X(f_{N-1}(x^a))$. Fig. 3 depicts the probabilistic histogram of legitimate and adversarial samples based on the learned PDF. As shown, the expected value of legitimate samples is orders of magnitude greater than that of adversarial samples.

4 CuRTAIL Methodology

Fig. 4 demonstrates the high-level block diagram of MRR methodology. In our proposed countermeasure, a number of modular redundancies (checkpoints) are trained to characterize the data density distribution in the space spanned by the victim model. The defender modules are then used in parallel to checkpoint the reliability of the ultimate prediction and raise an alarm flag for risky samples. We refer to MRR modules that checkpoint the intermediate DL layers as "*latent defenders*". Whereas, the redundancy modules operating on the input space are referred to as the "*input defenders*". We use the term *checkpoints* and *modular redundancies* interchangeably throughout the paper.

4.1 Latent Defenders

The goal of each intermediate defender (checkpointing) module is to learn the PDF of the explored sub-spaces in a particular DL feature map. The learned density function is then used to identify the rarely observed regions. We consider a Gaussian Mixture Model (GMM) as the prior probability to characterize the data distribution at each checkpoint location. We emphasize that our proposed approach is rather

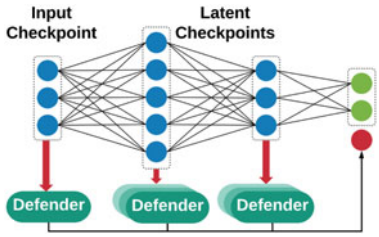

Fig. 4. High-level block diagram of MRR methodology. Multiple defenders checkpoint the input and intermediate activation maps in parallel. The output of the victim neural network (green neurons) is augmented with a confidence measure (red neuron) determining the prediction legitimacy.

generic and is not restricted to the GMM distribution. The GMM distribution can be replaced with any other prior depending on the application data.

4.1.1 Training a Single Latent Defender

To effectively characterize the explored sub-space as a GMM distribution, one is required to minimize the entanglement between pairs of Gaussian distribution (corresponding to every two different classes) while decreasing the inner-class diversity. Fig. 5 illustrates the high-level block diagram of the training procedure for devising a parallel checkpointing module. Training a defender module is a one-time offline process and is performed in three steps:

❶ Replicating the victim neural network and all its feature maps. An L_2 normalization layer is inserted in the desired checkpoint location. The normalization layer maps the latent feature variables, $f(x)$, into the euclidean space such that the acquired data embeddings live in a d-dimensional hyper-sphere, i.e., $\|f(x)\|_2 = 1$. This normalization is crucial as it partially removes the effect of over-fitting to particular data samples that are highly correlated with the underlying DL parameters. The L_2 norm is selected to be consistent with our assumption of GMM prior distribution. This norm can be easily replaced by an arbitrarily user-defined norm through our accompanying API.

❷ Fine-tuning the replicated neural network to enforce disentanglement of data features (at a particular checkpoint location) to characterize the PDF of occupied (explored) subspaces. To do so, we optimize the defender module by adding the following loss function to the conventional cross entropy loss

$$\gamma \; [\; \underbrace{\|C^{y^*} - f(x)\|^2}_{loss_1} \; - \; \underbrace{\Sigma_{i \neq y^*} \|C^i - f(x)\|^2}_{loss_2} \; + \; \underbrace{\Sigma_i (\|C^i\| - 1)^2}_{loss_3} \;]. \tag{3}$$

Here, γ is a trade-off parameter that specifies the contribution of the additive loss term, $f(x)$ is the corresponding feature vector of input sample x at the checkpoint location, y^* is the ground-truth label, and C^i denotes the center of all data abstractions ($f(x)$) corresponding to class i. The center values C^i and intermediate feature vectors $f(x)$ are trainable variables that are learned by fine-tuning the defender module.

Fig. 6 illustrates the optimization goal of each defender module per Eq. (3). The first term ($loss_1$) in Eq. (3) aims to condense latent data features $f(x)$ that belong to the same class. Reducing the inner-class diversity, in turn, yields a sharper Gaussian distribution per class. The second term

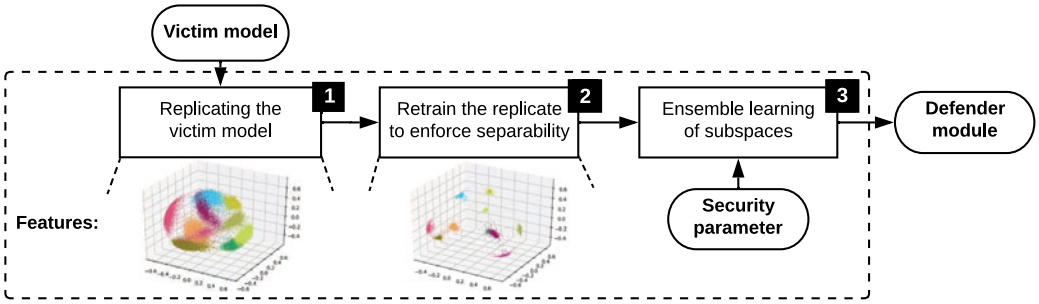

Fig. 5. Block diagram of the training procedure for devising parallel redundancy modules. Each latent defender is built by minimizing the entanglement of intermediate data features in a Euclidean space at a particular checkpoint location. This goal is achieved through several rounds of iterative realignment of data abstractions. The latent data space is then characterized as an ensemble of lower dimensional sub-spaces to effectively learn the PDF of explored regions and detect atypical samples based on a user-defined security parameter.

($loss_2$) intends to increase the intra-class distance between different categories and promote separability. The composition of the first two terms in Eq. (3) can be arbitrarily small by pushing the centers to ($C^i \leftarrow \pm\infty$). We add the term, $loss_3$, to ensure that the underlying centers lie on a unit d-dimensional hyper-sphere and avoid divergence in training the latent defender modules.

Figs. 7a and 7b demonstrate the distance of legitimate (blue) and adversarial (red) samples from the corresponding centers C^i in a checkpoint module before and after retraining. The centers C^i before fine-tuning the checkpoint (defender) module are equivalent to the mean of the data points in each class. As shown, fine-tuning the defender module with proposed objective function can effectively separate the distribution of legitimate samples from malicious data points. Note that training the latent defender modules is carried out in an unsupervised setting, meaning that no adversarial sample is included in the training phase.

❸ High dimensional real-world datasets can be represented as an ensemble of lower dimensional sub-spaces [17], [18]. As discussed in [17], under a GMM distribution assumption, data points belonging to each class can be characterized as a spherical density in two sub-spaces: (i) The sub-space where the data actually lives and (ii) its orthogonal complementary space. We use High Dimensional Discriminant Analysis (HDDA) [17] to learn the mean and conditional covariance of each class as a composition of lower dimensional sub-spaces.

The learned PDF variables (i.e., mean and conditional covariance) are used to compute the probability of a feature point $f(x)$ coming from a specific class. In particular, for each incoming test sample x, the probability $p(f(x)|y^i)$ is evaluated where y^i is the predicted class (output of the victim neural network) and $f(x)$ is the corresponding data abstraction at the checkpoint location. The acquired likelihood is then compared against a user-defined *cut-off threshold* which we refer to as the *security parameter*. The Security Parameter (SP) is a constant number in the range of $[0\% - 100\%]$ that determines the hardness of defender modules. Fig. 8 illustrates how the SP can control the hardness of the pertinent decision boundaries. In this example, we have depicted the latent features of one category that are projected into the first two Principal Component Analysis (PCA) components in the euclidean space (each point corresponds to a single input image). The blue and black contours correspond to security parameters of 10 and 20 percent, respectively. For example, 10 percent of the legitimate training samples lie outside the contour specified with $SP = 10\%$.

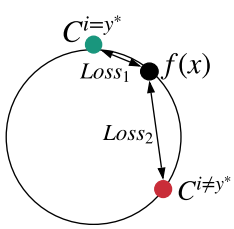

Fig. 6. Defender module optimization objective.

An active adversary can find a structured noise that moves the data point from one cluster to the center of the other clusters; thus fooling the defender modules (Fig. 11a). The risk of such attack approach is significantly reduced in our proposed MRR countermeasure due to three main reasons: (i) Increasing intra-class distances in each checkpointing module; The latent defender modules are trained such that not only the inner-class diversity is decreased, but also the distance between each pair of different classes is increased (see Eq. (3)). (ii) Use of parallel checkpointing modules as explained in Section 4.1.2; the attacker requires to simultaneously deceive all the defender models in order to succeed. (iii) Learning a separate defender module in the input space to validate the Peak Signal-to-Noise Ratio (PSNR) level of the incoming samples as discussed in Section 4.2.

4.1.2 Training Multiple Latent Defenders

In this section, we explain our methodology for creating multiple defender modules that are negatively correlated.

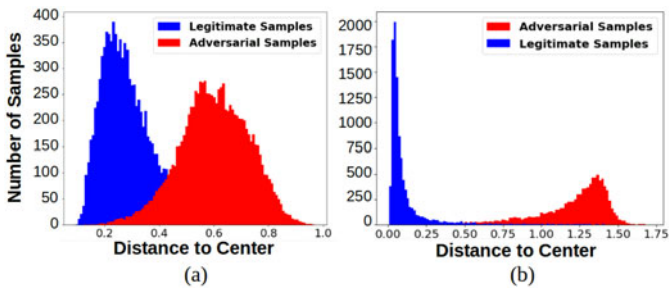

Fig. 7. (a) Distance of legitimate (blue) and adversarial (red) samples from the corresponding centers C^i before and (b) after realignment of data samples. In this example, we consider the *LeNet* model [16] trained on MNIST. The checkpoint is inserted in the second-to-last layer and adversarial samples are generated by FGS attack.

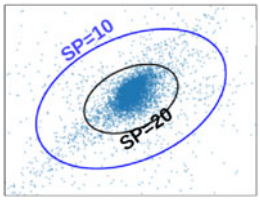

Fig. 8. Illustration of the effect of security parameter on the detection policy. A high SP leads to a tight boundary which treats most samples as adversarial examples.

Specifically, two MRRs are negatively correlated if deceiving one of them raises a high suspicion in the other one and vice versa. Consider the ith MRR that maps a legitimate input x to feature vector $f_i(x)$, where $f_i(x)$ is close (in terms of euclidean distance) to the ground-truth cluster center $C_i^{y^*}$. An adversary trying to mislead this defender would generate a perturbed input $x+\eta$ such that $f_i(x+\eta)$ is far from $C_i^{y^*}$. Negative correlation means that for the subsequent MRR with feature vector $f_{i+1}(x)$, adding the perturbation η will bring $f_{i+1}(x+\eta)$ closer to its ground-truth cluster center $C_{i+1}^{y^*}$. Fig. 9 shows this effect where the colored cloud represents the data points in each MRRs latent feature map and the decision boundary specified by the security parameter (SP) is shown with the oval contour.

To mitigate such adaptive attacks, we propose to train a Markov chain of detector modules as illustrated in Fig. 10. To build this chain of MRRs, we first train a single defender module as described in Section 4.1. Next, we generate a new set of training data from samples that are deemed adversarial for the previous defender module. In particular, the nth defender of this chain takes an input data x, generates a perturbation η, and feeds $clip(x+\eta)$ to the $(n+1)$th defender. The perturbation η is chosen as $\eta = \frac{\partial L1}{\partial x}$, where $L1$ is the $loss_1$ term in Eq. (3) corresponding to the nth defender. Given these new perturbed samples, data points that deviate from the centers in the nth defender will be close to the corresponding center in the $(n+1)$th defender. As such, deceiving all the defenders requires a larger perturbation.

4.2 Input Defender

We leverage dictionary learning and sparse signal recovery techniques to measure the PSNR of each incoming sample and automatically filter out atypical samples in the input space. Fig. 11b illustrates the block diagram of an input defender module. An input checkpoint is configured in two main steps: (i) dictionary learning, and (ii) characterizing the typical PSNR per class after sparse recovery.

❶ Dictionary learning; we learn a separate dictionary for each class of data by solving

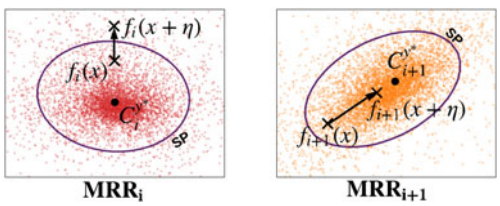

Fig. 9. Enforcing negative correlation between MRRs.

Fig. 10. Training multiple negatively correlated defenders at each checkpoint layer of the DL model.

$$\underset{D^i}{argmin}\ \frac{1}{2}\|Z^i - D^i V^i\|^2 + \beta\|V^i\|_1 \quad s.t. \quad \|D_k^i\| = 1, \quad 0 \le k \le K_{max}. \tag{4}$$

Here, Z^i is a matrix whose columns are pixels extracted from different regions of input images belonging to category i. For instance, if we consider 8×8 patches of pixels, each column of Z^i would be a vector of 64 elements. The goal of dictionary learning is to find matrix D^i that best represents the distribution of pixel patches from images belonging to class i. We denote the number of columns in D^i by k_{max}. For a certain D^i, the image patches Z^i are represented with a sparse matrix V^i, and $D^i V^i$ is the reconstructed patches. We leverage Least Angle Regression (LAR) method [19] to solve the Lasso problem defined in Eq. (4).

During the execution phase, the input defender takes the output of the victim DL model (e.g., predicted class i) and uses Orthogonal Matching Pursuit (OMP) routine [20] to sparsely reconstruct the input data with the corresponding dictionary D^i. The reconstructed image is formed by denoising all of the non-overlapping patches within the image by the corresponding class dictionary. Algorithm 1 outlines the pseudo code of the OMP routine. As shown, Performing OMP requires iterative execution of three main steps: (i) finding the best matching sample in the dictionary matrix D (Line 4 of Algorithm 1), (ii) least-square (LS) optimization (Line 5 of Algorithm 1), and (iii) residual update (Line 6 of Algorithm 1). In the provided pseudo code D_{col} represents the col^{th} column of the dictionary matrix D, and D_Λ is the subset of D columns that have been chosen so far in the routine. The OMP algorithm terminates when the number of non-zero elements in the output coefficient vector (V^*) is more than the sparsity level k.

❷ Characterizing typical PSNR in each category as defined in Eq. (5); a benign sample belonging to class i should be well-reconstructed as $D^i V^*$ with a high PSNR value, where V^* is the optimal solution obtained by the OMP routine. We profile the PSNR percentile of legitimate samples within each class and find the corresponding threshold that satisfies the user-defined security parameter. If an incoming sample has a PSNR lower than the threshold

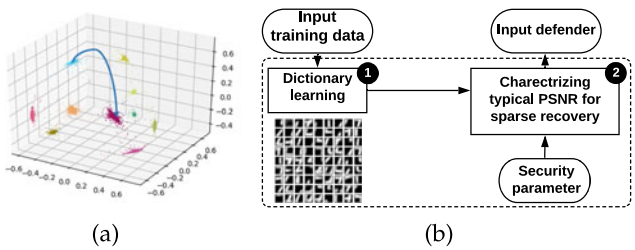

Fig. 11. An input defender module is devised based on robust dictionary learning techniques to automatically filter out test samples that highly deviate from the typical PSNR of data points within the corresponding predicted class.

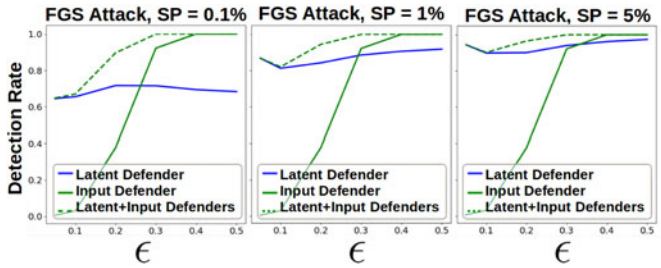

Fig. 12. Adversarial detection rate of input and latent defenders as a function of the perturbation level for various SP. Here, FGS is used to generate adversarial samples and the perturbation is adjusted by changing attack parameter ϵ.

(i.e., high perturbation after reconstruction by the corresponding dictionary), it will be regarded as a malicious data point

$$PSNR = 20\ log_{10}(MAX_I) - 10\ log_{10}(MSE), \tag{5}$$

where the mean square error (MSE) is defined as the L_2 difference of the input image and the reconstructed image based on the corresponding dictionary, i.e., $\|r^k\|$. The MAX_I is the maximum possible pixel value of the image (e.g., 255). Note that since the decision of the defender solely depends on the norm of the residual vector, $\|r^k\|$, there is no need to explicitly compute the sparse vector V^*.

Algorithm 1. OMP Algorithm

Inputs: Dictionary D, input sample V, maximum sparsity level k.
Output: Coefficient vector V^*.
1: $r^0 \leftarrow X$
2: $\Lambda^0 \leftarrow \emptyset$
3: **for** $i = 1,...,k$ **do**
4: $\quad \Lambda^i \leftarrow \Lambda^{i-1} \cup argmax_{col} | < r^{i-1}, D_{col} > |$
5: $\quad V^i \leftarrow argmin_V \|r^{i-1} - D_{\Lambda^i} V\|$
6: $\quad r^i \leftarrow r^{i-1} - D_{\Lambda^i} V^i$

Fig. 12 demonstrates the impact of perturbation level on the pertinent adversarial detection rate for three different security parameters (cut-off thresholds). In this experiment, we have considered the FGS attack with different ϵ values on *LeNet* MNIST benchmark. Table 2 summarizes the DL model topology used in each benchmark. The latent defender module (checkpoint) is inserted at the second-to-last layers. As shown, the use of input dictionaries facilitate detection of adversarial samples with relatively high perturbation (e.g., $\epsilon > 0.25$) while the latent defender effectively distinguishes malicious samples with very small perturbations. We extensively evaluate the impact of security parameter on system performance for various benchmarks in Section 6.

4.3 Model Fusion

Fig. 13 depicts the configuration of the defender modules in the execution phase. Each defender module gets the input sample x and generates a binary output $d_k \in \{0, 1\}$ with value 1 denoting an adversarial sample. Each defender makes the binary decision based on the security parameter (SP) discussed in Section 4.1. To aggregate the binary random variables $\{d_1, \ldots, d_N\}$ into a single decision a, we compute the probability of the input being adversarial as follows:

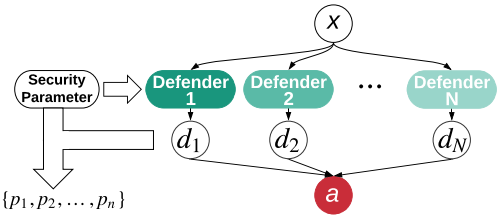

Fig. 13. CuRTAIL uses a score-based statistical method to aggregate the decision of all MRRs.

$$P(a = 1|\{d_1, d_2, \ldots, d_n\}) = 1 - \prod_{n=1}^{N} (1 - P_n)^{d_n}, \tag{6}$$
$$P_n = P(a = 1|d_n = 1).$$

This formulation resembles the well-known noisy-OR terminology used in statistical learning [21]. In MRR methodology, each defender has a parameter P_n which indicates the likelihood of a sample being adversarial given that the nth defender has labeled it as a malicious sample. If all detectors have a parameter of $P_n = 1$, then the formulation in Eq. (6) is equivalent to the logical OR between $\{d_1, \ldots, d_N\}$.

Inserting P_n into the noisy-OR operation enables learning different weights for different defender modules to improve adversarial detection rate without increasing false alarms. In practice, the P_n parameters are estimated by evaluating the performance of each individual defender. For this purpose, we use a subset of the training data and create adversarial samples. In particular, for each legitimate sample x, we generate $x^a = x + \epsilon \cdot \nabla_x(\mathcal{L})$ where $\mathcal{L}$ is the victim model's cross-entropy loss. We next use the generated samples to obtain P_n. By using this generic form of adversary, we ensure that the calculated P_n is attack-agnostic and works well with different adversaries. The probability P_n is estimated as

$$P_n = \frac{S_{True}}{S_{False} + S_{True}}, \tag{7}$$

where S_{True} is the number of adversarial samples that are correctly detected by defender n and S_{False} denotes the number of legitimate samples that were mistaken for adversaries.In our experiments, we raise alarm flags for samples with $P(a = 1|\{d_1, d_2, \ldots, d_n\}) \geq 0.5$.

4.4 Sensitivity Analysis

The effectiveness of adversarial perturbations on DNN classification can be quantified by their induced variations on intermediate activations. To study this effect, we extract the cluster centers corresponding to each hidden layer using a subset of the (benign) training data. Let X^{y^*} denote the set of samples with label y^*. The corresponding cluster center $C_l^{y^*}$ at layer l is calculated as

$$C_l^{y^*} = \mathbb{E}_{x \sim X^{y^*}}[P_l \cdot f_l(x)], \tag{8}$$

where $f_l(x)$ is the layer activations. To reduce dimensionality, we perform PCA (P_l operator in Eq. (8)) on the activation vectors such that more than 99 percent of the energy is preserved.

The perturbation signal in adversarial samples can be modeled as an additive noise to the input data. At each

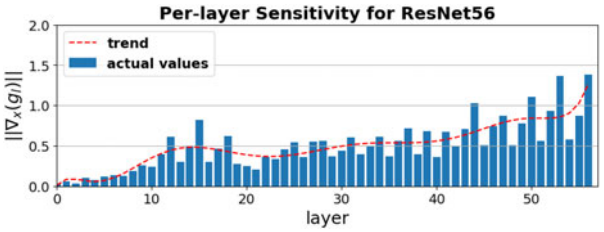

Fig. 14. Per-layer sensitivity analysis for ResNet56.

intermediate layer l, the added perturbation increases the distance between the activation vectors and their ground-truth cluster centers. Given the pre-computed center $C_l^{y^*}$ and PCA matrix P_l, the distance is measured as

$$g_l(x) = \|P_l \cdot f_l(x) - C^{y^*}\|^2. \tag{9}$$

For each layer l we define the instability as

$$\underset{r \neq 0}{Sup} \frac{g_l(x+r) - g_l(x)}{\|r\|}, \tag{10}$$

where Sup denotes supremum and r is the input noise.

For small perturbations, Taylor series can be leveraged to closely approximate the supremum by

$$\underset{r \neq 0}{Sup} \frac{< r, \nabla_x(g_l) >}{\|r\|}. \tag{11}$$

The upper bound in Eq. (11) is achieved if and only if the perturbation vector r is aligned with the gradient

$$r = \|r\| \frac{\nabla_x(g_l)}{\|\nabla_x(g_l)\|}. \tag{12}$$

Substituting this value in Eq. (11), suggests that the instability of DNN layers is bounded by the magnitude of the gradient $\|\nabla_x(g_l)\|$. This measure allows for identification of most sensitive intermediate layers; layers with larger $\|\nabla_x(g_l)\|$ are better suited for MRR placement. We observed that the last layer shows highest sensitivity towards input perturbations. Fig. 14 shows an example analysis for ResNet56 trained on CIFAR-100. We thus place all latent defenders at the output of the second-to-last layer in our experiments.

5 CuRTAIL Hardware Implementation

Motivation. There is an inherent trade-off between the computational complexity (e.g., runtime overhead) of the modular redundancies and the reliability of the system. On the one hand, a high number of validation checkpoints increases system reliability, but it also increases the computational load. On the other hand, a small number of checkpoints degrades the defense mechanism performance by treating adversarial samples as legitimate ones. Let us consider a naïve implementation of MRRs on commodity hardware where the checkpoints are executed*sequentially*.

Fig. 15 demonstrates the pertinent utility and reliability trade-off under such settings for LeNet model on MNIST dataset. Here, runtime is normalized to the cost of one forward propagation in the target neural network. As seen, the runtime in this setting increases linearly with the number of

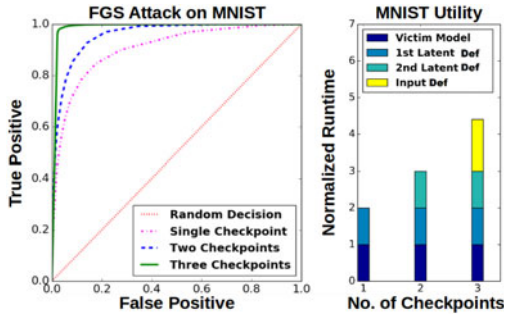

Fig. 15. Complexity and reliability trade-off for the LeNet model on MNIST dataset performed on an NVIDIA Geforce 980 GPU hosted by an Intel Core-i7 CPU.

checkpoints, which is not desirable. To address this, we design an FPGA-based accelerator for optimized parallel execution of CuRTAIL MRRs. In the following, we elaborate on various components of the CuRTAIL accelerator.

5.1 CuRTAIL Hardware Acceleration

CuRTAIL hardware acceleration stack enables just-in-time online detection of adversarial samples. Once the MRRs are trained, CuRTAIL automatically generates the hardware implementation for the modules by performing two main phases as illustrated in Fig. 16: (i) offline pre-processing phase to obtain the MRR configurations, and (ii) online execution phase in which the legitimacy of each incoming input data is validated on the fly.

Pre-Processing Phase. This phase consists of one main task, i.e., resource profiling and design customization. There is a trade-off between the computational complexity (e.g., runtime overhead) of the modular redundancies and the overall system reliability in terms of successful adversarial detection rate. CuRTAIL uses physical profiling to estimate resource utilization for the victim model as well as the defender modules. The output of physical profiling along with a set of user-defined constraints (e.g., real-time requirements) is then fed into the design customization unit to determine the viable number of checkpoints (defenders) as will be discussed in Section 5.2. The customization unit analyzes this trade-off between model reliability (robustness), resource limitation, and throughput to decide the best number of defenders suitable to the task and target hardware. This stage is performed only once and incurs a negligible overhead.

Execution Phase. Once the redundancy modules are customized per hardware and user-defined physical constraints,

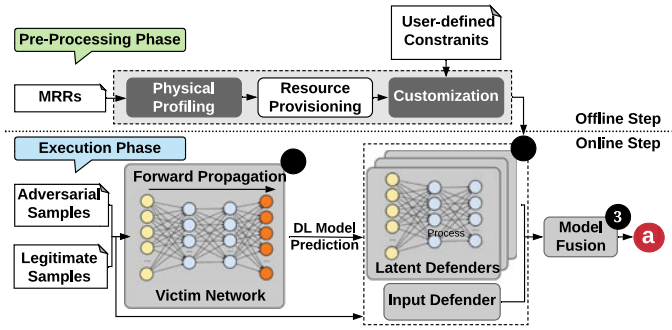

Fig. 16. High-level flow of CuRTAIL hardware acceleration stack. Based on the user-provided constraints, CuRTAIL outputs the best defense layout that ensures maximum robustness and throughput.

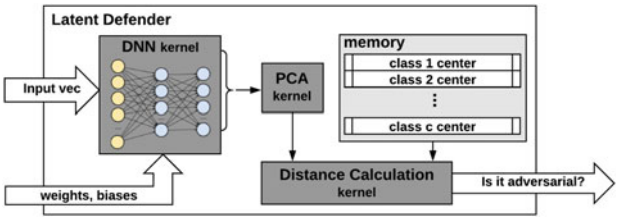

Fig. 17. Latent defender structure: the pertinent activations are acquired by propagating the input sample through the defender. PCA is then applied to reduce the dimensionality of the obtained activation. The L_2 distance with the corresponding GMM center determines the legitimacy of the input.

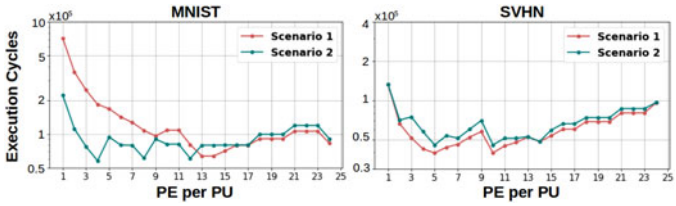

Fig. 18. Design space exploration for MNIST and SVHN benchmarks on *Xilinx Zynq-ZC702* FPGA. CuRTAIL finds the optimal configuration of PEs and PUs to best fit the DL architecture and the available hardware resources.

the DL model is ready to be deployed for online execution. CuRTAIL performs three tasks in the execution phase.

1. *Forward Propagation.* The predicted class for each incoming sample is acquired through forward propagation in the victim DL model. The predicted output is then fed to defenders for validation.
2. *Validation.* CuRTAIL executes the learned MRRs (Section 4) on FPGA to validate the legitimacy of the input data and its associated label. In particular, samples that do not lie in the*user-defined* probability interval, i.e., SP, are discarded.
3. *Model Fusion.* The output of redundancy modules are finally aggregated to determine the legitimacy of the input data and its associated inference label (Section 4.3).

In the following, we first discuss the hardware architecture of latent and input defenders that enables high throughput and low energy realization of the recurring execution phase. We then discuss resource profiling, automated design customization, and the scalability of CuRTAIL.

5.1.1 Latent Defenders

During the execution phase, each incoming sample is passed through the latent defender modules that are trained offline (Section 4.1). The legitimacy of each sample is then determined by measuring the L_2 distance with the corresponding GMM center. The latent defenders can be situated in any layer of the victim network, therefore, the extracted feature vector from the DL model can be of high cardinality. High dimensionality of the GMM centers may cause shortage of memory as well as increasing the computational cost and system latency. To mitigate the curse of dimensionality, we perform Principal Component Analysis (PCA) on the outputs of the latent defenders before measuring the L_2 distance. For our latent defenders, PCA is performed such that more than 99 percent of the energy is preserved. Fig. 17 illustrates the high-level schematic of a latent defender kernel.

The most computationally-intensive operation in DL model execution is matrix-matrix multiplication. Recent FPGAs provide hardeneds of DSP units together with the re-configurable logic to offer a high computation capacity. The basic function of a DSP unit is a multiplication and accumulation (MAC). In order to optimize the design and make use of the efficient DSP slices, we took a parallelized approach to convert the DL layer computations into multiple operations running simultaneously as suggested in [22]. In this setting, DNN layer computations are performed within several parallel-working processing units (PUs), each of which comprises a number of parallel processing elements (PEs). The parallelism can be controlled by parameters N_{PE} and N_{PU} which are static across all layers of the DL model. In order to achieve maximum throughput, it is essential to fine-tune the parallelism parameters.

There is a trade-off between the number of parallel employed Processing Units (N_{PU}) and hardware complexity in terms of memory access. An increase in the number of parallel computation units will not always result in better throughput since the dimensionality of the data and divisibility into ready-to-process batches highly affects the efficiency of these parallel units. There are two implementation scenarios in CuRTAIL; A Processing Unit (PU) can either be assigned a subset of the layer output features (scenario 1) or the whole feature map for computation (scenario 2). In the first scenario, multiple PUs work in parallel to gradually compute all output features in each DL layer while in the second scenario, batches of input samples can be processed simultaneously where the batch size is equal to the number of PUs. CuRTAIL switches between these two scenarios based on model architecture, layer dimensionality, and/or available hardware execution resources. Fig. 18 shows an example of the design space exploration for the MNIST and SVHN benchmarks.[3] Note that based on resource constraints, N_{PU} is uniquely determined by the number of Processing Engines (PE) per PU (the horizontal axis in Fig. 18).

To minimize the latency of latent defenders, we infuse the PCA kernel inside the defender modules. Collectively, all transformations from the original input space to the space spanned by principal components can be shown as a vector-matrix multiplication $t_l = P_l \cdot f_l(x)$ where P_l is a matrix whose rows are eigenvectors obtained from the legitimate data $f_l(x)$. The PCA kernel can thus be replaced with a *Dense* layer, appended to the defender's DL architecture. Note that the extraction of P_l from $f_l(x)$ is a one-time offline process.

5.1.2 Input Defender

The input defender module relies on sparse signal reconstruction to detect anomalies in the victim model's input space. Execution of OMP is the main computational bottleneck in the input defender. We provide a scalable implementation of the OMP routine on FPGA to enable low-energy and in-time analysis of input data. By modifying the OMP algorithm such that it maximally utilizes the available on-chip resources, we boost the performance of our OMP core

3. See Section 6 for details of each benchmark.

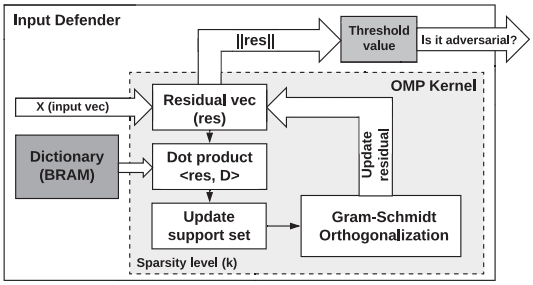

Fig. 19. Input defender structure: the OMP core iteratively reconstructs input vectors using a previously learned dictionary. The reconstruction error is used to determine input legitimacy.

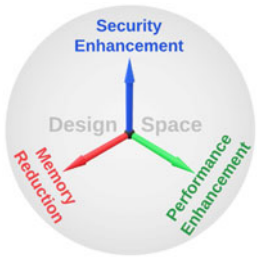

Fig. 21. CuRTAIL provides customized defense by balancing the design-space trade-offs. The goal of CuRTAIL is to maximize model robustness while adhering to the underlying memory and runtime constraints.

for large chunks of data. Fig. 19 shows the high-level schematic of an input defender's kernel. Here, the *support set* contains columns of the dictionary matrix that are chosen so far in the routine.

OMP execution includes two computationally expensive steps, namely the matrix-vector multiplication and the LS optimization. Each of these steps includes multiple dot product computations. The sequential nature of the dot product, renders the usage of pipelining inefficient. Thereby, we use a *tree-based* reduction technique to find the final value by adding up the partial results produced by each of the parallel processes. Fig. 20 outlines the realization of a tree-based reduction module. The reduction module takes an array of size $2M$ as its input (array a) and oscillates between two different modes. In mode 0, the function reduces a by using $temp$ as a temporary array. In mode 1, $temp$ is reduced using a. This interleaving between the two arrays ensures maximal utilization of memory blocks. The final result is returned based on the final mode of the system.

Algorithm 2. Incremental QR Decomposition With Modified Gram-Schmidt

Inputs: New column D_{Λ^n}, Q^{n-1}, R^{n-1}.
Output: Q^n, R^n.
1: $R^n \leftarrow \begin{bmatrix} R^{n-1} & 0 \\ 0 & 0 \end{bmatrix}$, $\epsilon^n \leftarrow D_{\Lambda^n}$
2: **for** $j = 1,...,n\text{-}1$ **do**
3: $\quad R^n_{jn} \leftarrow (Q^{n-1})^T_j \epsilon^n$
4: $\quad \epsilon^n \leftarrow \epsilon^n - R^n_{jn} Q^{n-1}_j$
5: $R^n_{nn} = \|\epsilon^n\|$
6: $Q^n = Q^{n-1} \epsilon^n / R^n_{nn}$

The LS optimization step is performed using QR decomposition to reduce implementation complexity and make it well-suited for hardware accelerators. The Gram-Schmidt orthogonalization technique gradually forms the orthogonal matrix Q and upper-triangular matrix R to iteratively calculate the decomposition. Algorithm 2 outlines the modified Gram-Schmidt incremental orthogonalization method [23].

Using the Gram-Schmidt methodology, the residual update can be considerably simplified by replacing Line 6 of Algorithm 1 with Eq. (13):

$$r^i \leftarrow r^{i-1} - Q_i (Q_i)^T r^{i-1}. \tag{13}$$

The updated residual vector r at the end of each iteration is made orthogonal to the selected dictionary samples. As such, none of the columns of matrix D would be selected twice during one call of the OMP algorithm. Based on this observation, we reuse the same set of block memories initially assigned to the dictionary matrix D to store the newly computed columns of the Q matrix, per iteration [24].

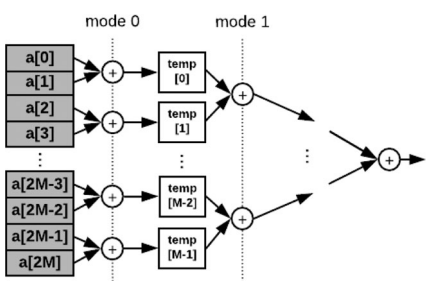

Fig. 20. Tree-based vector reduction algorithm.

5.2 Automated Design Customization

CuRTAIL provides an automated customization unit that maximizes DL model robustness while adhering to the limitations dedicated by the underlying hardware platform and/or application. These limitations include the available memory, computing resources, and system throughput. Our automated optimization ensures ease of use and reduces the non-recurring engineering cost. Fig. 21 depicts the trade-offs optimized by CuRTAIL customization unit. This unit takes as input the high-level description of the defenders in Caffe together with the application-specific runtime constraint and available hardware resources, e.g., storage and computational cores. It then outputs the best combination of defender modules to ensure maximum robustness against adversarial attacks while adhering to the available resources.

To characterize the design trade-offs, we thoroughly examine the performance and resource utilization for different building blocks of a DL model. For FPGA platforms, the main resource bottlenecks for DNN implementation are the Block RAM (BRAM) capacity and the number of DSP units. The dictionary matrices used in the input defender as well as the latent defender weights and biases are stored in the DRAM memory to be accessed during the execution phase. Upon computation, data is moved from the DRAM to BRAM which enables faster computation.

CuRTAIL sets the configuration of the defenders with regard to these two constraints (number of DSP units and the available BRAM). In particular, CuRTAIL solves the following optimization to find the best configuration for the number of defenders N_{def} and the number of processing units N_{PU} per defender

TABLE 1
Runtime and Computational Complexity of Each Custom Layer in CuRTAIL Framework

		Runtime	Computational Complexity
Input Defender	OMP Kernel	$\beta \times patch_{len}(kD_{size} + k^2)$	$\mathcal{O}(patch_{len} D_{size}^2)$
Latent Defender	Conv Layer	$\beta\lceil \frac{W_{in}}{N_{PE}} \rceil \times H_{in} \times f_{in} \times \lceil \frac{f_{out}}{N_{PU}} \rceil \times kernel_{size}^2$	$\mathcal{O}(W_{out} H_{out} f_{in} f_{out} kernel_{size}^2)$
	Dense Layer	$\beta\lceil \frac{N_{in} \times N_{out}}{N_{PE} \times N_{PU}} \rceil$	$\mathcal{O}(N_{in} N_{out})$
	PCA Layer	$\beta\lceil \frac{P \times L}{N_{PE} \times N_{PU}} \rceil$	$\mathcal{O}(PL)$

$$\begin{aligned} &\underset{N_{PU}, N_{def}}{Maximize}\ (DL_{robustness}) \qquad s.t.: \\ &T_{def}^{max} \leq T_u, \\ &N_{def} \times N_{PU} \times DSP_{PU} \leq R_u, \\ &N_{PU} \times [max(size(W^i)) + max(|X^i| + |X^{i+1}|)] \leq M_u, \end{aligned} \tag{14}$$

where T_u, M_u, and R_u are user-defined constraints for system latency, BRAM budget, and available DSP resources, respectively. Here, $size(W^i)$ denotes the total number of parameters and $|X^i|$ is the cardinality of the input activation in layer i. DSP_{PU} indicates the number of DSP slices used in one processing unit. Variable T_{def}^{max} is the maximum required latency for executing the defender modules. CuRTAIL considers both sequential and parallel execution of defenders based on the available resource provisioning and size of the victim model. Once the optimization is solved for N_{PU}, N_{PE} is uniquely determined based on available resources.

The OMP unit in CuRTAIL incurs a fixed memory footprint and latency for a given application. As such, the optimization of Eq. (14) does not include this constant overhead. Instead, we exclude this overhead from the user-defined constraints and use the updated upper bounds. The memory requirement for an OMP kernel is equivalent to $(patch_{len} \times (D_{size} + 1) + D_{size}^2) \times 4$ bytes, where $patch_{len}$ is the total number of pixels within a patch of an input sample (usually set to 64), and D_{size} is the number of columns in the dictionary matrix. The term $patch_{len} \times (D_{size} + 1)$ corresponds to the storage space required for the dictionary matrix as well as the input vector of a data patch. D_{size}^2 stands for the memory space required to store the R matrix while performing Algorithm 2. Note that the Q matrix reuses the space originally dedicated to the dictionary to eliminate unnecessary use of memory resources. The required OMP computational time per input patch is superposed by the computational latency of the latent defenders that are ran in parallel with the input defender.

Our customization unit is designed such that it maximizes the resource utilization to ensure maximum throughput. CuRTAIL performs an exhaustive search over the parameter N_{PU} and solves the equations in (14) using the Karush-Kuhn-Tucker (KKT) method to calculate N_{def}. The calculated parameters capture the best trade-off between model robustness and throughput. Our optimization outputs the most efficient layout of defender modules as well as the sequential or parallel realization of defenders. This constraint-driven optimization is non-recurring and incurs negligible overhead ($10 - 100$ $msec$ depending on the hardware platform.).

5.3 Computational Analysis and Scalability

Table 1 summarizes the computational complexity as well as the corresponding number of clock-cycles required for execution of each custom layer in CuRTAIL. In all entries from the third column of the table, β denotes a system-dependant constant that characterizes the runtime requirement per unit of floating point operation. For an OMP kernel (employed in the input defender), $patch_{len}$ indicates the number of elements in an input data patch, D_{size} is the dictionary size, and k represents the sparsity level (usually set to 5). Runtime of the *Convolution* layer is dependent on the input dimensionality ($W_{in} \times H_{in}$), convolution kernel size ($kernel_{size}$), number of input and output filter channels (f_{in}, f_{out}), and the values of N_{PE} and N_{PU} acquired from solving Eq. (14). Same pattern holds for *Dense* layers where the input and output dimensions are denoted by N_{in} and N_{out}. PCA can be cast as a *Dense* layer as discussed in Section 5.1.1 with P and L representing the input and output dimensionalities, respectively.

6 Evaluations

We evaluate CuRTAIL on five machine learning datasets: MNIST, SVHN, CIFAR-10, CIFAR-100, and ImageNet.

MNIST Benchmark. The MNIST data is a 28×28 grayscale images of handwritten digits with 60,000 train images and 10,000 test samples. The images are normalized such that each pixel takes a real value in the range of [0,1]. For this dataset, we train and use the DL topology proposed in [26] which is also available in Table 2.

SVHN Benchmark. This dataset consists of 32×32 real-world color images of house numbers in Google Street View images. We split the data into $\sim 60,500$ train images and 26,000 test samples. The image pixels are normalized to the [0,1] range. Table 2 encloses the DL architecture used for this benchmark in our experiments.

CIFAR Benchmarks. We carry out our experiments on the two available CIFAR [27] datasets. CIFAR-10 and CIFAR-100 benchmarks consist of colored (RGB) images with dimensionality 32×32 that are categorized in 10 and 100 classes, respectively. We split the data samples into a set of 50,000 training data and a set of 10,000 test data. The images are normalized using per-channel mean and standard deviation such that each pixel takes a value in the $[0 - 1]$ range. In our experiments, we train and use the state-of-the-art DL topology proposed in [26] for CIFAR-10 and ResNet56-v2 [25] for CIFAR-100, as enclosed in Table 2.

ImageNet Benchmark. ImageNet is a large database consisting of over 15 million data samples. Typically, a subset of images belonging to 1000 different categories is used by the research community for learning evaluation of ImageNet

TABLE 2
Benchmarked DL Models for Evaluating CuRTAIL Effectiveness

	Conv+BN+ReLU	MaxPool	Conv+BN+ReLU	MaxPool	Conv+BN+ReLU	Conv+BN+ReLU	Conv+BN+ReLU	MaxPool	Classifier
MNIST	$3 \xrightarrow[\text{stride 1}]{5\times5} 20$	2×2 stride 2	$20 \xrightarrow[\text{stride 1}]{5\times5} 50$	2×2 stride 2	-	-	-	-	500FC 10FC, softmax
SVHN	$3 \xrightarrow[\text{stride 1}]{5\times5} 20$	2×2 stride 2	$20 \xrightarrow[\text{stride 1}]{5\times5} 50$	2×2 stride 2	-	-	-	-	1000FC 500FC 10FC, softmax
CIFAR-10	$[3 \xrightarrow[\text{stride 1}]{3\times3} 96]\times3$	2×2 stride 2	$[96 \xrightarrow[\text{stride 1}]{3\times3} 192]\times3$	2×2 stride 2	$192 \xrightarrow[\text{stride 1}]{3\times3} 192$	$192 \xrightarrow[\text{stride 1}]{1\times1} 192$	$192 \xrightarrow[\text{stride 1}]{1\times1} 10$	8×8 average pool	10FC, softmax
CIFAR-100 (ResNet56-v2 [25])	$3 \xrightarrow[\text{stride 1}]{3\times3} 16$	-	$\begin{bmatrix}16 \xrightarrow{1\times1} 16\\ 16 \xrightarrow{3\times3} 16\\ 16 \xrightarrow{1\times1} 64\end{bmatrix}\times6$	-	$\begin{bmatrix}64 \xrightarrow{1\times1} 64\\ 64 \xrightarrow{3\times3} 64\\ 64 \xrightarrow{1\times1} 128\end{bmatrix}\times6$	$\begin{bmatrix}128 \xrightarrow{1\times1} 128\\ 128 \xrightarrow{3\times3} 128\\ 128 \xrightarrow{1\times1} 256\end{bmatrix}\times6$	-	8×8 average pool	100FC, softmax
ImageNet	$\begin{bmatrix}3 \xrightarrow[\text{stride 4}]{11\times11} 96\\ 96 \xrightarrow[\text{stride 1}]{5\times5} 256\end{bmatrix}$	3×3 stride 2	$256 \xrightarrow[\text{stride 1}]{3\times3} 128$	3×3 stride 2	$128 \xrightarrow[\text{stride 1}]{3\times3} 128$	$128 \xrightarrow[\text{stride 1}]{3\times3} 128$	-	3×3 stride 2	1024FC 1024FC 10FC, softmax

Conv layers are represented as $\langle input-channels\rangle \rightarrow stride]\langle kernel\ size\rangle\langle output-channels\rangle$ *and FC layers are denoted by* $\langle output-elements\rangle FC$.

data [28]. In our experiments, we train and use a DL architecture inspired by the well-known AlexNet [28] topology for ImageNet classification. Details about the trained model are available in Table 2. We down-sample ImageNet classes by a factor of 100 for execution efficiency purposes. Fig. 22 illustrates several samples from each of the selected classes.

6.1 Details of MRR Training

In the following, we enclose the details for training CuRTAIL defenders that are evaluated in the experiments. Note that the MRR training phase is a one-time process and its cost will be amortized among all future executions of CuRTAIL.

*Training Input Dictionaries.*We learn separate input dictionaries for each class in the benchmarked dataset. For each image in the dataset, we randomly sub-sample 30 small patches and create a new training set. Patch are set to 7×7 for MNIST and SVHN, 8×8 for CIFAR-10 and CIFAR-100, and 16×16 for ImageNet. We set the number of columns in each dictionary to 225. Dictionaries are learned following the description in Section 4.2. Once the dictionaries are learned, we execute OMP (Algorithm 1) to denoise input samples. The PSNRs are then computed as in Eq. (5), and compared against a cut-off threshold to raise alarms for high distortion values. We set the cut-off threshold value of input defender such that all the training data are considered legitimate samples.

*Training Latent Defenders.*For each application, we train a maximum of 16 latent defenders all of which checkpoint the second-to-last layer of the victim model. For ImageNet benchmark we only used 1 defender due to the high computational complexity of the pertinent neural network and attacks. We initialize the weights of latent defenders using those of the victim model, then retrain them by adding the extra term to the loss function with parameter γ set to 0.01 for all applications (see Eq. (3)). Each defender module is trained for the same number of epochs as the original training of the victim model with the same optimizer. The learning rate is set to $\frac{1}{10}$ of that of the victim model as the model is already in a relatively good local minimum.

Fig. 22. Example legitimate samples in ImageNet benchmark. Samples are randomly selected from the target classes.

6.2 Attack Analysis and Resiliency

We leverage a wide range of attack methodologies (namely, FGS [10], BIM [11], CarliniL2 [5], and Deepfool [4]) with varying parameters to ensure CuRTAIL's generalizability. The perturbation levels are selected such that the adversarial noise is undetectable by a human observer (Table 3 summarizes the pertinent attack parameters). We evaluate our defense mechanism in two attack scenarios:

- The attacker has complete access to the parameters and the architecture of the victim model but is not aware of the defense mechanism (a.k.a., *black-box attack*).
- The attacker has complete access to the parameters and the architecture of the victim model as well as the defender modules (a.k.a., *adaptive white-box attack*).

To characterize the performance of the proposed defense methodology against adversarial attacks, we evaluate CuRTAIL in terms of both the True Positive (TP) and False Positive (FP) detection rates. In this context, TP refers to the ratio of adversarial samples correctly detected by the system while FP denotes the ratio of legitimate samples that are mistakenly categorized as being malicious. There is an inherent trade-off between the TP and FP detection rates that can be controlled using the security parameter discussed in Section 4. The Area Under Curve (AUC) for a TP versus FP plot fully encapsulates this trade-off and can be used as a measure to quantify the quality of adversarial detection. A random decision has an AUC of 0.5 while an ideal detector will have an AUC of 1.

TABLE 3
Attack Parameters

Attack	Attack Parameters
FGS	$\epsilon \in \{0.01^a, 0.05^a, 0.1^{m,c}, 0.2^m\}$
Deepfool	$n_{iters} \in \{2^a, 5^a, 10^a, 20^a, 50^a, 100^a\}$
BIM	$\epsilon \in \{0.001^a, 0.002^a\}, n_{iters} \in \{5^a, 10^a, 20^a, 50^m, 100^m\}$
CarliniL2	$C \in \{0^a, 10^a, 20^{s,c}, 30^{s,c}, 40^{s,c}, 50^c, 60^c, 70^c\}$ LR = 0.1^a, steps = 10^a, iterations = 500^a

For CarliniL2 attack [5], "C" denotes the confidence, "LR" is the learning rate, "steps" is the number of binary search steps, and "iterations" stands for the maximum number of iterations. Superscripts ($m \rightarrow$ MNIST, $s \rightarrow$ SVHN, $c \rightarrow$ CIFAR, $a \rightarrow$ all*) are used to indicate the benchmarks for which the parameters are used.*

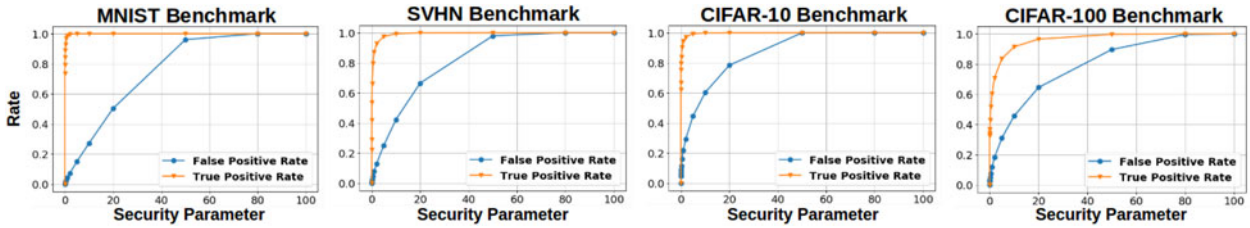

Fig. 23. CuRTAIL security parameter controls the TP and FP rates. The number of latent defenders in this experiment is 16.

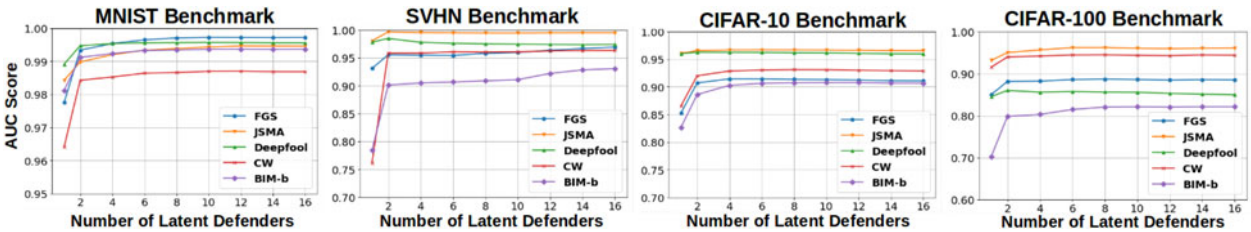

Fig. 24. The use of more modular redundancies improves the detection performance for all datasets.

6.3 Black-Box Attacks

In our first analysis, we study the relationship between detection success and the security parameter. We present the transition of the FP and TP rates with SP in Fig. 23. In this experiment, we use the test data to generate a dataset of adversarial samples using all attack algorithms/parameters of Table 3; we then evaluate the performance of our MRRs against these diverse adversarial samples to obtain the TP curve (shown in orange). The FP curve (shown in blue) is obtained by evaluating the defenders on the clean test data. An ideal defense would have FP≈ 0 and TP≈ 1.

In our next analysis, we evaluate the effect of the number of defenders on CuRTAIL detection. Fig. 24 shows the AUC obtained by CuRTAIL for different attack configurations where the adversary knows everything about the model but is not aware of the defenders. For a given number of defenders, the AUC for MNIST is relatively higher compared to more complex benchmarks (e.g., CIFAR-10). This is consistent with our hypothesis since the unexplored sub-space is larger in higher-dimensional benchmarks. Note that using more defenders eventually increases the AUC. We further summarize the performance of the CuRTAIL methodology against each of the FGS, JSMA, Deepfool, BIM, and Carlini&WagnerL2 attacks in Table 4. We used the open source library[4] provided by [29], for implementation of the attack algorithms. The JSMA attack was too slow on ImageNet, thus, we did not include the results in this study.

6.4 Adaptive White-Box Attack

To further corroborate the robustness of MRR methodology, we applied the state-of-the-art Carlini&WagnerL2 attack in a white-box setting. A similar strategy was used in [12] to break the state-of-the-art countermeasures including MagNet [26], APE-GAN [31], and other recently proposed efficient defences methods (e.g., [30]). The attacks in [12] are gray-box attacks, meaning that the attacker is aware of the defense mechanism but does not have access to its parameters. We perform a more powerful attack against our defense where the attacker also knows the parameter set of the defenders. Following the guidelines in [12], we modify the objective function of the Carlini&WagnerL2 attack as follows:

$$\begin{aligned} &minimize\ \|x - x^a\|^2 + c \cdot l_c(x^a) + d \cdot l_d(x^a) \\ &l_d = \sum_{n=1}^{N} max(D_n(x^a) - \tau_n^i, 0), \end{aligned} \tag{15}$$

where x is the original input, x^a is the adversarial sample, $l_c(\cdot)$ is a cost designed to mislead the victim classifier, and $l_d(\cdot)$ is a cost designed to deceive the defender models. Parameters c and d are constants which are tuned using binary search. We followed the instructions of [12] to set the loss function l_d in order to incorporate the MRR specific defense parameters. In particular, the value $D_n(x^a)$ in Eq. (15) is the L_2 distance $\|f(x^a) - C^i\|^2$ in the nth defender, i is the target class for misclassification attack, and τ_n^i is the cut-off threshold for class i in the nth defender. We set the learning rate of the attack to 0.01, the confidence rate to 0, the maximum number of iterations to 10,000, and the number of binary searches to 20 as suggested in [12]. For attack implementation, we used the library provided by the authors of the Carlini&WagnerL2 adaptive attack[5] and incorporated our defender specific loss into their algorithm.

Table 5 presents the success rate of the attack algorithm for different number of defender modules and security parameters for the MNIST benchmark. As shown in Table 5, with a single defender, the defense success rate (TP) is 43 and 46 percent for security parameters of SP = 1% and SP=5%, respectively. If we employ 16 defender modules in parallel, the TP rate increases to 66 percent for SP = 1% and 84% for SP=5%, at the cost of a small increase in FP rate. Higher number of MRR modules in our defense mechanism also results in a larger perturbation in the generated adversarial samples. Note that such high perturbations can be detected via CuRTAIL input defenders; however, we did not include input defenders in this evaluation for comparison fairness. Compared to the state-of-the-art defenses of

4. https://github.com/tensorflow/cleverhans

5. https://github.com/carlini/MagNet

TABLE 4
AUC Obtained by 16 Latent Defenders Checkpointing the Second-to-Last Layer of the Victim Model

	MNIST	SVHN	CIFAR-10	CIFAR-100	ImageNet
FGS	0.997	0.969	0.911	0.885	0.881
JSMA	0.995	0.995	0.966	0.961	-
Deepfool	0.996	0.974	0.960	0.850	0.908
CarliniL2	0.987	0.963	0.929	0.944	0.907
BIM	0.994	0.931	0.907	0.821	0.820

For ImageNet, we only used 1 defender due to the high computational complexity of the pertinent neural network and attacks.

Table 5, CuRTAIL achieves a better TP rate. In addition, the attacker is required to inject a higher amount of perturbation (in terms of L_2 norm) to mislead CuRTAIL defenders in a white-box setting.

6.5 Performance Analysis

We implement the customized defender modules on *Xilinx Zynq-ZC702* and *Xilinx UltraScale-VCU108* FPGA platforms. All modules are synthesized using *Xilinx Vivado v2017.2* . We integrate the synthesized modules into a system-level block diagram with required peripherals, such as the DRAM, using Vivado IP Integrator. The frequency is set to $150\ MHz$ and power consumption is estimated using the synthesis tool. For comparison purposes, we evaluate CuRTAIL performnace against a highly-optimized TensorFlow-based implementation on two low-power embedded boards: (i) The *Jetson TK1* development kit which contains an *NVIDIA Kepler* GPU with 192 CUDA Cores as well as an *ARM Cortex-A15* 4-core CPU. (ii) A more powerful *Jetson TX2* board with an *NVIDIA Pascal* GPU with 256 cores and a 6-core *ARM v8* CPU.

Robustness and Throughput Trade-Off. Increasing the number of checkpoints improves the reliability of model prediction in presence of adversarial attacks (Section 6.2) at the cost of reducing the effective throughput of the system. In applications with severe resource constraints, it is crucial to optimize system performance to ensure maximum immunity while adhering to the user-defined timing constraints. In scenarios with more flexible timing budget, the customization tool automatically allocates more instances of the defender modules while under strict timing constraints, the robustness is decreased in favor of the throughput.

Fig. 25 demonstrates the throughput versus the number of defender modules for MNIST benchmark on Zynq

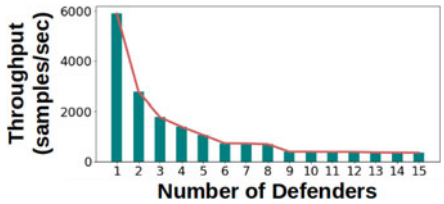

Fig. 25. CuRTAIL throughput with samples from the MNIST dataset versus the number of instantiated defenders (implemented on *Xilinx Zync-ZC702* FPGA).

FPGA. CuRTAIL accelerator can reach a throughput of 1,400 samples per second with 16 MRRs. For the SVHN benchmark, which has a similar DNN architecture to MNIST, the ARM v8 CPU can achieve a throughput of 1,400 samples per second when only one defender is executed. CuRTAIL implementation on UltraScale FPGA can run 8 defenders in parallel the same throughput. This directly translates to an improvement in the AUC from 0.76 to 0.96.

Throughput and Energy Analysis. To corroborate the efficiency of CuRTAIL, we also evaluate MRR performance on *TK1* and *TX2* boards operating in CPU-GPU and CPU-only modes. We define the performance-per-Watt measure as the throughput over the total power consumed by the system. This metric is an effective representation of the system performance since it integrates two influential factors for real-world embedded applications. All evaluations in this section are performed with only one input and latent defender. Fig. 26 (left) illustrates the performance-per-Watt for different hardware platforms. Numbers are normalized by the performance-per-Watt of the *TK1* platform in CPU mode. As shown, CuRTAIL implementation on Zynq shows an average of $38\times$ improvement over *TK1* and $6.2\times$ improvement over *TX2* in CPU mode. The more expensive UltraScale FPGA performs relatively better with an average improvement of $193\times$ and $31.7\times$ over *TK1* and *TX2*, respectively.

Fig. 26 (right) shows the comparisons with GPU platforms. All values are normalized by the *TK1* performance-per-Watt in the CPU-GPU mode. Our evaluations show an average of $9\times$ and $45.7\times$ improvement over *TK1* by the Zynq and UltraScale FPGAs, respectively. Comparisons with *TX2* demonstrate $2.74\times$ and $41.5\times$ improvement for the Zynq and UltraScale implementations. Note that UltraScale performs noticeably better than Zynq which emphasizes the effect of resource constraints on parallelism and throughput.

TABLE 5
Evaluation of MRR Methodology Against Adaptive White-Box Attack

	MRR Methodology (White-box Attack)												Prior-Art Defenses (Gray-box Attack)		
Security Parameter	SP=1%						SP=5%						Magnet	Efficient Defenses	APE-GAN
Number of Defenders	N=0	N=1	N=2	N=4	N=8	N=16	N=0	N=1	N=2	N=4	N=8	N=16	N=16	-	-
Defense Success (TP Rate)	-	43%	53%	64%	65%	66%	-	46%	63%	69%	81%	84%	1%	0%	0%
Normalized Distortion (L_2)	1.00	1.04	1.11	1.12	1.31	1.38	1.00	1.09	1.28	1.28	1.63	1.57	1.37	1.30	1.06
FP Rate	-	2.9%	4.4%	6.1%	7.8%	8.4%	-	6.9%	11.2%	16.2%	21.9%	27.6%	-	-	-

We compare our results with prior work including Magnet [26], Efficient Defenses Against Adversarial Attacks [30], and APE-GAN [31]. For each evaluation, the L_2 distortion is normalized to that of the attack without the presence of any defense mechanism. Note that highly disturbed images (with large L_2 distortions) can be easily detected using the input defenders; however, for fair comparison to prior work, we did not include our non-differentiable input defenders in this experiment.

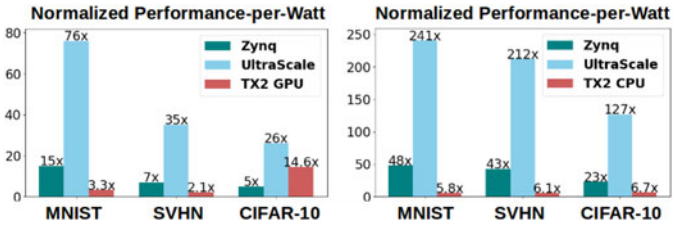

Fig. 26. Performance-per-Watt comparison with embedded CPU (left) and CPU-GPU (right) platforms. Reports are normalized by the performance-per-Watt of *TK1* .

6.6 Discussion on Transferability of Adversarial Samples

Fig. 27 demonstrates an example of the adversarial confusion matrices for victim neural networks with and without using parallel checkpointing learners. In this example, we set the security parameter to only 1 percent. As shown, the adversarial sample generated for the victim model *are not transferred* to the checkpointing modules. In fact, the proposed approach can effectively remove/detect adversarial samples by characterizing the rarely explored sub-spaces and looking into the statistical density of data points in the pertinent space.

Note that the remaining adversarial samples that are not detected in this experiment are crafted from legitimate samples that are inherently hard to classify even by a human observer due to the closeness of decision boundaries corresponding to such classes. For instance, in the MNIST application, such adversarial samples mostly belong to class 5 that is misclassified to class 3 or class 4 misclassified as 9. Such misclassifications are indeed the model approximation error which is well-understood to the statistical nature of the models. As such, a more precise definition of adversarial samples is extremely required to distinguish malicious samples form those that simply lie near the decision boundaries.

7 Related Work

The threat of adversarial samples to the integrity of autonomous systems have been shown in the literature for both shallow and deep models [3], [10], [14], [32], [33], [34], [35], [36]. Related work ties the existence of adversarial samples to several factors, including high feature dimensionality [10], bias to texture [37], and inherent brittle features [38]. DL models trained on ImageNet are shown to have bias towards texture rather than object shapes, resulting in low robustness against distortions [37]. This can be tied directly with CuRTAIL hypothesis in the existence of rarely explored regions in trained models. To address this, authors of [37] propose augmenting the training data with strong textual cues, thus forcing the model to focus on shapes rather than textures. Such augmentations effectively reduce the rarely explored regions and increase robustness. Authors of [38] show that seemingly unrelated features, generally imperceptible to humans, can inherently exist in visual datasets. Surprisingly, such brittle features contribute significantly to the underlying model's generalization capability during training. Authors also connect adversarial samples to these brittle features. Building upon this connection, CuRTAIL input defenders aim at identifying such redundant feature at test time.

In response to the various adversarial attacks proposed in the literature [3], [4], [5], [10], several research attempts have been made to design DL strategies that are more robust in the face of adversarial examples. Existing countermeasures can be classified into two categories:

(i) Supervised strategies which incorporate noisy inputs as training samples [39], [40] and/or inject adversarial examples into the training phase [10], [36], [41], [42]. Such defenses are tailored for specific perturbation patterns and can only partially evade adversarial samples generated by other attack scenarios [40].

(ii) Unsupervised approaches which aim to smoothen the underlying gradient space (decision boundaries) by incorporating a regularization term in the loss function [5], [43] or compressing the neural network [9]. These works are mainly oblivious to the pertinent data density as they implicitly assume that adversarial samples originate from the piece-wise linear behavior of decision boundaries. As such, their integrity can be jeopardized by crafting data points with specific perturbations that pass the smoothened decision boundaries [44]. [26] proposes manifold projection via auto-encoders to reform adversarial samples, which can be evaded by adaptive gray-box attacks as shown in [12].

The above works generally aim at correcting the decision of the victim network in face of adversaries. Alternatively, a line of research (including CuRTAIL) focuses on detection of adversarial samples without decision correction. They speculate that adversarial samples are not drawn from the same distribution as legitimate data. [45] proposes using Local Intrinsic Dimensionality (LID) to characterize properties of adversarial examples. However, LID is not able to

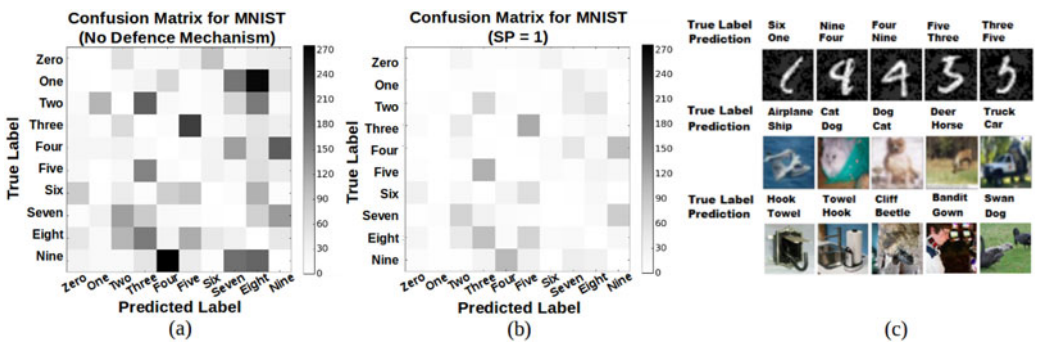

Fig. 27. Example adversarial confusion matrix (a) without MRR defense mechanism and (b) with MRR defense and $SP = 1\%$. (c) Example adversarial samples for which accurate detection is hard due to the closeness of decision boundaries.

detect high confidence adversarial examples [46]. [47] measures the uncertainty in the victim model's predictions using Mahalanobis distance-based scores. [48] attempts to detect adversaries by measuring the changes in the victim model's logits as the input is randomly perturbed. Nevertheless, all aforementioned methods study the statistics of the victim model's features which cannot optimally identify the rarely explored regions. This is because the victim model's main task and objective is accurate classification of *explored* regions. CuRTAIL alternatively focuses on altering the training objective of the MRRs to enable high-margin robust classification, which is later used for PDF estimation.

CuRTAIL unsupervised defense does not assume any particular attack strategy and/or perturbation pattern and is capable of withstanding all the existing attacks to date. Note that we target detection of adversarial samples without decision correction. As such, a stand-alone application of CuRTAIL remains vulnerable to denial of service attacks.

8 Conclusion

This paper proposes CuRTAIL, a novel end-to-end framework for online accelerated defense against adversarial samples in the context of deep learning. We introduce modular robust redundancy as a viable countermeasure to significantly reduce the risk of integrity attacks. The proposed MRR methodology explicitly characterizes statistical properties of the features within different layers of a neural network by learning the corresponding probability density functions. Using a hardware/algorithm co-design approach, CuRTAIL automated customization tool optimizes the defense layout to maximize model reliability (safety) while complying with the hardware and/or user constraints. This, in turn, ensures applicability to various deep learning tasks and hardware platforms. CuRTAIL robustness is evaluated against a wide range of attack models including FGS, Deepfool, BIM, and Carlini&WagnerL2. Proof-of-concept experiments on various data collections including MNIST, SVHN, CIFAR, and a subset of ImageNet dataset corroborate successful detection of adversarial samples with relatively small probability of false alarm. Our evaluations on various hardware platforms indicates the effectiveness and practicality of CuRTAIL.

Acknowledgments

This work was supported by the ARO Grant W911NF1910317 and the SRC-Auto Grant 2019-AU-2899.

References

[1] P. McDaniel, N. Papernot, and Z. B. Celik, "Machine learning in adversarial settings," *IEEE Secur. Privacy*, vol. 14, no. 3, pp. 68–72, May/Jun. 2016.
[2] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 3422–3426.
[3] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2016, pp. 372–387.
[4] 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.*, 2016, pp. 2574–2582.
[5] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 39–57.
[6] B. Wang *et al.*, "A theoretical framework for robustness of (deep) classifiers under adversarial noise," 2016, *arXiv:1612.00334*.
[7] M. Denil *et al.*, "Predicting parameters in deep learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.
[8] A. Madry *et al.*, "Towards deep learning models resistant to adversarial attacks," 2017, *arXiv: 1706.06083*.
[9] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2016, pp. 582–597.
[10] I. J. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*.
[11] A. Kurakin *et al.*, "Adversarial examples in the physical world," 2016, *arXiv:1607.02533*.
[12] N. Carlini and D. Wagner, "Magnet and "efficient defenses against adversarial attacks" are not robust to adversarial examples," 2017, *arXiv: 1711.08478*.
[13] B. Rouhani *et al.*, "DeepFense: Online accelerated defense against adversarial deep learning," in *Proc. Int. Conf. Comput.-Aided Des.*, 2018, Art. no. 134.
[14] L. Huang *et al.*, "Adversarial machine learning," in *Proc. 4th ACM Workshop Secur. Artif. Intell.*, 2011, pp. 43–58.
[15] M. Barreno *et al.*, "Can machine learning be secure?" in *Proc. ACM Symp. Inf. Comput. Commun. Secur.*, 2006, pp. 16–25.
[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[17] C. Bouveyron *et al.*, "High-dimensional discriminant analysis," *Commun. Statist. Theory Methods*, vol. 36, pp. 2607–2623, 2007.
[18] B. D. Rouhani *et al.*, "Deep3: Leveraging three levels of parallelism for efficient deep learning," in *Proc. 54th Annu. Des. Automat. Conf.*, 2017, Art. no. 61.
[19] B. Efron *et al.*, "Least angle regression," *Ann. Statist.*, vol. 32, no. 2, pp. 407–499, 2004.
[20] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Inf. Theory*, vol. 53, no. 12, pp. 4655–4666, Dec. 2007.
[21] F. J. Diez, "Parameter adjustment in Bayes networks. The generalized noisy OR–gate," in *Proc. 9th Int. Conf. Uncertainty Artif. Intell.*, 1993, pp. 99–105.
[22] H. Sharma *et al.*, "From high-level deep neural models to FPGAs," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, Art. no. 17.
[23] G. H. Golub and C. F. Van Loan, *Matrix Computations*, vol. 3. Baltimore, MD, USA: The John Hopkins Univ. Press, 2012.
[24] B. D. Rouhani, E. M. Songhori, A. Mirhoseini, and F. Koushanfar, "SSketch: An automated framework for streaming sketch-based analysis of big data on FPGA," in *Proc. IEEE 23rd Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2015, pp. 187–194.
[25] K. He *et al.*, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.
[26] D. Meng and H. Chen, "MagNet: A two-pronged defense against adversarial examples," 2017, *arXiv: 1705.09064*.
[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.
[28] A. Krizhevsky *et al.*, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[29] N. Papernot *et al.*, "CleverHans v2.0.0: An adversarial machine learning library," 2017, *arXiv:1610.00768*.
[30] V. Zantedeschi *et al.*, "Efficient defenses against adversarial attacks," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 39–49.
[31] S. Shen *et al.*, "APE-GAN: Adversarial perturbation elimination with GAN," *ICLR Submission, Available on OpenReview*, 2017.
[32] B. Biggio *et al.*, "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2013, pp. 387–402.
[33] B. Biggio *et al.*, "Pattern recognition systems under attack: Design issues and research challenges," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 28, no. 07, 2014, Art. no. 1460002.
[34] A. Anjos and S. Marcel, "Counter-measures to photo attacks in face recognition: A public database and a baseline," in *Proc. Int. Joint Conf. Biometrics*, 2011, pp. 1–7.
[35] P. Fogla and W. Lee, "Evading network anomaly detection systems: Formal reasoning and practical techniques," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 59–68.

[36] C. Szegedy *et al.*, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*.
[37] R. Geirhos *et al.*, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," 2018, *arXiv: 1811.12231*.
[38] A. Ilyas *et al.*, "Adversarial examples are not bugs, they are features," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 125–136.
[39] J. Jin *et al.*, "Robust convolutional neural networks under adversarial noise," 2015, *arXiv:1511.06306*.
[40] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," 2014, *arXiv:1412.5068*.
[41] R. Huang *et al.*, "Learning with a strong adversary," 2015, *arXiv:1511.03034*.
[42] U. Shaham *et al.*, "Understanding adversarial training: Increasing local stability of neural nets through robust optimization," 2015, *arXiv:1511.05432*.
[43] T. Miyato *et al.*, "Distributional smoothing with virtual adversarial training," 2015, *arXiv:1507.00677*.
[44] N. Carlini and D. Wagner, "Defensive distillation is not robust to adversarial examples," 2016, *arXiv:1607.04311*.
[45] X. Ma *et al.*, "Characterizing adversarial subspaces using local intrinsic dimensionality," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.
[46] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 3–14.
[47] K. Lee *et al.*, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7167–7177.
[48] K. Roth *et al.*, "The odds are odd: A statistical test for detecting adversarial examples," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5498–5507.

Mojan Javaheripi (Student Member, IEEE) received the BSc degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2017. She is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of California, San Diego, California. Her research interests include co-developing machine learning algorithms and their corresponding specialized hardware with the goal of maximizing efficiency and performance.

Mohammad Samragh (Student Member, IEEE) received the BSc degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2015. He is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of California, San Diego, California. His research interests include deep learning, safety and robustness of machine learning algorithms, hardware acceleration of learning algorithms, and low-power computing.

Bita Darvish Rouhani (Student Member, IEEE) received the BSc degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2013, and the MSc degree in electrical and computer engineering from Rice University, Houston, Texas, in 2015. She is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of California, San Diego, California. Her research interests include deep learning, sfatey of machine learning models, low-power computing, distributed optimization, and big data analysis.

Tara Javidi received the BSc degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1996, the MSc degree in electrical engineering: systems and applied mathematics: stochastics from the University of Michigan, Ann Arbor, Michigan, and the PhD degree in electrical engineering and computer science (EECS) from the University of Michigan, Ann Arbor, Michigan, in 2002. From 2002 to 2004, she was an assistant professor with the Electrical Engineering Department, University of Washington, Seattle. She joined the University of California, San Diego in 2005, where she is currently a professor of electrical and computer engineering. She was a Barbour Scholar during 1999–2000 academic year and received the NSF CAREER Award, in 2004. Her research interests include communication networks, stochastic resource allocation, and wireless communications

Farinaz Koushanfar (Fellow, IEEE) received the BS degree from the Sharif University of Technology, Tehran, Iran, the MS degree from the UCLA, Los Angeles, California, both in electrical engineering, and the MA degree in statistics and the PhD degree in electrical engineering and computer science, both from UC Berkeley, Berkeley, California, in 2005. She is a professor and Henry Booker Faculty Scholar with the Department of Electrical and Computer Engineering, University of California, San Diego where she is directing the Adaptive Computing and Embedded Systems (ACES) Lab. Before joining the faculty with UCSD, she was a professor of electrical and computer engineering with Rice University. Her research interests include embedded and cyber-physical systems design, embedded systems security, and design automation of domain-specific/mobile computing and machine learning applications. She has received a number of awards and honors for her research, mentorship, and teaching, including the Presidential Early Career Award for Scientists and Engineers (PECASE) from President Obama, the ACM SIGDA Outstanding New Faculty Award, MIT TR-35, and Young Faculty/CAREER awards from NSF, DARPA, ONR, and ARO. She is a fellow of the Kavli Foundation Frontiers of the National Academy of Engineering.