# CLEANN: Accelerated Trojan Shield for Embedded Neural Networks

Mojan Javaheripi*
UC San Diego
mojan@ucsd.edu

Mohammad Samragh*
UC San Diego
msamragh@ucsd.edu

Gregory Fields
UC San Diego
grfields@ucsd.edu

Tara Javidi
UC San Diego
tjavidi@ucsd.edu

Farinaz Koushanfar
UC San Diego
farinaz@ucsd.edu

## ABSTRACT

We propose CLEANN, the first end-to-end framework that enables online mitigation of Trojans for embedded Deep Neural Network (DNN) applications. A Trojan attack works by injecting a backdoor in the DNN while training; during inference, the Trojan can be activated by the specific backdoor trigger. What differentiates CLEANN from the prior work is its lightweight methodology which recovers the ground-truth class of Trojan samples without the need for labeled data, model retraining, or prior assumptions on the trigger or the attack. We leverage dictionary learning and sparse approximation to characterize the statistical behavior of benign data and identify Trojan triggers. CLEANN is devised based on algorithm/hardware co-design and is equipped with specialized hardware to enable efficient real-time execution on resource-constrained embedded platforms. Proof of concept evaluations on CLEANN for the state-of-the-art Neural Trojan attacks on visual benchmarks demonstrate its competitive advantage in terms of attack resiliency and execution overhead.

## KEYWORDS

Deep Learning, Trojan Attack, Embedded Systems, Sparse Recovery

**ACM Reference Format:**
Mojan Javaheripi, Mohammad Samragh, Gregory Fields, Tara Javidi, and Farinaz Koushanfar. 2020. CLEANN: Accelerated Trojan Shield for Embedded Neural Networks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415671

## 1 INTRODUCTION

With the growing popularity of AI-powered autonomous systems, the demand for superior intelligence has led to increasingly more complex model development processes. Training contemporary deep learning models requires massive datasets and high-end hardware platforms [18, 26]. Amid this trend, clients rely on third party databases and/or major cloud providers to build their models. Unfortunately, outsourcing of content or computations opens up new challenges as it extends the potential attack surface to malicious third party entities [29]. In this paper, we focus on Trojan attacks [14, 23], where the malicious third party provider inserts a hidden Trojan trigger, also dubbed a "backdoor", inside the model during training. During inference, the attacker can hijack the model prediction by inserting the Trojan trigger inside the input data. Figure 1 illustrates examples of Neural Trojans.

*Both authors contributed equally to this research.


*ICCAD '20, November 2–5, 2020, Virtual Event, USA*

ACM ISBN 978-1-6654-2324-3/20/11.
https://doi.org/10.1145/3400302.3415671


**Figure 1: Example Trojans: (a) BadNets [14] with a sticky note and TrojanNN [23] with (b) square and (c) watermark triggers.**

Identification and mitigation of Trojans is particularly challenging for the clients since the compromised model performs as expected on their benign data, i.e., when the Trojan is not activated. To tackle Trojan attacks, contemporary research proposes either reverse-engineering the trigger pattern from the model [5, 15, 22, 36] or identifying the presence of a trigger at the input [6, 8, 12]. The former class of methods require time-consuming reverse-engineering and retraining. The latter approaches induce a high overhead on DNN inference that hinders their applicability to embedded systems. To ensure model robustness in safety-sensitive autonomous systems, it is crucial to augment the models with an online Trojan mitigation strategy. To the best of our knowledge, none of the earlier works provide the needed lightweight defense strategy.

We propose CLEANN, the first end-to-end accelerated framework that enables real-time Trojan shield for embedded DNN applications. CLEANN's lightweight method is devised based on algorithm/hardware co-design; our algorithmic insights offer a highly accurate and low-overhead method in terms of both the offline defense establishment and online execution; our hardware accelerator enables low-latency and energy-efficient defense execution on embedded platforms. CLEANN harvests the irregular patterns caused by Trojan triggers in the input space and/or the latent feature-maps of the victim DNN to detect adversaries. Our method leverages key concepts and theoretical bounds from sparse approximation [9] to learn dictionaries that absorb the distribution of the benign data. We then utilize the reconstruction error obtained from the sparse approximation to characterize the benign space and identify the Trojans.

To ensure applicability to various attacks and trigger patterns, CLEANN sparse recovery acts on both frequency and spatial domains. Our proposed defense is compatible with the challenging threat model in which the attacker has full control over the geometry, location, and content of the Trojan trigger. The contaminated model is shipped to the client, who is unaware of the existence of the Trojan and does not have access to any labeled data. CLEANN countermeasure is unsupervised, meaning that no labeled training data or contaminated Trojan sample is required to establish the defense. Notably, CLEANN is the first defense to recover the ground-truth labels of Trojan data without performing any model training and/or fine-tuning.

We validate the effectiveness of CLEANN by performing extensive experiments on various state-of-the-art Trojan attacks reported to-date. CLEANN outperforms prior art both in terms of Trojan resiliency and algorithm execution overhead. CLEANN brings down

the attack success rate to 0% for a variety of physical [14] and complex digital [23] attacks with minimal drop in classification accuracy. Our customized accelerated defense shows orders of magnitude higher throughput and performance-per-watt compared to commodity hardware. In brief, the contributions of CLEANN are as follows:

- Introducing CLEANN, the first end-to-end accelerated framework for online detection of Neural Trojans in embedded applications.
- Constructing a novel unsupervised Trojan detection scheme based on sparse recovery and outlier detection. The proposed lightweight defense is, to our best knowledge, the first to enable recovering the original label of Trojan samples without model fine-tuning/training.
- Providing bounds on detection false positive rate using the theoretical ground of sparse approximation and outlier detection.
- Devising the first customized library of Trojan shields on FPGA which enables high-throughput and low-energy Trojan mitigation.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Trojan Attacks

Throughout this paper, we focus on Trojan attacks on DNN classifiers. Below, we overview state-of-the-art attack algorithms.

► **BadNets.** Authors of *BadNets* [14] propose adding the Trojan trigger into a random subset of training samples and labeling them as the attack target class. The DNN is then trained on the poisoned dataset. The shape of the Trojan trigger can be arbitrarily chosen by the attacker, e.g., a sticky note on a stop sign as shown in Figure 1-a. Thus, BadNets are considered a viable **physical** attack.

► **TrojanNN.** More recently, *TrojanNN* [23] assumes the attacker does not have access to the training data but can modify the DNN weights. The attack first selects one or few neurons in one of the hidden layers, then extracts the Trojan trigger in the input domain to activate the target neurons. The DNN weights are then modified such that the model predicts the attacker's target class whenever the selected neurons fire. Unlike BadNets, the triggers generated by TrojanNN, e.g., the square and watermark patterns in Figure 1-c,d, are not similar to natural images. However, TrojanNN is a viable attack algorithm in the **digital** domain; Notably, most Trojan mitigation methods are less successful in identifying the complex triggers of TrojanNN [5, 24].

### 2.2 Existing Defense Strategies

► **Robust Training and Fine-tuning.** One plausible threat model assumes that the client has access to the training dataset but is unaware of the existing Trojans. Robust learning methods aim at identifying malicious samples during training [3, 20, 34]. For an already infected DNN, authors of [21] perform pruning to remove the embedded Trojans at the cost of clean accuracy degradation. We assume a more constrained attack model where the victim does not have any access to the training dataset. Additionally, CLEANN does not rely on expensive model retraining to establish the defense.

► **Trigger Extraction.** Several methods inspect the DNN model for existence of a backdoor attack by reverse engineering the trigger. Neural Cleanse [24] provides a method for extracting Trojan triggers without access to the training dataset. Follow up work improves the search overhead [5] and reverse engineered trigger quality [15]. Though effective for simple Trojan patterns, their performance drops when reverse engineering more complex triggers, e.g., those created by TrojanNN [23]. Our method is different than the above works in that, instead of reverse engineering the trigger, we study the statistics of sparse representations from benign samples and detect abnormal (outlier) triggers during inference. This allows CLEANN to identify complex Trojan triggers without prior knowledge about the attack algorithm. Additionally, CLEANN does not involve expensive reverse engineering and can be executed in real-time on embedded hardware.

► **Data and Model Inspection.** Perhaps the closest method to CLEANN are those that check the input samples to identify the presence of Trojan triggers. Authors of [4] query the infected model and use activation clustering on hidden layers to detect Trojans. Similarly, NIC [25] compares incoming samples against the benign and Trojan latent features to detect adversaries. These method require access to the labeled contaminated training dataset, which may not viable in real-world settings. CLEANN, in contrast, does not require access to the training data or infected data samples to construct the defense.

Sentinet [6] extract critical regions from input data using gradient information obtained by back propagation. Februus [8] takes a similar approach along with utilizing GANs to inpaint Trojan triggers with the caveat that the number of data samples required for GAN training is large. STRIP [12] runs the model multiple times on each image with intentional injected noise to identify Trojans. While the above works show high detection accuracy, their computational burden of multiple forward/backward propagations is prohibitive for embedded applications. CLEANN achieves a better detection accuracy with low computational complexity and sample count, making it amenable for real-time deployment in embedded systems.

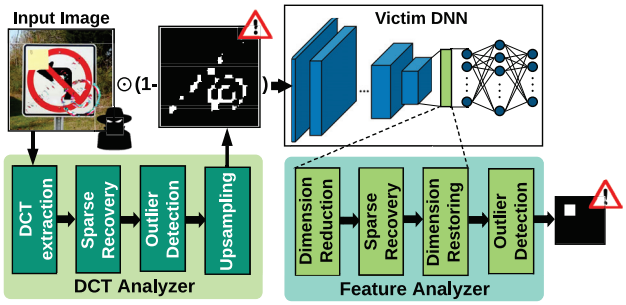


**Figure 2: High-level overview of CLEANN Trojan detection methodology. CLEANN detects both digital and physical attacks using a pair of input and latent feature analyzers.**

## 3 CLEANN METHODOLOGY

Figure 2 illustrates the high-level flow of CLEANN methodology for Trojan detection. CLEANN comprises two core modules, dubbed the DCT and feature analyzers, specializing in the characterization of the DNN input space and latent representations, respectively. By aggregating the decision of the two analyzers, CLEANN is able to thwart a wide range of physical and digital Trojan attacks.

❶ **DCT Analyzer.** The DCT analyzer acts as an image preprocessing step. This module investigates all incoming samples in the frequency domain in search for suspicious frequency components that are anomalous in clean data. Towards this goal, we design four components for this module as shown in Figure 2. First, the Discrete Cosine Transform (DCT) extraction module transforms the input image to the frequency domain. We then perform sparse recovery on the extracted frequency components and reconstruct the signal using

a sparse approximation. The outlier detection module uses a concentration inequality to detect anomalous reconstruction errors and generate a binary mask with non-zero values denoting the potential Trojan-carrying regions. The anomalous regions in the input image are then suppressed by the binary mask before entering the victim DNN. To ensure compatible dimensions between the input image and the binary mask, a nearest neighbor upsampling component is also included inside the DCT analyzer. Frequency analysis is particularly useful for detecting digital Trojans. However, physical attacks, e.g., the sticky note in Figure 2, might evade frequency-domain detection.

❷ **Feature Analyzer.** This module investigates patterns in the latent features extracted by the victim DNN to find abnormal structures. The feature analyzer is placed at the penultimate layer inside the victim DNN. This choice of location allows us to leverage all the visual information extracted from the input image by the DNN for making the classification decision. The sparse recovery module in the feature analyzer serves two purposes: (i) denoising input features for use in the remaining layers of the victim DNN, (ii) anomaly detection on the reconstruction errors for distinguishing Trojans. Notably, the first property allows CLEANN to recover the ground-truth labels for Trojan samples by effective removal of Trojan triggers. To ensure scalability to various output dimensions, we include a dimension reduction module that adaptively adjusts the feature size while maximally preserving the informative content of the signals. To allow the reconstructed output to flow in the remaining layers of the DNN, a twin dimension restoring layer recovers the original tensor shape. The extracted distributions from latent layers successfully detect attacks in the physical domain.

### 3.1 Defense Construction and Execution

CLEANN consists of two main phases to mitigate Trojan attacks:

► **Offline Preprocessing.** During this phase, we learn the parameters for dictionary-based sparse recovery and outlier detection modules by leveraging a small set of unlabeled benign samples. Our methodology is entirely unsupervised, meaning no Trojan data is involved in defense construction. This, in turn, ensures applicability to a wide range of Trojan patterns and attacks. CLEANN pre-processing phase is low-complexity as it does not involve any training or fine-tuning of the victim DNN. We only perform this step once for each (model, dataset) pair. The learned analyzer modules can then be transferred to a variety of attacks without any fine-tuning overhead.

► **Online Execution.** CLEANN methodology is devised based on light-weight solutions to enable efficient adoption in embedded systems. We provide a hardware-accelerated pipeline for end-to-end execution of CLEANN where the analyzer modules are either integrated inside the victim DNN or running in parallel with it. The DCT extraction and upsampling components are implemented as an additional convolution layer at the input of the victim DNN. We devise a customized library for implementing the sparse recovery, outlier detection, and dimensionality reduction and restoring modules on FPGA. These FPGA-accelerated modules are executed synchronously with the victim DNN to raise alarm flags for Trojans.

### 3.2 Threat Model

In our threat model, we assume the client has purchased the trained DNN model infected with Trojans from a malicious party. Accordingly, we consider the following constraints on our defense strategy: (1) The client has access to model weights but not the training data. (2) The client has access to clean test data but they are unlabeled. (3) The client is not aware whether or not the model is infected with Trojans. (4) No prior knowledge is available about possible Trojan trigger shapes and/or patterns.

To construct the defense, we assume access to a small corpus of *unlabeled* data[1]. This is a realistic assumption as access to small amounts of data is possible via online resources. For instance, publicly available repositories enable data generation through generative networks for Faces[2]. We consider the most generic and challenging form of Trojan attacks in which the attacker can control the trigger size, shape, and content. CLEANN mitigation is made possible in such scenario by constructing the defense using benign unlabeled data.

## 4 CLEANN COMPONENTS

### 4.1 DCT extraction

In natural images, most of the energy is contained in low frequencies. However, this property does not necessarily hold true for the Trojan triggers. Figure 3 shows the visualization of the frequency components for a Trojan sample, normalized by the magnitude of frequency components for benign data. Here, the magnitudes are averaged across 100,000 image patches and the Trojan samples contain a watermark trigger generated by [23]. As seen, Trojans have much larger components in the high-frequency domain compared to benign samples.

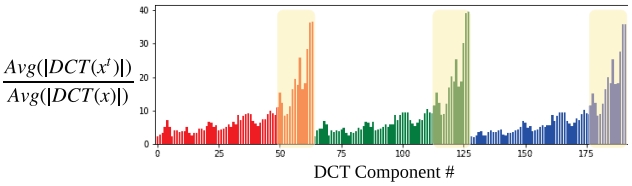


**Figure 3: Average magnitude of DCT components for Trojan samples, normalized by benign data, shown in the three RGB channels. Trojans contain abnormally larger amounts of high-frequency components (highlighted regions).**

To perform frequency analysis, we divide each input image into non-overlapping patches[3] of size $P \quad P$. We transform each image patch to another patch of same size in the frequency domain using DCT. Eq. (1) encloses the formula used to compute the DCT transformation $F_{u,v}$ of a $P \quad P$ patch.

$$F_{u,v} = C_{u,v} \sum_{i=0}^{P} \sum_{j=0}^{P} x_{i,j} \cos\left[\frac{u\pi}{P}\left(i \quad \frac{1}{2}\right)\right] \cos\left[\frac{v\pi}{P}\left(j \quad \frac{1}{2}\right)\right] \quad (1)$$

Here, $x_{i,j}$ is the input pixel located at the $i,j$ coordinate and $C_{u,v}$ is a scalar constant that depends on the frequency coordinates. The extracted $2D$ DCT components $F_{u,v}$ are then sorted in decreasing order in terms of the information they carry following a zigzag pattern [30].

We represent the DCT transform as a group convolution with kernel size $P$ and $c_{in}$ groups where $c_{in} = 3$ and 1 for RGB and grayscale images, respectively. The kernel weights of the convolution layer are initialized with the DCT basis coefficients which are precomputed based on Eq. (1). The stride of the convolution is set to $P$ to

[1] less than 1% of the training set size across all of our evaluations

[2] http://www.whichfaceisreal.com/index.php

[3] We use $P = 4$ for small image benchmarks where input image dimensions are less than 32 pixels. For larger input image sizes we use $P = 8$.

account for image patching. Such representation allows for an efficient implementation of the DCT Analyzer, which can be easily integrated into the architecture of the victim model as a pre-processing layer.

## 4.2 Sparse Recovery

Sparse coding is referred to learning methods where the goal is to efficiently represent the data using sets of over-complete bases. Given a matrix of ($n$) data observations $X \in \mathbb{R}^{l \times n}$, sparse coding extracts a dictionary of normalized basis vectors $D \in \mathbb{R}^{l \times m}$ and the sparse representation matrix $V \in \mathbb{R}^{m \times n}$. Formally, the sparse coding objective can be written as:

$$\min_{D,V} f_D(X) = \min_{D,V} \|X - D.V\|_2 + \gamma \|V\|_0 \quad (2)$$

where $\gamma$ is a regularization coefficient that promotes sparsity in the coded representation $V$. Dictionary learning algorithms provide solutions to the above optimization problem by finding a dictionary $D$ that minimizes $\mathbb{E}_{x \sim} f_D(x)$, where $\sim$ is the distribution over the inputs. CLEANN extracts $D$ by performing dictionary learning over legitimate (benign) data. The out-of-distribution Trojan samples are thus expected to show a high reconstruction error, whereas benign samples will be accurately reconstructed with small error.

Figure 4 illustrates this behavior in an example $2D$ space. The light-blue dots represent the distribution of benign samples; the two solid arrows $d_1$ and $d_2$ are the dictionary atoms and only one of them is used for sparse reconstruction $\tilde{x}$. As seen, the magnitude of the reconstruction error on the outlier sample $x_2$ is larger than that of regular data $x_1$, i.e., $\|x_2 - \tilde{x}_2\|_F >> \|x_1 - \tilde{x}_1\|_F$ where $\|\cdot\|_F$ is the Frobenius norm.

While the above simple illustration shows the effectiveness of dictionary learning in 2 dimensions, a similar behavior is observed when generalizing sparse coding to higher dimensions. For a dictionary trained on $n$ samples $x \in \mathbb{R}^l$, there exist theoretical bounds on the generalization error for unseen samples drawn from the same distribution $\sim$. Let us denote the average reconstruction error over the set of $n$ observed samples by $E_o$. The generalization error of the dictionary $E_D$ on unseen samples $x_u \sim X$ is bounded by $E_D(x_u) \leq E_o + \delta$. Vainsencher et al. [35] prove that the generalization error $\delta$ for a $\lambda$-sparse representation is $\sqrt{ml \ln(n\lambda)/n}$ under some orthogonality assumptions for the dictionary. CLEANN dictionaries are devised to minimize reconstruction error on benign samples. We therefore carefully tune the dictionary size $m$ and sparsity level $\lambda$ to ensure a low reconstruction error on the data at hand ($E_o$) as well as a low error bound $\delta$.

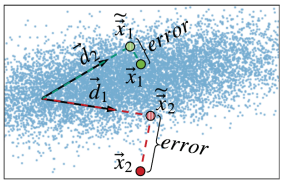


**Figure 4: Illustration of sparse reconstruction for regular data (green circle) and out-of-distribution samples (red circle).**

► **Data.** We apply sparse recovery on two data subsets extracted from a small corpus of randomly selected benign samples.

(1) At the input of the neural network (Section 4.1), each column of matrix $X$ is the DCT of a single patch in the input image. For instance, for an $8 \times 8$, DCT window, the dimensionality would be $l = 3 \times 64$ (64 DCT coefficients per RGB channel).

(2) At the latent space, each column of $X$ represents a flattened feature-map with reduced dimensionality.

► **Dictionary Learning.** We use an adaptive sampling distribution based on the reconstruction error of $X$, dubbed Column Selection-based Sparse Decomposition (CSSD) [27] for learning the dicrionaries. This algorithm initializes $D$ by a small random subset of $X$ and then iteratively adds columns to $D$; the probability of a data sample being appended at each step is proportional to its reconstruction error with the current column set. Formally, the probability of the $i$-th sample $x_i$ being selected at the $t+1$-th iteration is given by:

$$p(i) = \frac{\|D_t D_t^{+} x_i - x_i\|_2}{\|x_i\|_2} \quad (3)$$

where $D_t$ corresponds to the columns of the dictionary selected up to the $t$-th iteration and $D_t^{+} = (D_t^T D_t)^{-1} D_t^T$ is the pseudo inverse of $D_t$. The intuition behind Eq. (3) is to give a higher chance of selection to those elements of $X$ with higher reconstruction errors. This approach allows us to maximize the amount of embedded information from the data distribution inside $D$. While more sophisticated algorithms can be used [1, 2, 10], our empirical evaluations show that CSSD can sufficiently express the data distribution with minimal generalization error.

► **Reconstruction Algorithm.** We use Orthogonal Matching Pursuit (OMP) [7] for sparse recovery as summarized in Algorithm 1. OMP iteratively finds non-zero elements to construct the sparse representation $x$. The added non-zero element at each iteration is chosen such that it minimizes the $L_2$ norm of the remaining residual error $\|r_i - D v\|_2$ which can be solved using Least-square (LS) optimization. The subset of dictionary columns ($D$) that contribute to the sparse recovery is also expanded over iterations. After $\lambda$ iterations, the reconstruction is returned as $\tilde{x} = D v$.

**Algorithm 1** OMP algorithm

**Inputs:** Dictionary $D \in \mathbb{R}^{l \times m}$, input sample $x \in \mathbb{R}^l$, number of non-zero coefficients for sparse recovery ($\lambda$).
**Output:** reconstruction $\tilde{x} \in \mathbb{R}^l$.

```
r_0 ← x                                   ▷ residual error: r_0 ∈ R^l
D ← ∅                                     ▷ empty dictionary subset
for i = 0,..., λ − 1 do
    p = D^T r_i                           ▷ projection vector: p ∈ R^m
    j = argmax p
    D ← D ∪ D_{:,j}                       ▷ update dictionary subset
    v ← argmin ||r_i − D v||_2
    r_{i+1} ← r_i − D v                   ▷ update residual error
return D v
```

► **Distribution Learning with Few Samples.** An "over-complete" dictionary is necessary to ensure representation sparsity [27] and effective separation of outlier and benign samples. The term over-complete is used when the number of columns in the dictionary is higher than the data dimensionality ($m >> l$). In real-world DNN applications, however, the number of data samples ($m$) is often small while the feature-map dimensionality ($l$) is large. To tackle this, we apply Singular Value Decomposition on the high-dimensional feature-maps to reduce $l$. Inverse SVD can then be applied on the reconstructed output to recover the original dimensionality. We choose the SVD rank such that more than 90% of the original energy is preserved.

## 4.3 Outlier Detection

As discussed in Section 4.2, we leverage the disparity between the reconstruction error of benign and Trojan samples after undergoing

sparse recovery to detect Trojans. Towards this goal, we first extract the statistical properties of the reconstruction error across benign samples. The out-of-distribution samples, i.e., outliers, are then marked as Torjan. In order to model out of distribution samples, we utilize a multivariate extension of Chebyshev's inequality [33]. Consider a random variable $\mathbb{R}^1$ $^d$ and let $x_i$ $_{i=1}^N$ denote a set of observed samples drawn from . Based on the $N$ observations, we calculate the empirical mean $\mu$ and the covariance $\Sigma$ as follows:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i\,, \quad \Sigma = \frac{1}{N\ 1}\sum_{i=1}^{N} x_i\ \ \mu\ \ x_i\ \ \mu\ ^T \tag{4}$$

The Chebyshev's inequality provides an upper bound on the probability of samples lying outside ellipsoids of the form $x\ \ \mu\ \Sigma\ ^1\ x\ \ \mu\ ^T = \epsilon^2$. Let us denote the *distance* of each sample from the distribution by:

$$dist\ x\ =\ x\ \ \mu\ \Sigma\ ^1\ x\ \ \mu\ ^T \tag{5}$$

The Chebyshev's inequality can then be formally written as:

$$dist\ \ \epsilon^2\ \ \ min\left\{1, \frac{d\ N^2\ \ 1\ \ N\epsilon^2}{N^2\epsilon^2}\right\} \tag{6}$$

The above inequality implies that one can categorize samples satisfying large enough values of $\epsilon$ as out-of-distribution, i.e., outlier. Based on this intuition, we measure the empirical mean and covariance in Eq. (4) on a held-out dataset of benign samples and use the Chebyshev's inequality to characterize Trojaned data that do not belong to the benign probability distribution. The right-hand side of Eq. (6) provides the probability of a benign sample being categorized as outlier or Trojan. For large-enough values of $N$ ($N$ ), this probability tends to $min\{1, \frac{d}{\epsilon^2}\}$.

Figure 5-a, b illustrates example Trojan data together with the corresponding reconstruction error heat maps. As seen, the Trojan trigger patterns have relatively larger reconstruction error compared to the rest of the image. Figure 5-c visualizes the output of the outlier detection. Here, we generate a binary mask where the values of 0 and 1 correspond to in-distribution and outlier labels, respectively. As seen, parts of the input image that are covered with the Trojan trigger are correctly distinguished from benign regions.

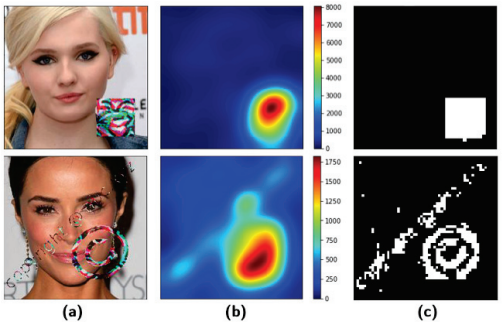


**Figure 5: (a) Example Trojan data with watermark and square triggers [23], (b) reconstruction error heatmap, and (c) output mask from the outlier detection module.**

► **Tuning the parameter $\epsilon$.** We provide a systematic way to tune the parameter $\epsilon$ for outlier detection, based on the user-defined constraints on Trojan defense performance. An incoming sample $I$ $\mathbb{R}^{d\ K\ K}$ is labeled as Trojan if at least one of its enclosing components $I_k$ $\mathbb{R}^d$ is categorized as an outlier based on Eq. (6). The probability of an image being categorized as Trojan is therefore:

$$_I\ Trojan\ = 1\ \prod_{k=1}^{K\ K}\ _{I_k}\ Benign \tag{7}$$

When examining the outlier detection scheme on benign samples, the left-hand side of Eq. (7) is equivalent to the False Positive Rate (FPR), i.e., the probability of a benign image being mistaken for a Trojan. Eq. (6) provides that for benign samples $_{I_k}\ Benign\ I_k\ \ Benign\ \ \ 1\ \ \frac{d}{\epsilon^2}$. The FPR is thus upper-bounded by:

$$FPR = \ _I\ Trojan\ I\ \ Benign\ \ \ 1\ \ \left(1\ \ \frac{d}{\epsilon^2}\right)^{K\ K} \tag{8}$$

We can therefore determine the parameter $\epsilon$ based on the desired application-specific FPR denoted by $FPR_{target}$:

$$\sup_{\epsilon} FPR = 1\ \ \left(1\ \ \frac{d}{\epsilon^2}\right)^{K\ K}\ \ \ FPR_{target} \tag{9}$$

$$\frac{d}{\epsilon^2}\ \ \ 1\ \ \ ^{K\ K}\sqrt{1\ \ FPR_{target}} \tag{10}$$

where $\frac{d}{\epsilon^2}$ is the per-patch FPR, i.e., $_{I_k}\ Trojan\ I_k\ \ Benign$ .

► **Reducing FPR with Morphological Transforms.** As seen in Figure 5, certain benign elements in the samples might be marked as Trojan, thus increasing the FPR. To reduce such patterns, we utilize two operations from morphological image processing, namely, erosion and dilation, implemented as convolution layers. Erosion emphasizes contiguous regions in the input mask and removes small, disjoint regions. Once erosion is applied, binary dilation restores high-density non-zero regions in the original input mask. Figure 6-a demonstrates the obtained binary mask from the outlier detection where the benign regions mistaken for being Trojan are marked with red boxes around them. Figure 6-b shows how erosion successfully removes the false alarms and Figure 6-c demonstrates how dilation restores the original shape of the binary mask in Trojan regions.

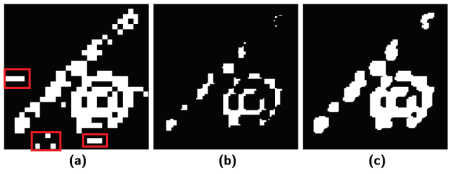


**Figure 6: (a) Binary Trojan mask with the red rectangles indicating False alarms. (b) Output mask obtained after applying $2D$ binary erosion. (c) Output mask after restoring the high-concentration Trojan regions with $2D$ binary dilation.**

## 4.4 Decision Aggregation

Figure 7 illustrates the decision flowchart for CLEANN Trojan detection. As shown, a successful Trojan attack needs to satisfy two conditions: (1) both the DCT and feature analyzers mistakenly mark the sample as benign, and (2) the victim model classifies the sample in the target Trojan class. For each Trojan sample $x_i^t$, the attack success $S_i$ is computed as:

$$S_i = \ 1\ \ d_{DA}\ x_i^t\ \ \ 1\ \ d_{FA}\ x_i^t\ \ \ \ x_i^t\ ==c^t \tag{11}$$

where $d_{DA}$ and $d_{FA}$ denote the decision of the DCT and feature analyzer modules, respectively, with the value of 1 meaning the Trojan has been detected. Here, represents the classification decision made by the victim model and $c_t$ is the Trojan attack target class. The overall attack success rate (ASR) is the expectation of $S$

over Trojan samples ($x^t$ $t$). Since the three terms in Eq. (11) are independent, we can write ASR as:

$$ASR = \mathbb{E}_{t}\ 1\ d_{DA}\quad \mathbb{E}_{t}\ 1\ d_{FA}\quad \mathbb{E}_{t}\quad x_i^t\ ==c^t \qquad (12)$$

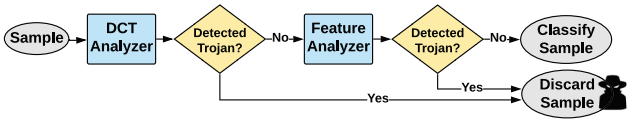


**Figure 7: Decision flowchart for Trojan detection in CLEANN.**

The first and second terms in the equation above are quantified using the True Positive detection rate (TPR). In this context, TPR measures the ratio of Trojan samples that are correctly identified by the defense. Let us denote the TPR for the DCT and feature analyziers with $TPR_{DA}$ and $TPR_{FA}$, respectively. Eq. (12) can then be equivalently written as:

$$ASR = 1\ \ TPR_{DA}\ \ 1\ \ TPR_{FA}\quad \frac{1}{N}\sum_{i=1}^{N}\quad x_i^t\ ==c^t \qquad (13)$$

Similarly, the classification accuracy on benign samples $ACC\ \ C$ can be written in terms of the FPR of the DCT and feature analyzers:

$$ACC\ \ C = 1\ \ FPR_{DA}\ \ 1\ \ FPR_{FA}\quad \frac{1}{N}\sum_{i=1}^{N}\quad x_i\ ==c_i \qquad (14)$$

where $c_i$ denotes the correct class for the $i$ th sample.

## 5 CLEANN HARDWARE

In the following, we delineate the hardware architecture of CLEANN components that enable a high throughput and low energy execution.

► **Matrix-Vector Multiplication Core.** Many of the fundamental operations performed in CLEANN include matrix-vector multiplication (MVM). In particular, the outlier detection module requires two MVMs to calculate the distance function shown in Eq. (5). Additionally, the dimensionality reduction and restoring components in the feature analyzer are realized using MVMs with weight matrices $W\ \ \mathbb{R}^{l\ \ r}$ and $W\ \ \mathbb{R}^{r\ \ l}$, respectively, where $l$ is the dimensionality of the input and $r$ is the SVD rank. We devise an FPGA core for MVM and vector addition, realized using DSP blocks with Multiplication Accumulation (MAC) functionality [17, 31]. Figure 8 presents the high-level schematic of CLEANN vector-matrix multiplication.

We provide two levels of parallelism in our design controlled by parameters P and SIMD in figure (8). This approach allows our design to achieve maximum resource utilization and throughput on various FPGA platforms. The weight matrix is divided into subsets of length P and fed into parallel processing elements (PEs). These subsets are read from DRAM using a Ping-Pong weight buffer to overlap memory reads with PE computations. At each cycle, PEs perform partial dot-product on the fetched weight and input partitions of length *SIMD*; the same input partition is shared across all PEs. We devise a tree-based reduction module and an accumulator to enable summation of partial dot-product outputs.

► **Sparse Recovery Core.** The sparse recovery module performs OMP to reconstruct input signals. We provide a reconfigurable and scalable OMP core on FPGA to accelerate sparse recovery. OMP relies on sequential execution of three steps: (1) The dictionary column with the maximum dot-product with the current residual vector is selected. (2) An LS optimization step generates the sparse

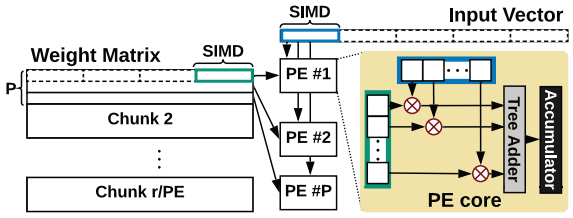


**Figure 8: Schematic representation of CLEANN MVM core with its internal parallelization levels.**

representation of the current residual vector with the columns of the dictionary selected so far. (3) The residual is updated based on the new sparse representation and the selected dictionary columns.

We utilize CLEANN MVM core to implement the first step above. For the second step, we implement the LS optimization using a *QR* factorization of the dictionary matrix. We leverage the modified Gram Schmidt (MGS) method [13] to perform the factorization. Since the dictionary matrix expands by one column each iteration, it is not necessary to recompute the $Q$ and $R$ matrices very time. Instead, we iteratively form the $Q$ and $R$ matrices as outlined in Algorithm 2. Using the acquired new column for the $Q$ matrix, the residual update step takes the following form:

$$r_{i\ 1}\quad r_i\quad Q_i\ Q_i\ ^T r_i \qquad (15)$$

Due to the low memory footprint of CLEANN components, we store all required data in the available on-chip Block RAMs. By eliminating the overhead of external memory access, CLEANN enjoys a low latency and high power efficiency.

**Algorithm 2** QR factorization with MGS

**Inputs:** New dictionary column $D_i$, $Q_{i\ 1}$, $R_{i\ 1}$.
**Output:** $Q_i$, $R_i$.

1: $R_i\quad \begin{bmatrix} R_{i\ 1} & 0 \\ 0 & 0 \end{bmatrix}, \epsilon_i\quad D_i$
2: **for** $j = 1,\ \ ,\ i\ \ 1$ **do**
3: $\quad R_{i\ j,i}\quad Q_{i\ 1\ j}\ ^T \epsilon_i$
4: $\quad \epsilon_i\quad \epsilon_i\quad R_{i\ j,i}\ Q_{i\ 1\ j}$
5: $R_{i\ i,i} = \epsilon_{i\ 2}$
6: $Q_i = Q_{i\ 1}\quad \epsilon_i\ R_{i\ i,i}$

## 6 EXPERIMENTS

We evaluate CLEANN on three visual classification datasets of varying size and complexity, namely, MNIST [19] for handwritten digits, GTSRB [32] for road signs, and VGGFace [28] for face data. The number of classes for each dataset is 10, 43, and 2622, respectively. We corroborate CLEANN effectiveness against variations of two available state-of-the-art Neural Trojan attacks. In what follows, we provide detailed performance analysis and comparisons with prior art. We further demonstrate CLEANN accelerated execution on embedded hardware.

### 6.1 Attack Configuration

Throughout the experiments, we consider input-agnostic Trojans where adding the trigger to any image causes misclassification to the attack target class. Table 1 summarizes the evaluated benchmarks along with their corresponding Trojan attacks and triggers.

► **BadNets.** We implement the BadNets [14] attack with various triggers as an example of a realistic physical attacks. The injected Trojans include a white square and a Firefox logo placed at the

bottom right corner of the input image. We embed the backdoor by injecting 10% poisoned data samples during training.

► **TrojanNN.** We evaluate CLEANN against TrojanNN [23] as a digital attack with complex triggers. The attack is implemented using the open-source models shared by TrojanNN authors[4]. We perform experiments with two variants of TrojanNN triggers, namely, square and watermark, crafted for the VGGFace dataset.

**Table 1: Evaluated datasets and attack algorithms.**

| Dataset | Input Size | Architecture | Attack | Trigger |
|---|---|---|---|---|
| **MNIST** | 1x28x28 | 2CONV, 2MP, 2FC | BadNets | square |
| **GTSRB** | 3x32x32 | 6CONV, 3MP, 2FC | BadNets | square<br>Firefox |
| **VGGFace** | 3x224x224 | 13CONV, 5MP, 3FC | TrojanNN | square<br>watermark |

## 6.2 Detection Performance

We apply CLEANN Trojan mitigation at the input and latent space of infected DNNs. To create the defense, we separate 500, 430, and 2622 clean samples from MNIST, GTSRB, and VGGFace test sets, respectively. The aforementioned size for the benign dataset corresponds to 1% of the training data size for MNIST and GTSRB and 0.1% VGGFace training data. Such low data size requirements provide a competitive advantage for CLEANN defense in real-world scenarios. We summarize other defense parameters for our evaluated benchmarks in Table 2. These parameters are selected to maintain a high classification accuracy over the benign data.

**Table 2: Parameters of CLEANN modules for various datasets. $P$: DCT windows size, $l$: feature size for sparse recovery, $m$: number of dictionary columns for sparse recovery, $\lambda$: sparsity parameter in sparse recovery, $\epsilon^2$: distance threshold for outlier detection.**

| Dataset | Trigger | Input Analyzer | | | | | Feature Analyzer | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $P$ | $l$ | $m$ | $\lambda$ | $\epsilon^2$ | $l$ | $m$ | $\lambda$ | $\epsilon^2$ |
| **MNIST** | Square | 4 | 48 | 1000 | 5 | $5\ 10^{4}$ | 279 | 500 | 80 | $2\ 10^{3}$ |
| **GTSRB** | Square | 4 | 48 | 1000 | 5 | $5\ 10^{4}$ | 85 | 420 | 80 | $3\ 10^{3}$ |
| | FireFox | | | | | | | | 50 | $1\ 10^{2}$ |
| **VGGFace** | Square | 8 | 192 | 1000 | 5 | $5\ 10^{4}$ | 520 | 2622 | 80 | $1\ 10^{4}$ |
| | Watermark | | | | | $8\ 10^{4}$ | | | 80 | $1\ 10^{4}$ |

We evaluate CLEANN Trojan resiliency on physical and digital attacks in Table 3. Specifically, under "Defended Model", we evaluate the drop in clean data accuracy (ACC ), the attack success rate (ASR), and Trojan ground-truth label recovery (TGR). In addition to our results, we include prior art performance in terms of the above-mentioned criteria. On MNIST, CLEANN achieves 0% ASR, with only 0.1% drop in clean data accuracy, outperforming the prior art. For GTSRB, CLEANN achieves an ASR of 0% and a lower drop of accuracy compared to all prior work, except for Deep Inspect, which suffers from a much higher ASR of 8.8%.

On digital attacks, CLEANN achieves 0.0% ASR with only 0.8% and 2.0% degradation of accuracy for square and watermark shapes. The watermark trigger covers a large area of the input image, obstructing the critical features. As such, while CLEANN detects the Trojan with high success, it shows a lower TGR compared to our other triggers. Note that Neural Cleanse and Deep Inspect perform DNN training on synthetic datasets achieved with model inversion [11]. As a result, their post-defense accuracy is not directly comparable with CLEANN, which does not perform DNN retraining.

[4]https://github.com/PurduePAML/TrojanNN

We emphasize that while such retraining contributes to accuracy, it may not be feasible in real-world applications.

**Table 3: Evaluation of CLEANN on various physical and digital attacks. Comparisons with state-of-the-art prior works, i.e., Neural Cleanse(NC) [36], Deep Inspect (DI) [5], Februus [8], and SentiNet [6] are provided where applicable.**

| Dataset | Trigger | Work | Retrain | Infected Model | | Defended Model | | |
|---|---|---|---|---|---|---|---|---|
| | | | | ACC-C | ASR | ACC | ASR | TGR |
| **MNIST (Physical Attack)** | Square 4 4 | NC | yes | 98.5 | 99.9 | 0.8 | 0.6 | NA |
| | | DI | yes | 98.8 | 100.0 | 0.7 | 8.8 | NA |
| | | CLEANN | no | 99.3 | 100.0 | **0.1** | **0.0** | **98.7** |
| **GTSRB (Physical Attack)** | Square 4 4 | NC | yes | 96.5 | 97.4 | 3.6 | 0.1 | NA |
| | | DI | yes | 96.1 | 98.9 | **-1.0** | 8.8 | NA |
| | | Februus | yes | 96.8 | 100 | 1.2 | **0.0** | **96.5** |
| | | CLEANN | no | 96.5 | 99.4 | 0.0 | **0.0** | 94.7 |
| | Firefox 6 6 | CLEANN | no | 92.6 | 99.8 | 0.4 | 1.7 | 83.5 |
| **VGGFACE (Digital Attack)** | Square 59 59 | NC | yes | 70.8 | 99.9 | **-8.4** | 3.7 | NA |
| | | DI | yes | 70.8 | 99.9 | 0.7 | 9.7 | NA |
| | | SentiNet | no | NA | 96.5 | NA | 0.8 | NA |
| | | CLEANN | no | 74.9 | 93.52 | 0.8 | **0.0** | **70.1** |
| | Watermark | NC | yes | 71.4 | 97.60 | **-7.4** | **0.0** | NA |
| | | DI | yes | 71.4 | 97.60 | 0.5 | 8.9 | NA |
| | | CLEANN | no | 74.9 | 58.6 | 2.0 | **0.0** | **41.38** |

Februus performs GAN training. SentiNet only reports results on LFW [16] dataset.

► **Sensitivity to Trigger Size.** We perform experiments on the GTSRB dataset with a square Trojan trigger and change the trigger size such that it covers between 0.4% to 14% of the input image area. The size range is chosen to ensure that the corresponding triggers are viable in real settings and provide a high ASR. We summarize the obtained results in Figure 9. CLEANN significantly reduces the ASR while enabling recovery of ground-truth labels with a high accuracy across all trigger sizes. This is expected since CLEANN does not rely on the trigger size to construct the defense. For average sized Trojans, CLEANN successfully detects the existence of triggers and reduces the ASR to less than 1%. For larger trigger sizes, the TGR is relatively lower since the Trojan occludes the main objects in the image.

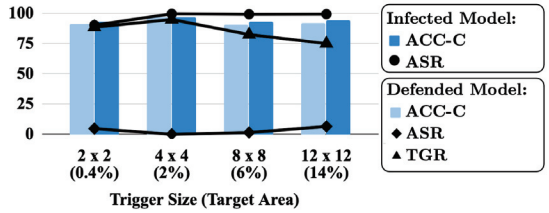


**Figure 9: Analysis of CLEANN sensitivity to Trojan trigger size.**

► **Offline Preprocessing Overhead.** The preparation of CLEANN defensive modules consists of the following steps:

DCT extraction and dictionary leaning on benign inputs.
Computing $\mu$ and $\Sigma$ in Eq. (4) for input outlier detection.
Computing SVD and dictionary learning at latent feature maps.
Computing $\mu$ and $\Sigma$ for latent outlier detection.

In practice, the above computation incurs negligible runtime compared to DNN training. We implement the above steps in PyTorch and measure the runtime on an NVIDIA TITAN Xp GPU. For our GTSRB benchmark, the above operations require 0.06, 0.19, 10.47, and 0.1 seconds, respectively. The defense construction time is therefore 11 seconds which is 1.8% of the time required to train the victim DNN on this benchmark. For the more complex VGGFace dataset, the above operations require 1.05, 0.54, 48.3, and 1.2 seconds, respectively, resulting in a total of 51 seconds for defense preparation.

### 6.3 Hardware performance

We implement the proposed Trojan defense strategy on various hardware platforms and compare the performance of CLEANN components. The evaluated platforms include server-grade CPUs and GPUs, embedded CPUs and GPUs, and FPGA. We base our comparisons on performance-per-Watt defined as the throughput over the total power consumed by the system. This measure effectively encapsulates two major performance metrics for embedded applications. Throughout this section, we will target our study on the GTSRB benchmark but similar trends are observed for other datasets.

► **Performance on General Purpose Hardware.** We provide an optimized software library for CLEANN defense components in Python. In order to benefit from highly optimized backend compilers for tensor operations on CPU and GPU, our codes are developed on top of the PyTorch deep learning library. Our provided software library can be readily instantiated within PyTorch API to enable simultaneous DNN execution and Trojan defense. We implement our defense pipeline on the *Jetson TX2* embedded development board running in CPU-GPU and CPU-only modes. We further run the defense on a server-grade *Intel Xeon E5* CPU and an *NVIDIA TITAN Xp* GPU. The overall achieved defense throughput with a batch size of 1 ranges from 11 fps on the embedded CPU up to 28 fps on the server GPU.

Figure 10 illustrates the runtime breakdown for various components of CLEANN running on each platform. Here, the sparse recovery and outlier detection modules are abbreviated as `SR` and `OLD` and the prefixes `D-` and `F-` correspond to the DCT and feature analyzers, respectively. The experiments are performed using a batch size of 1 to resemble real-world applications and runtimes are averaged across 100 runs. For each platform, we normalize the runtime of each component by the total defense execution time for one sample. As seen, the bulk of defense runtime belongs to the sparse recovery module. This is due to the inherently sequential nature of the OMP algorithm performed inside this module. CPU and GPU platforms are designed to excel in massively parallel operations while this does not hold for OMP. Such behavior further motivates us to design specialized hardware to accelerate the execution of CLEANN components on FPGA.

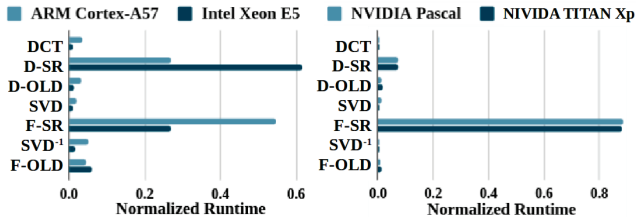


**Figure 10: Latency breakdown of CLEANN components running on embedded and high-end CPUs (left) and GPUs (right).**

► **Performance on Customized Accelerator.** We implement CLEANN components on FPGA using the developed sparse recovery and `MVM` cores as the basic blocks. The design is developed in Vivado High-Level Synthesis and synthesized in Vivado Design Suite for the *Xilinx UltraScale VCU108* board. Power consumption is estimated during synthesis with Vivado Design Suite. Finally, a comprehensive timing and resource utilization analysis is performed. To maximize throughput, we tuned the parallelism factors in the `MVM` modules to the highest value such that the design fits within the available resources.

Figure 11 demonstrates the breakdown of execution cycles for CLEANN components. As seen, the sequential execution of the sparse recovery core accounts for the majority of computation cycles. Our FPGA-based sparse recovery core enjoys up to 10 and 18 faster execution, respectively, compared to their CPU and GPU counterparts. This is enabled by pipelined execution, fine-grained optimizations to data access patterns, and parallel computation.

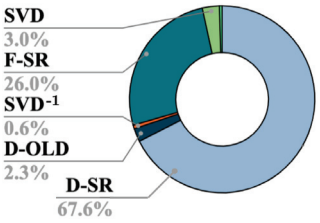


**Figure 11: Cycle-count breakdown for execution of CLEANN components implemented on FPGA.**

We compare the performance-per-Watt and throughput of CLEANN on different hardware platforms in Figure 12. The performance-per-watt numbers are normalized by *TITAN Xp* and the throughput numbers are normalized by *ARM Cortex-A57*. As seen, the power-efficient implementation of CLEANN on FPGA not only enjoys a high throughput, but it also significantly increases performance-per-watt compared to commodity hardware. Note that due to the light-weight nature of CLEANN defense strategy, the server-grade GPU performs poorly in terms of performance-per-watt compared to other platforms due to under-utilization and excessive power consumption.

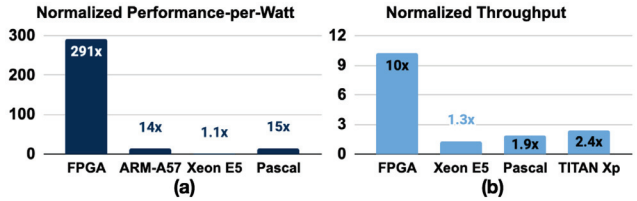


**Figure 12: (a) Performance-per-Watt and (b) throughput across hardware platforms. Reported values for performance per-watt are normalized by *TITAN Xp* and throughput values are normalized by *ARM Cortex-A57*.**

## 7 CONCLUSION

This paper presents CLEANN, an end-to-end framework for online accelerated defense against Neural Trojans. The proposed defense strategy offers several intriguing properties: (1) The defense construction is entirely unsupervised and sample efficient, i.e., it does not require any labeled data and is established using a small clean dataset. (2) It is the first work to recover the original label of Trojan data without need for any fine-tuning or model training. (3) CLEANN provides theoretical bounds on the false positive rate. (4) The framework is devised based on algorithm/hardware co-design to enable accurate Trojan detection on resource-constrained embedded devices. We consider a challenging threat model where the attacker can use Trojan triggers with arbitrary shapes and patterns while no knowledge about the attack is available to the client. CLEANN light-weight defense and realistic threat model makes it an attractive candidate for practical deployment. Our extensive evaluations corroborate CLEANN's competitive advantage in terms of attack resiliency and execution overhead.

## 8 ACKNOWLEDGMENT

This work was supported in part by ARO (W911NF1910317).

# REFERENCES


[1] Michal Aharon and Michael Elad. 2008. Sparse and redundant modeling of image content using an image-signature-dictionary. SIAM Journal on Imaging Sciences 1, 3 (2008), 228–247.

[2] Michal Aharon, Michael Elad, and Alfred Bruckstein. 2006. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. IEEE Transactions on signal processing 54, 11 (2006), 4311–4322.

[3] Moses Charikar, Jacob Steinhardt, and Gregory Valiant. 2017. Learning from untrusted data. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. 47–60.

[4] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. arXiv preprint arXiv:1811.03728 (2018).

[5] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. 2019. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press. 4658–4664.

[6] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. 2018. Sentinet: Detecting physical attacks against deep learning systems. arXiv preprint arXiv:1812.00292 (2018).

[7] Geoffrey M Davis, Stephane G Mallat, and Zhifeng Zhang. 1994. Adaptive time-frequency decompositions. Optical engineering 33, 7 (1994), 2183–2192.

[8] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. 2019. Februus: Input purification defence against trojan attacks on deep neural network systems. (2019).

[9] David L Donoho and Michael Elad. 2003. Optimally sparse representation in general (nonorthogonal) dictionaries via L1 minimization. Proceedings of the National Academy of Sciences 100, 5 (2003), 2197–2202.

[10] Kjersti Engan, Sven Ole Aase, and J Hakon Husoy. 1999. Method of optimal directions for frame design. In 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258), Vol. 5. IEEE, 2443–2446.

[11] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 1322–1333.

[12] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. 2019. Strip: A defence against trojan attacks on deep neural networks. In Proceedings of the 35th Annual Computer Security Applications Conference. 113–125.

[13] Gene H Golub and Charles F Van Loan. 2012. Matrix computations. Vol. 3. John Hopkins University Press.

[14] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017).

[15] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. 2019. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. arXiv preprint arXiv:1908.01763 (2019).

[16] Gary B. Huang, Marwan Mattar, Honglak Lee, and Erik Learned-Miller. 2012. Learning to Align from Scratch. In NIPS.

[17] Shehzeen Hussain, Mojan Javaheripi, Paarth Neekhara, Ryan Kastner, and Farinaz Koushanfar. 2019. FastWave: Accelerating Autoregressive Convolutional Neural Networks on FPGA. In 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 1–8.

[18] Mojan Javaheripi, Bita Darvish Rouhani, and Farinaz Koushanfar. 2019. SWNet: Small-World Neural Networks and Rapid Convergence. arXiv preprint arXiv:1904.04862 (2019).

[19] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 1998. The MNIST database of handwritten digits. (1998).

[20] Chang Liu, Bo Li, Yevgeniy Vorobeychik, and Alina Oprea. 2017. Robust linear regression against training data poisoning. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security. 91–102.

[21] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, 273–294.

[22] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 1265–1282.

[23] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2017. Trojaning attack on neural networks. (2017).

[24] Yuntao Liu, Yang Xie, and Ankur Srivastava. 2017. Neural trojans. In 2017 IEEE International Conference on Computer Design (ICCD). IEEE, 45–48.

[25] Shiqing Ma and Yingqi Liu. 2019. Nic: Detecting adversarial samples with neural network invariant checking. In Proceedings of the 26th Network and Distributed System Security Symposium (NDSS 2019).

[26] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2019. Mlperf training benchmark. arXiv preprint arXiv:1910.01500 (2019).

[27] Azalia Mirhoseini, Eva L Dyer, Ebrahim M Songhori, Richard Baraniuk, and Farinaz Koushanfar. 2017. RankMap: A Framework for Distributed Learning From Dense Data Sets. IEEE transactions on neural networks and learning systems 29, 7 (2017), 2717–2730.

[28] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. 2015. Deep face recognition. (2015).

[29] Bita Darvish Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. 2018. Deepfense: Online accelerated defense against adversarial deep learning. In 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 1–8.

[30] David Salomon. 2004. Data compression: the complete reference. Springer Science & Business Media.

[31] Mohammad Samragh, mojan javaheripi, and Farinaz Koushanfar. [n. d.]. EncoDeep: Realizing Bit-Flexible Encoding for Deep Neural Networks. ACM Transactions on Embedded Computing Systems (TECS) ([n. d.]).

[32] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural Networks 0 (2012), –. https://doi.org/10.1016/j.neunet.2012.02.016

[33] Bartolomeo Stellato, Bart PG Van Parys, and Paul J Goulart. 2017. Multivariate Chebyshev inequality with estimated mean and variance. The American Statistician 71, 2 (2017), 123–127.

[34] Brandon Tran, Jerry Li, and Aleksander Madry. 2018. Spectral signatures in backdoor attacks. In Advances in Neural Information Processing Systems. 8000–8010.

[35] Daniel Vainsencher, Shie Mannor, and Alfred M Bruckstein. 2011. The sample complexity of dictionary learning. Journal of Machine Learning Research 12, Nov (2011), 3259–3281.

[36] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 707–723.