

# Robust Machine Learning Systems: Challenges, Current Trends, Perspectives, and the Road Ahead

**Muhammad Shafique and Mahum Naseer**

Technische Universität Wien (TU Wien)

**Onur Mutlu and Lois Orosa**

ETH Zürich

**Theocharis Theocharides and**

**Christos Kyrkou**

University of Cyprus

**Jungwook Choi**

Hanyang University

*Editor's note:*

Currently, machine learning (ML) techniques are at the heart of smart cyber–physical systems (CPSs) and Internet-of-Things (IoT). This article discusses various challenges and probable solutions for security attacks on these ML-inspired hardware and software techniques.

—Partha Pratim Pande, Washington State University

■ **FUELED BY INDEPENDENT** developments in semiconductor technology, computing, communication, control signal generation, sensors, and actuators, the concept of a unified smart cyber–physical system (CPS) has evolved into a ubiquitous paradigm. CPS, as the name implies, links the cyber and the physical environments with smart control. Together with the evolution of Internet-of-Things (IoT), which provides remote access to the CPSs for controlling and monitoring the interconnected computing devices, the standard architecture of a smart CPS comprises three major layers [1]: edge, fog, and cloud. The *edge* of the system is what

connects the system to the physical environment, for instance, the sensors. The *fog* is the central layer where most system computations generally occur. However, to reduce transmission overhead or for data privacy, initial computations may occur at the edge too. The *cloud* is what connects the system to a large-scale cyberspace, which performs extensive processing, storage, and communication between different CPSs.

Improving the decision-making, monitoring, and control capabilities across different CPS/IoT layers is critical for emerging applications. As the complexity, volume, and rate of data produced by IoT with many smart CPSs are increasing, machine learning (ML) has emerged as a dominant paradigm for analytics, decision-making, perception, and understanding. Consequently, reliability and security vulnerabilities of ML systems can have cascading effects on smart CPS applications and critically impact ML operation across all layers.

The most recent developments in ML, especially in deep learning, evolved from the concept of a single-layer neural network (NN), the perceptron [2],

Digital Object Identifier 10.1109/MDAT.2020.2971217

Date of publication: 3 February 2020; date of current version: 20 April 2020.

to the multilayer perceptron (MLP) [2] and the current intricate multilayer deep NNs (DNNs) [3], [4], with the objective of approaching and even exceeding human decision-making capabilities for a certain set of tasks. Due to their effectiveness at handling large amounts of data, learning (and sometimes relearning [5]) input characteristics, and demonstrating high accuracy during inference of unseen inputs, ML systems have proliferated to numerous real-world applications. These include object detection [6], face recognition [7], speech recognition [8], spam filtering [9], malware detection [10], smart grids [11], and even safety-critical applications such as autonomous driving [12], intelligent transportation [13], and healthcare [14], [15], where errors may lead to catastrophic results.

Despite their high inference accuracy in practical applications, ML systems are highly vulnerable to security and reliability threats at both the cloud and the edge. Poisoning the training data (e.g., by inserting random or crafted noise to the data) with incorrectly labeled inputs, inserting malicious components into the system hardware, polluting inputs with imperceptible noise during inference (i.e., during the run-time operation of a system), and monitoring system-side channels to deduce the underlying model are some of the ways in which an attacker can breach the security of an ML system. Even in the absence of an explicit attacker, process variation during hardware fabrication, memory errors, and environmental conditions around the system during training and inference can compromise the reliability of an ML system. Approaches to defend ML systems against these concerns exist, but each approach has its own limitations. Figure 1 summarizes both the security and reliability threats that can affect the accuracy of ML systems and their respective countermeasures.

In this article, we aim to provide a comprehensive overview of vulnerabilities that affect modern ML systems, survey state-of-the-art attacks and defense mechanisms, describe different solution directions and challenges, and identify potential promising avenues to research.

To ease reading, we provide the list of the acronyms used in this article in Table 1 and the rest of this article is organized as shown in Figure 2.

## ML: Concepts and terminology

An ML system, like any other traditional system, takes in the input(s) and generates the corresponding output(s). However, unlike traditional systems, the ML system is capable of learning via input features and using the learned features in decision-making, which provide ML systems with the ability to perform tasks that are very challenging to perform using traditional systems.

NNs are often involved in the main decision-making of many modern ML systems. An NN comprises an *input layer* that connects the external environment to the ML system, an *output layer* that outputs a decision, and *hidden layer(s)* sandwiched between the input and output layers. State-of-the-art ML systems commonly use DNNs with two or more hidden layers. Each layer comprises *neurons* or *nodes* which connect to other neurons in the corresponding layers via a nonlinear *activation* function. Each neuron has its associated parameters, i.e., weight, bias, and/or filter coefficient. A detailed overview of NNs can be found in [16] and [17].

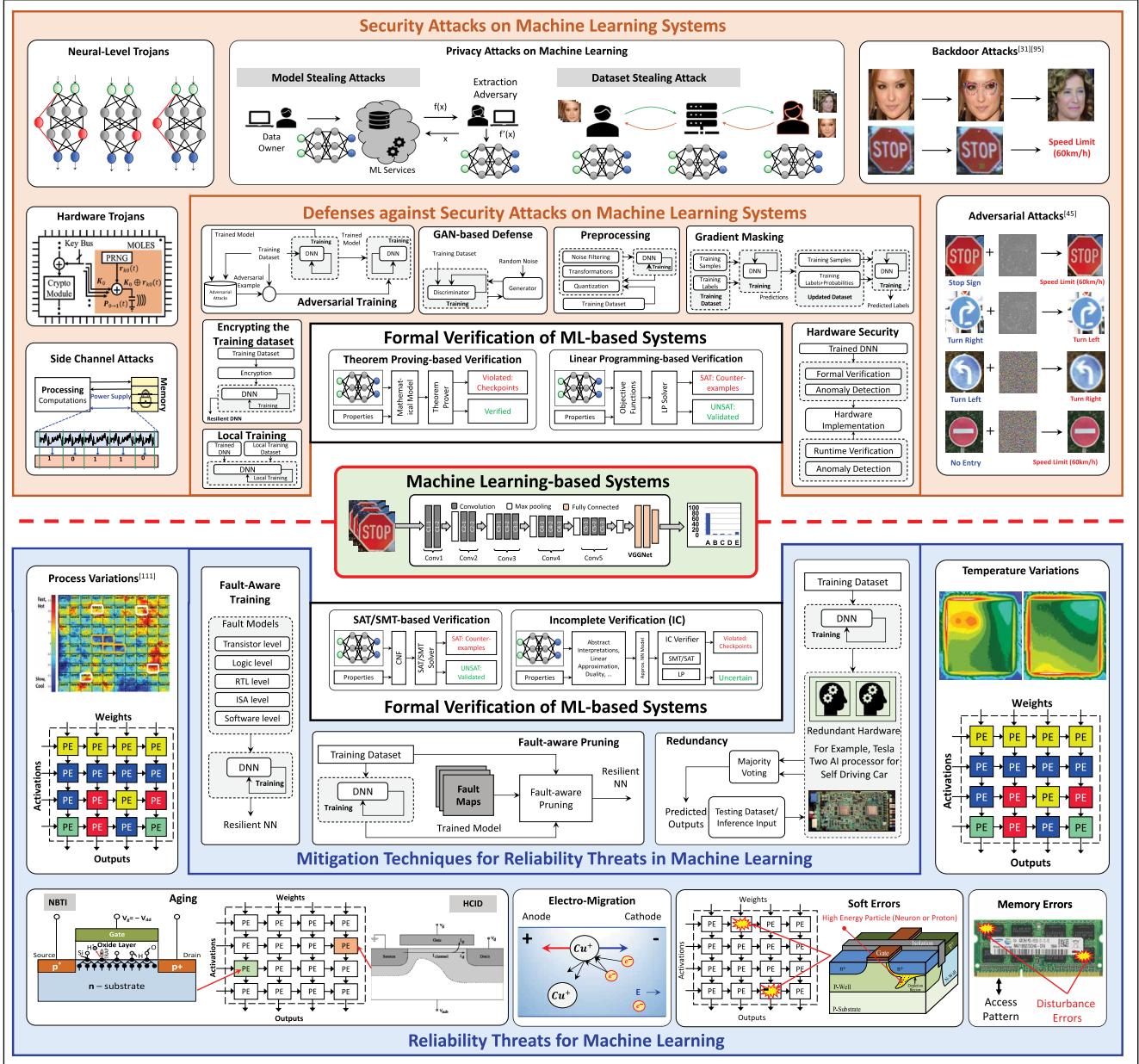
### Neural network taxonomy

If the input propagates through the network in only one direction, the network is said to be *feedforward*. If there are feedback loops in the network, the network is called a recurrent NN (RNN) [18]. Long short-term memories (LSTMs) [19] are a branch of recurrent networks that retain information for a long duration, which makes them well suited for time-series prediction. When every neuron in one layer is connected to “all” neurons in the preceding layer, the network is said to be *fully connected*; otherwise, the network is *sparse*.

Since their advent, NNs have progressively improved over three generations (Figure 3). The details of the NN types of each generation can be found in the Appendix.

*First generation of NNs:* The first generation of NNs [20] is comprised of single-layer and MLPs [21]. MLPs are generally made up of multiple fully connected layers connected with thresholding activations.

*Second generation of NNs:* To reduce the number of parameters in a network, this generation of NN introduces convolutional NNs (CNNs), which make use of convolutional layers comprising of convolutional filters to extract important features from the input, while providing a certain degree of shift



**Figure 1. Overview of threats and challenges associated with ML-based systems: reliability threats and corresponding mitigation techniques (bottom), and security attacks and corresponding defenses (top).**

invariance to the network [22]. A convolutional layer typically uses continuous nonlinear activations, and it is often followed by a pooling layer. Pooling layers reduce the network parameters even further by retaining only the most important features from the preceding layer, which leads to information loss. NNs of this generation are increasingly being used in practical ML systems.

A relatively new approach to solve the problem of information loss in CNNs is the use of capsule networks (CapsNets) [23]. CapsNets have hidden layers comprised of interconnected vectors that have input features and input probabilities, which allow these networks to learn spatial correlations between input features. As a result, CapsNets are able to infer high-level entities quite similarly to human perception.

Another interesting approach toward NNs is the generative adversarial networks (GANs) [24]. These networks make use of the simultaneous interplay between a generator and a discriminator, where the generator produces realistic synthetic inputs, while the discriminator learns to differentiate between the real and synthetic inputs. This enables the NNs to generate synthetic outputs that are very difficult to distinguish from the real ones.

*Third generation of NNs:* This generation of NNs makes use of spiking NNs (SNNs) [25] in an attempt to emulate human brain-like functioning. Unlike the networks discussed earlier, which consider the normalized firing frequency of neurons, SNNs use spike trains to mimic the spatiotemporal characteristics of the biological neurons.

#### Neural network design cycle

Figure 4 provides an overview of the NN-based ML design cycle, which can be categorized in the *training* and *inference* stages. Training is typically performed at the cloud, whereas inference is typically performed at the edge in real-world smart CPSs (e.g., autonomous vehicles and wearable healthcare devices). In certain IoT/CPS systems, which are not constrained with resources or real timeliness, inference may also be performed at the fog or cloud (e.g., predictions on social networks and large-scale hospital data).

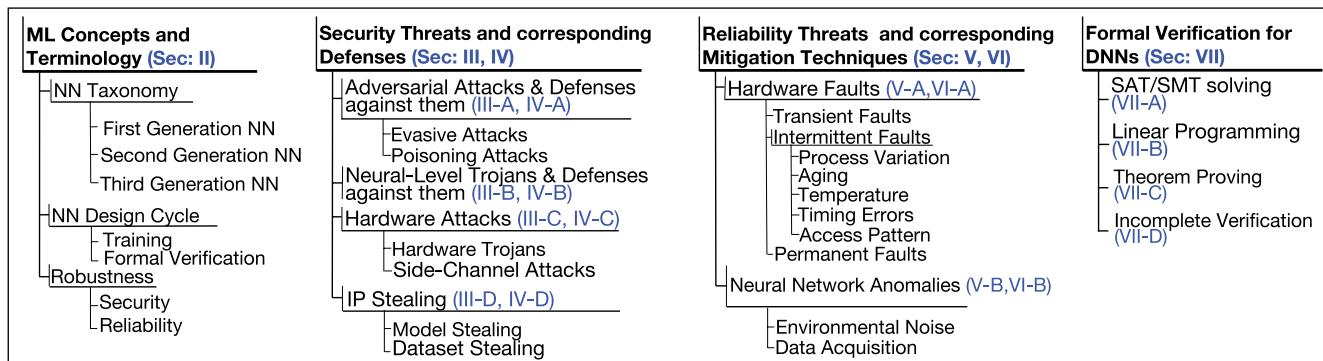
*Training:* Before deploying the NN into an ML system, the NN must be trained. *Training* is a resource-intensive process, generally carried out by third-party cloud servers, which involves the use of a *training data set* to find suitable values for the network parameters. Training is composed of a forward pass and a backward pass. The forward pass calculates

**Table 1. List of acronyms used in this survey.**

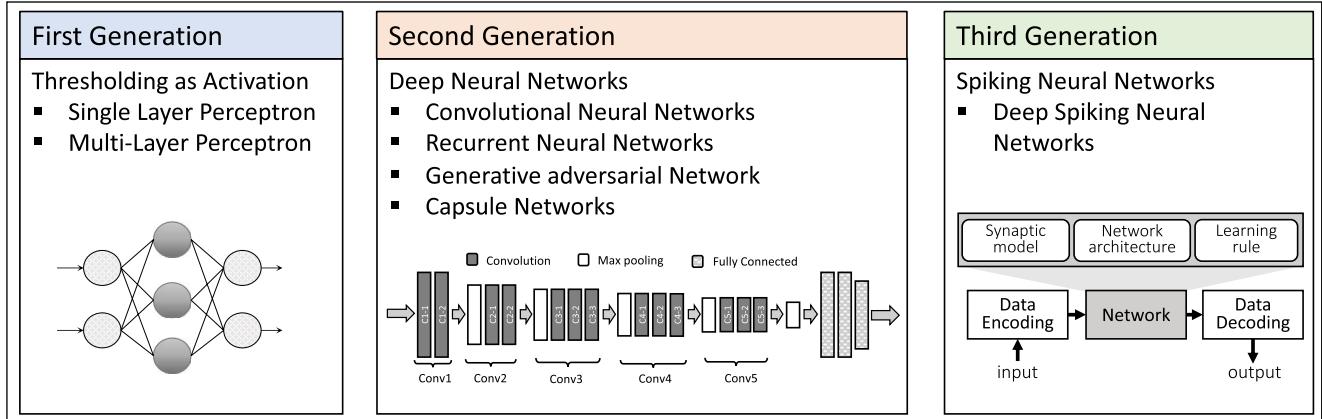
Terminology	Acronym
Artificial Intelligence	AI
Binarized Neural Network	BNN
Capsule Network	CapsNet
Conjunctive Normal Form	CNF
Counter-Example Guided Abstraction Refinement	CEGAR
Convolutional Neural Network	CNN
Cyber-Physical System	CPS
Deep Neural Network	DNN
Denial-of-Service	DoS
Generative Adversarial Network	GAN
Internet-of-Things	IoT
Linear Programming	LP
Long Short-Term Memory	LSTM
Machine Learning	ML
Mixed Integer Linear Programming	MILP
Multi-Layer Perceptron	MLP
Multi-Processor System-on-Chip	MPSoC
Neural Network	NN
Recurrent Neural Network	RNN
Satisfiability Modulo Theories	SMT
Satisfiability solving	SAT solving
Spiking Neural Network	SNN
Very Large Scale Integration	VLSI

the predicted output values by propagating inputs through the network, using the current parameter values. The backward pass updates the network parameters, while minimizing the loss function associated with correct and predicted output values. This process (i.e., a forward pass and a backward pass), when repeated once for all the samples in the training data set, is called an *epoch*. The overall training process of an NN involves several *epochs*.

At the end of each *epoch*, the accuracy of the network is analyzed for some unseen data, which is not part of the training data set, i.e., the validation data set. The result of this testing can be used to fine tune the network hyper-parameters, like the number of layers, and select the best trained model. The training process then resumes and the network



**Figure 2. Organization of this article.**



**Figure 3. Summary of NN models proposed over time.**

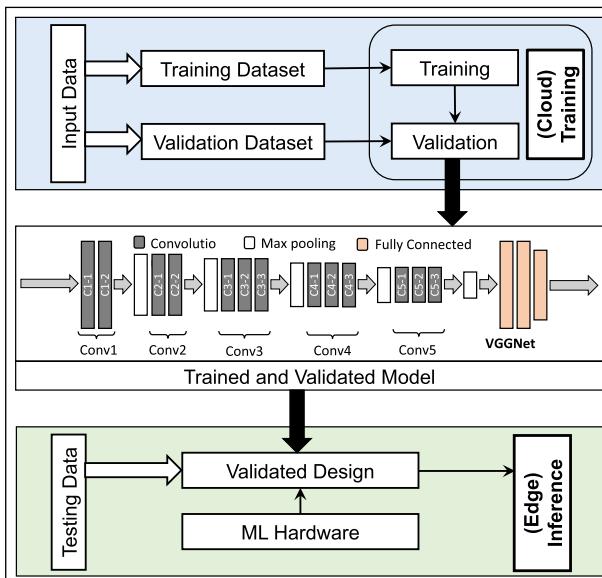
parameters are again updated using the training data set until either the process reaches the maximum number of epochs (or cycles), or the network reaches the desired level of accuracy with the validation data set.

The most common way to check the final inference accuracy of a trained network is to use a *testing data set*. If the trained network is able to classify testing inputs correctly for more than the desired number of testing inputs, the network is considered suitable for deployment into a practical system. However, a DNN might misclassify an input that is perceptually similar to another input correctly identified by the same DNN [26].

To ensure the security, reliability, and safety of ML systems for safety-critical applications, e.g., autonomous vehicles and smart healthcare, it is imperative to develop a framework to analyze and verify these critical misclassifications. An orthogonal research direction, therefore, is to use formal verification for ascertaining the dependability of the trained DNN.

Although an established research domain [27], [28], *formal verification* started gaining interest in the ML research community only since the last decade. Formal verification is an approach to check the correct behavior of a system on the basis of sound mathematical reasoning. Unlike testing, verification provides guarantees regarding system accuracy, independent of “specific” system inputs. Hence, as shown in Figure 5, the guarantees provided by verification are valid for the entire (infinite) input domain, whereas those provided by testing are limited only to the (finite) tested data. In terms of ML systems, due to the complexity of the underlying system, the objective of verification is usually to verify the correctness of the network for bounded input regions, as demonstrated in Figure 6, rather than for the entire input domain.

*Inference:* A trained and tested/verified NN can be deployed in a real-world ML system. At this stage, the NN performs classification/decision-making using actual, previously unseen, data (i.e., in real time). ML inference is typically carried out at the edge of the IoT/CPSs, thereby exposing the system to numerous security and reliability concerns during the operations under varying scenarios and harsh environmental conditions.



**Figure 4. Design cycle of an NN-based ML system.**

## Robustness

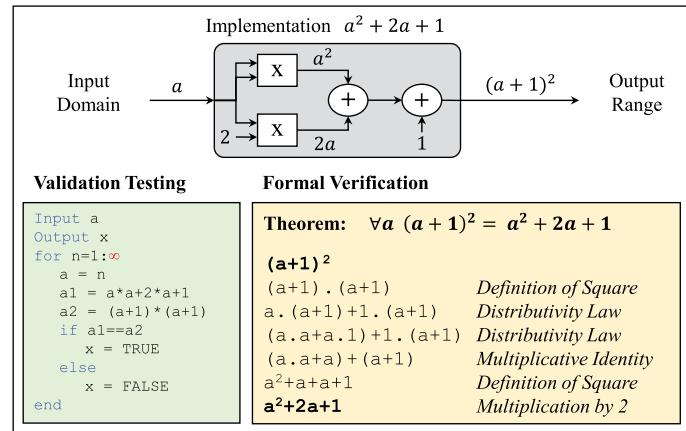
A common term associated with the performance of DNNs is robustness. *Robustness* is the DNN property that determines the integrity of the network under varying operating conditions, and the accuracy of DNN outputs in the presence/absence of input or network alterations. This can be divided into two subproperties: security and reliability [29]. The DNN is said to be *secure* against an attack if the attacker cannot steal information [via intellectual property (IP) stealing or side channel attack], engage the system resources [e.g., using hardware intrusion or denial-of-service (DoS) attack], modify the network parameters (e.g., by inserting hardware or neural-level Trojans), or render an incorrect input to the DNN (e.g., using an adversarial attack). In the case of reliability, there is no explicit attacker. The network is said to be *reliable* if it does not display any changes to its output, parameters, or behavior, due to the changes in environmental conditions, during fabrication and deployment.

## Security vulnerabilities of ML systems

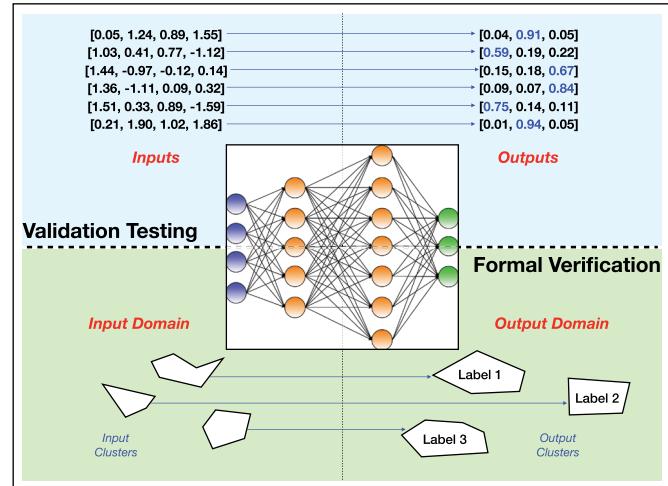
As hinted in the previous section, despite being highly sophisticated in learning and decision-making, ML systems are very vulnerable to attacks. Depending on the type and intensity of the attack(s), and the application where the system is deployed, these ML vulnerabilities can lead to slight discrepancies in the result, or can lead to lethal consequences in a safety-critical application [6]. This section describes the most common security issues in ML systems and DNNs at the cloud and the edge, as summarized in Table 2).

### Adversarial attack

Since their discovery, adversarial attacks [26] have been a widely studied DNN security threat



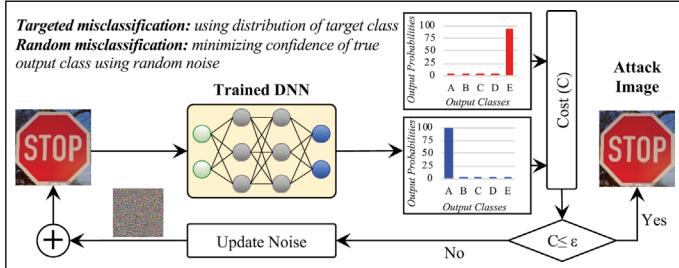
**Figure 5. Comparison between testing and verification for a small hypothetical system: ensuring behavioral correctness of the system for all possible inputs is not always feasible with testing.**



**Figure 6. Comparison between testing and verification for an NN-based system. Verification is intended to determine whether the bounded inputs are reachable to the correct output bounds.**

**Table 2. Summary of the various security threats and their countermeasures for ML-based systems.**

Threat	Insertion Point			Vulnerability	Countermeasures		
	Design Phase		DNN Inference				
	Hardware Design	DNN Training					
Adversarial Attack			✓	Input	Gradient Masking [30], Pre-Processing Filters [31], Adversarial Retraining		
Backdoor Attack		✓		Network Parameters ( $W, b$ )	Pruning [32], Fine Tuning [33]		
Data Poisoning	✓	✓		Input	Encryption [34][35][36], Local Training		
IP Stealing			✓	System Response	Obfuscation, Encryption [37]		
Hardware Trojan	✓			Hardware, System Response	Equivalence Checking [38], Side-Channel analysis [39]		
Side-channel Attack			✓	System Response	Randomness [40][41]		



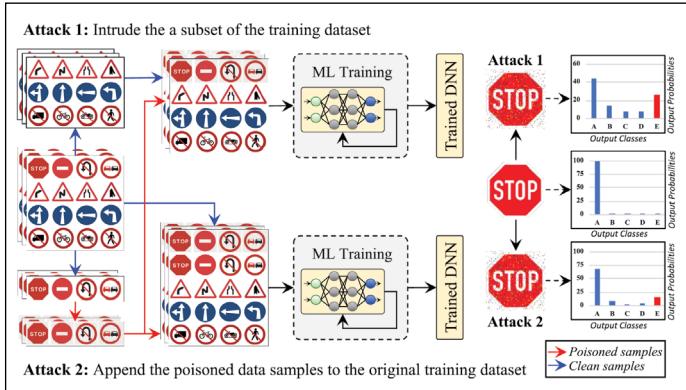
**Figure 7. Adversarial attack on a trained DNN: an adversarial attack can result in the misclassification (either targeted or random) of traffic sign boards, which is a concern in autonomous driving [46].**

[42]–[44]. In an adversarial attack, the known DNN parameters are exploited to minimize the cost function corresponding to noise patterns  $\delta x$ , which, when added to the input  $x$ , can cause *misclassification*, as shown in Figure 7. The noise added is usually imperceptible, making the task of distinguishing between clean and malignant inputs nearly impossible. This can formally be represented as

$$f(x) \neq f(x + \delta x + EN) \text{ s.t. } \delta x \leq \epsilon \quad (1)$$

where EN represents the noise existing in the physical environment even in the absence of an explicit attacker. The adversarial noise can lead to either a random incorrect output class, i.e., an *untargeted attack scenario*, a specific calculated output class, i.e., a *targeted attack*, or simply reduce the confidence of the correct output class [45], i.e., *confidence reduction*.

Adversarial attacks can be categorized as either *evasive* or *poisoning* [47], depending on the access



**Figure 8. Classification accuracy of DNN trained on a poisoned data set.**

of the attacker to the DNN design cycle. In evasive attacks, the attacker has no access to the DNN training process and the training data set. The attack is solely configured during the DNN inference stage, using either input gradients, output probability vectors, or the output decision [48]–[53]. For instance, the fast gradient sign method (FGSM) [42] determines the direction of the loss function via the input gradient, scales down its value, and adds the noise to the input. In the Jacobian saliency map approach (JSMA) [54], the input gradient (Jacobian) is again used, but the objective is to add the noise to a subset of input nodes, sufficient for misclassification. Other works [55], [56] make use of input gradients to propose adversarial attacks. TrlSec [46] improves the imperceptibility of an adversarial attack by introducing a new methodology that uses additional parameters (e.g., correlation coefficient between the target image and the original image, and the structural similarity index) in the DNN training algorithm. Works like [48], [54] make use of output labels to determine attacks in close proximity to the classification boundary.

In poisoning attacks [57], the attacker has access to the training data set/training procedure. The attack is implanted in the DNN during training by feeding the network with malicious training data. Figure 8 shows two examples of poisoning attacks that increase the probability of misclassification of a stop signal (red bars). The data could be poisoned with tailored noise [32], [58], also known as *backdoor attack*, or simply through random noise [45]. Sparsity of the network accounts for the success of poisoning adversarial attacks. Dormant neurons in a trained DNN have weights and biases too small to be of any practical significance to the output calculation. The existence of such neurons signifies that the network has the capacity to learn more. Hence, such networks can be trained on poisoned data (as shown in Figure 9). The DNN behaves correctly for the clean data but exhibits a malignant behavior for the poisoned data. A recent work demonstrates the use of poisoning (with noisy image patches) to either misclassify humans as different objects or completely hide a person from the object detection system [59].

For most of the adversarial attacks, a common inadequacy is to ignore the preprocessing filtering stage in an ML system [31]. The preprocessing stage generally employs different averaging filters, to

smooth out any noise in input. This undermines, if not completely eliminates, the threat of misclassification via adversarial attacks.

### Neural-level Trojans

Another class of attacks, the neural-level Trojans [60], involves the insertion of additional neurons into a pretrained DNN by third-party training servers. The number of extra neurons must be minimized to avoid raising suspicion regarding the DNN model. Conceptually, similar to hardware Trojans in system hardware (discussed later) and backdoor attacks, the additional neurons in neural-level Trojans trigger malicious DNN behavior only when prompted by specific inputs. However, most of these attacks require to retrain the network and use complex internal triggering mechanisms.

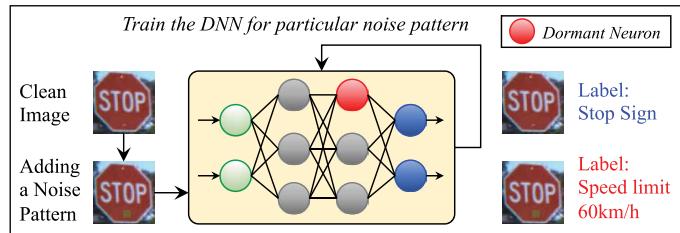
### Hardware attacks

Hardware Trojans [61]–[65] are malicious components implanted into the system hardware, which compromise the security of an ML system. Hardware Trojans can introduce undesired system behavior or be dormant in the normal system operation and be triggered at a specific instance. They may leak system information, thereby aiding IP stealing (discussed later) or simply consume system power and resources.

The attack is usually instigated by an untrusted manufacturer/foundry, at the manufacturing stage of the system lifecycle. The size of the Trojan is usually small, and hence goes unnoticed. Often, the overall number of components on the chip is kept unchanged and the power trace of the Trojan is also minimized [66] to ensure a successful stealthy attack.

Side-channel attacks, as shown in Figure 10, are another type of hardware attack that is crafted using leaking information from the system hardware. Most systems leak information via side channels such as components' power consumption [39]–[41], [67]–[69]. This information can be analyzed and used to: 1) compromise the security and privacy of the system and 2) reverse engineering and steal the model parameters [70], [71].

Analyzing the different side channels of a system enables us to target different parameters of an ML system. For instance, the leaking power traces close to the input of the DNN provide clues regarding the system input, whereas the information regarding execution times provides predictions for the network architecture [40], [41]. However, a common

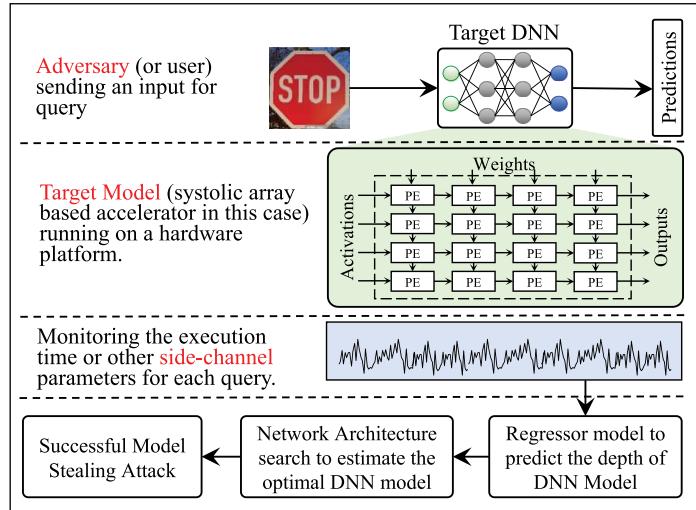


**Figure 9. Effect of a backdoor on DNN accuracy. The dormant neurons (red) learn to associate the backdoor with a targeted misclassification label.**

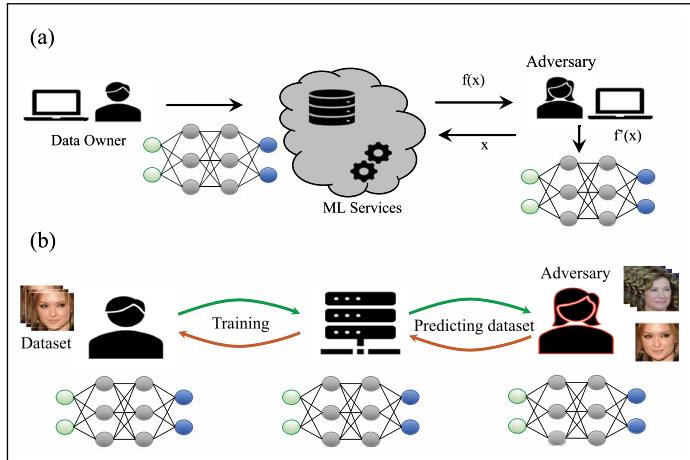
limitation with most side-channel attacks is assuming the absence of noise in the system. Inclusion of noise in the side-channel attack's threat model generates randomness in the leaked information, which reduces the chances of a successful attack.

### IP stealing

Attacks to steal IP are another significant security threat for ML systems. IP stealing involves determining either the underlying *model* of the ML system (model stealing attack), possibly without any access to the description or internal parameters of the system [72], or predicting the *data* the DNN was trained on using the available model description (dataset stealing attack) [73]. Both types of attacks are shown in Figure 11. Leaking side channels of the model, responses of queries to the system, and



**Figure 10. Side-channel attack based on the execution time of individual input queries, which can be used to decipher the depth of the DNN model and estimate the network parameters/model.**



**Figure 11. IP stealing from a trained DNN: the objective of a stealing attack can either be to (a) estimate the underlying DNN model or (b) predict the data set used for DNN training, using multiple queries. (a) Model stealing attack. (b) Dataset stealing attack.**

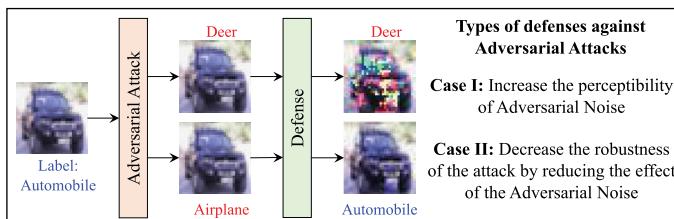
similar behavioral network characteristics can be exploited, analyzed, and reverse engineered to obtain the underlying IP.

### Defenses against security vulnerabilities of ML systems

To ensure correct operation in the presence of security attacks, several security defenses have been proposed over the years. This section describes some of the most prominent ML system defenses against security threats, categorized according to the threats they counter.

#### Defending against adversarial attacks

The concerns originating from adversarial attacks are the confidence reduction of the true output class and misclassification.



**Figure 12. Defenses against adversarial attacks either increase the perceptibility of adversarial noise (Case I) or decrease the effect of the adversarial noise (Case II).**

As shown in Figure 12, the defenses against adversarial attacks are generally intended either to: 1) increase the perceptibility of the attack, thereby ensuring that the clean and malignant inputs are perceptually distinguishable or 2) reduce the impact of the attack by enhancing the DNN's robustness against it.

For evasion-based adversarial attacks crafted using input gradients, a natural defense strategy is to hide these gradients using a technique called gradient masking [30]. This technique, as explained in Figure 13, reduces the dependability of output classification by retraining the DNN with the output probability vector. Adversarial training, as shown in Figure 14a, is another commonly used defense [74], [75], where a trained DNN is retrained with adversarial inputs and the correct corresponding output labels. This improves the accuracy of the system in the presence of a *known* attack. Another defense, which actually constitutes a part of most practical ML systems, is the use of input preprocessing [31]. This defense smooths out, transforms, and truncates the noise before it is even fed to the DNN. As shown in Figure 14b, this defense reduces the adversarial noise and hence reduces the chances of a successful attack. A recent defense against adversarial attacks is to train robust image classifiers [76]. This defense exploits the fact that images contain high redundancy due to the strong correlation between neighboring pixels, so that a subset of pixels can be used to represent the same information. This subset is chosen by randomly dropping pixels from input images, and it is used during DNN training and inference. The drop rates are chosen randomly between 0% and 100% for each input image and at each epoch. The model trained on such subsampled data sets is robust against adversarial attacks.

Most of the above defenses may work against a naive attack. However, for a strong attack, these defenses may fail. Many studies show that gradient masking does not increase the robustness of a DNN [77]–[79], and hence can be broken with the use of a substitute model to identify the approximate gradient direction [80]. Attacks aware of preprocessing defenses [31] can break the filtering defense. Likewise, as studied by several works [56], [81], adversarial training overfits a DNN to the adversarial examples and does not necessarily make the network more robust. Hence, a stronger attack can again make the DNN fail for certain inputs.

For poisoning-based adversarial attacks, a simple defense strategy is not to outsource the training

process to a third party (i.e., local training). However, training is a lengthy process, requiring large computational resources. Hence, local training is not always feasible for large DNNs. To outsource the training of large DNNs, the training data can be encrypted before outsourcing it to the third party [34]–[36], to overcome the impact of data poisoning.

For attacks exploiting dormant neurons in the network, pruning can be employed to remove the (dormant) neurons that are not significant to the network inputs, thereby reducing the chances for a successful backdoor attack. Yet, pruning-aware attacks [33] can be used to train only the significant network neurons with backdoor behavior, which eliminates the effectiveness of the pruning defense. Another defense is to fine-tune the DNN with clean inputs [33]. Although this does not eliminate the backdoors from the network, it significantly reduces the chances of a successful backdoor attack.

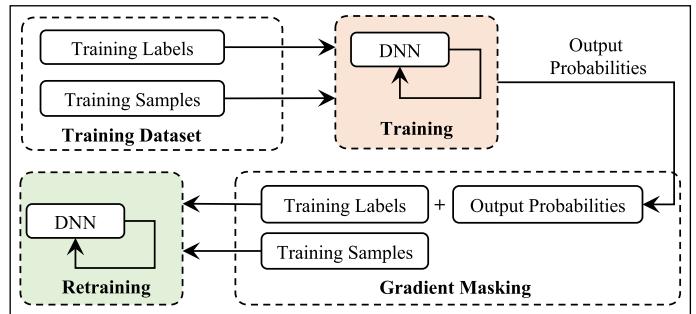
To formulate better defenses against adversarial attacks, a current research focus is to determine robustness bounds for DNNs using formal methods [82]–[84]. Although this area of study is relatively new and generally not scalable to practical DNNs, it has the potential to determine the actual boundaries where the DNNs will no longer be vulnerable to the adversarial attacks. However, the question of how the knowledge of these bounds can be used to actually prevent adversarial attacks is yet to be answered.

### Defending against neural-level Trojans

Similar to adversarial attacks, the trigger for incorrect DNN behavior in neural-level Trojans is a malicious input. Hence, techniques that manipulate or detect input discrepancies can reduce the effect of neural-level Trojans. Such approaches include input preprocessing [31] to smooth out the input trigger, input anomaly detection [85] to identify suspicious input patterns, and prediction distribution [58] to identify the bias of DNN toward the targeted output. Since Trojans are inserted into pretrained DNN models, their effect could also be negated using local training [33], i.e., training the DNN model locally instead of outsourcing the training process to third-party cloud servers.

### Defending against hardware attacks

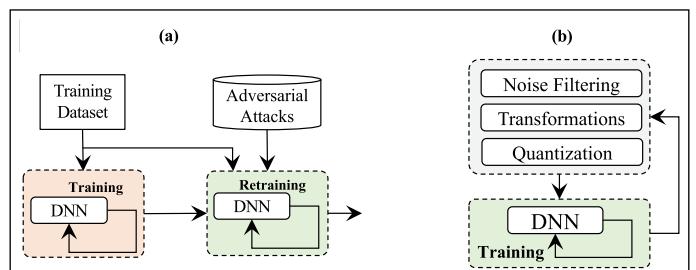
Hardware Trojans [61], [86]–[89] are a hardware-related security problem in ML systems. A hardware Trojan is a malicious modification of a



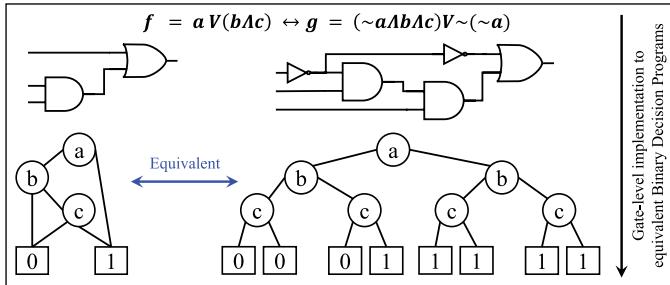
**Figure 13. Using gradient masking to hide the input gradients that might be used by the attacker to determine the perturbations that need to be inserted to perform the adversarial attack.**

circuit design that results in an undesired behavior, e.g., leakage of sensitive information, malfunction, or performance degradation. Since these attacks make use of hardware modifications, a suitable defense strategy against them is to use formal methods [27], particularly via equivalence checking. Figure 15 demonstrates the use of binary decision diagrams (BDDs) for equivalence checking [90] of simple gate-level circuits. The biggest obstacle to implement the equivalence checking defense is the absence of a golden/reference model of the actual system hardware to compare with the intended system model [61], [62].

Other potential defenses against hardware Trojans include side-channel analysis [39] for anomaly detection, and crosslayer attack modeling via bridging the gap between the hardware and software [38].



**Figure 14. (a) Improving a DNN's accuracy in the presence of a known attack by training the data set with adversarial examples obtained from known adversarial attacks, i.e., adversarial training. (b) Reducing the effects of adversarial noise added to the input via input-preprocessing techniques such as noise filtering, quantization, and other input transformations. (a) Adversarial training. (b) Preprocessing-based defense.**



**Figure 15. Using BDDs for hardware equivalence checking.**

This defense often assumes that the leaking information, e.g., power trace, of the Trojan is large enough to be detectable. The defense becomes ineffective when the assumption does not hold [66].

As mentioned in the previous section, side-channel attacks make use of side-channel leakage from the system, often giving rise to other security vulnerabilities in ML systems, such as hardware intrusion [45] and IP stealing [39]. Side-channel attacks often rely on the exactness of the leaking information; hence, the defense against them relies on the addition of random noise to system operations. For instance, a random selection of the next operation, whenever the sequence of operations does not matter, like selecting the sequence in which the image pixels are fed to the adder in an NN, could potentially make the inference of useful knowledge from side channels more difficult [40], [41], [91].

#### Defending against IP stealing

The most common IP stealing attacks involve stealing private or secret information (privacy infringement) and the robbery of the IP (piracy).

To protect *privacy* of data, the simplest defenses include blurring, obfuscation, and even the addition of adversarial noise to the data [92], [93]. In practice, these approaches may not work well as they may not be strong enough [94]. Relatively stronger defenses include the use of encryption [34], [35], i.e., data confidentiality, while outsourcing the data for training. Similarly, measures to ensure IP privacy during third-party DNN training include the use of multiple training servers for joint data set [95], verifying the training procedure [96], ensuring privacy after training by network transformation [97], obfuscating defenses against reverse engineering-based attacks [98], [99], and isolating the hardware accelerators [100].

To protect IP against *piracy*, the rounding approach [101] can be a potential defense. The leaking side channels could be a potential vulnerability exploited to deploy an IP stealing attack. Hence, the same side channels could be used for runtime monitoring to secure the ML system against IP stealing [39].

#### Reliability threats for ML systems

Security threats are not the only cause for an ML system not to work as expected. This section discusses several environmental/natural factors that lead to reduced ML system reliability.

##### Hardware faults

Errors in the hardware components that build up a system are generally classified into transient, intermittent, and permanent faults [102], [103]. As the name implies, *transient faults* induce temporary errors in the system. *Intermittent faults*, on the other hand, may cause recurring system glitches. Like transient faults, intermittent faults can be removed from the system, often by the use of additional circuitry. *Permanent faults* have a lasting impact on the system and can be removed mainly by replacing the faulty hardware component.

##### Transient faults

The nature of applications where the ML systems are deployed exposes these edge devices to harsh operating conditions like high temperature and altitude. These conditions, in addition to the increasing circuit clock frequencies, voltage reduction, and technology scaling, have been continuously increasing the occurrence of transient faults in systems over past decades [104]. Transient faults can be random, i.e., occurring unpredictably, or nonrandom, i.e., can be reproduced under certain circumstances [102]. Electrostatic discharge (ESD), electromagnetic radiation, noise in hardware interconnections, or flaw(s) in fabrication are among the leading factors contributing to transient faults [103], [105].

Soft errors [104] are a type of transient fault, mostly caused either by: 1) a high-speed particle (neutron or proton) strike from cosmic rays or 2) the emission of an alpha particle from impurities in IC packaging. Both particles generate a charge  $Q_{\text{rad}}$  in the transistor(s) (across the chip), and if this charge exceeds a certain threshold value  $Q_{\text{th}}$ , it is likely to change the state of the transistor, resulting in a bit-flip. This effect, known as the single-event upset (SEU), is becoming a

leading cause of concern with system hardware, particularly memory chips [106], [107]. With increasing technology miniaturization, such bit-flips can extend to multiple bits within a single data word [107], [108], i.e., a multiple-bit upset (MBU). This phenomenon of random bit-flips poses a challenge to robust ML. These effects may lead to misclassification in ML systems, as shown in Figure 16a.

### Intermittent faults

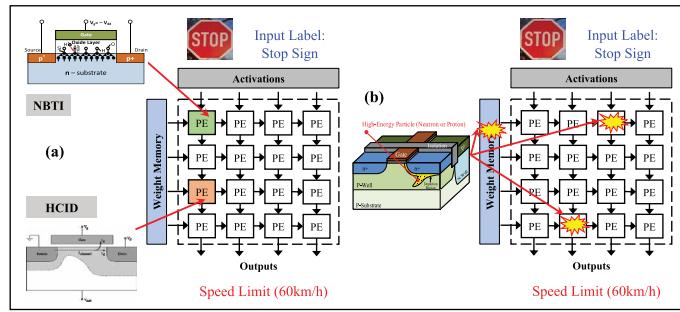
Such faults are intermittent and relatively unpredictable, which make them difficult to repeat, analyze, and understand. Process variation [29], [109] is the phenomenon that results in small differences in the physical characteristics of seemingly identical circuit components during fabrication. This may lead to intermittent faults, potentially leading to permanent damage to the system chip [102]. Similarly, aging [110] (Figure 16b) can cause deterioration of system performance and functions over time. Another important factor contributing to intermittent faults in hardware is temperature under which the edge device is operating. Temperature effects [111] reduce system reliability by increasing device aging and error rates.

Often as the result of component aging, timing errors occur, where the system is unable to provide correct output within the expected time. Usually, as the error propagates through the chain of components, the magnitude of error increases. Not only does this reduce ML classification accuracy, but it may also make the ML model vulnerable to serious security concerns [112].

Accessing memory with a specific access pattern can introduce access pattern-dependent faults, which could be caused by disturbance errors. These faults create a security vulnerability known as Rowhammer [113], [114], which is the phenomenon where repeatedly accessing a row in a modern dynamic random access memory (DRAM) chip causes disturbance errors in physically adjacent rows. DRAM data retention failures [115]–[118] can also cause intermittent and unpredictable faults due to DRAM variable retention time and data pattern dependence.

### Permanent faults

These faults are irreparable, where the system portrays fixed/repetitive errors like *stuck-at* faults.



**Figure 16. Effects of reliability threats, i.e., (a) aging and (b) soft errors on ML systems.**

Factors contributing to permanent faults include cosmic radiation, ESD in device, fabrication flaws [102], [105], [119], or recurring intermittent faults.

### Neural network anomalies

Environmental noise (EN) [120], [121] has the same impact on edge devices as adversarial attacks have on DNNs

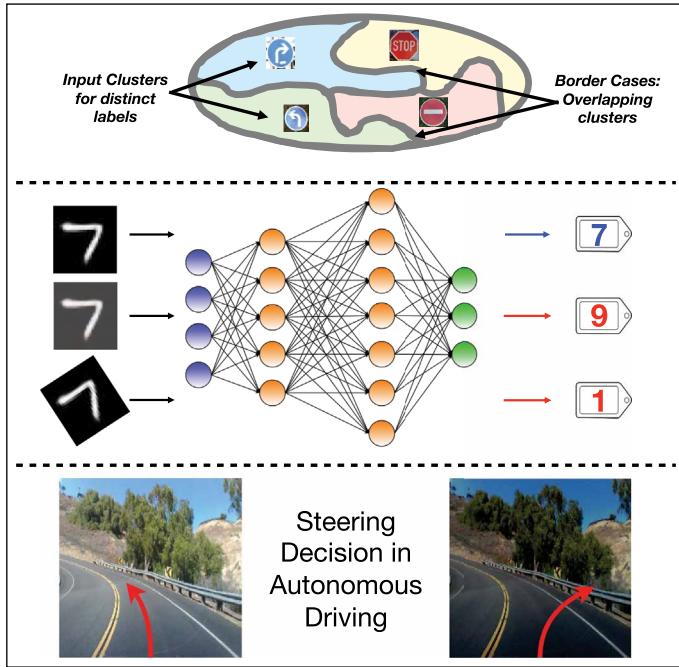
$$f(x) \neq f(x+EN). \quad (2)$$

For instance, for an object classification system, possible EN could be due to fog or pollution in the atmosphere, which can produce effects of blurring on the input. Similarly, variations in data acquisition by the edge sensors can also lead to faulty inference in an ML system. For an image-acquisition system deployed in an autonomous vehicle, change in either *brightness*, *contrast*, *camera angle*, or any other *photometric transform* [74], [122] can impact the decision-making of the vehicle and may lead to serious consequences [123].

The reason for such DNN anomalies is a lack of generalization of DNN for unseen inputs. The classification boundaries of the DNN outputs may overlap in the hyper-space, as depicted for a 2-D space in Figure 17 (top). The inputs closer to these boundaries are vulnerable, and slight changes in input, even in the absence of a malicious attacker, may lead to misclassification.

### Mitigation techniques for reliability threats in ML systems

This section discusses several mitigation techniques for the reliability threats in ML systems discussed in the “Reliability threats for ML systems” section.



**Figure 17. Inputs close to cluster boundaries (top) in hyperspace are most vulnerable to environmental adversarial transformations. Variation during data acquisition (middle) can cause misclassification, which can lead to drastic effects in ML systems (bottom) [77].**

#### Mitigation techniques for hardware faults

The most notable approaches to ensure system reliability in the presence of various hardware faults are as follows.

#### Protection against transient faults

Generally, transient faults can be removed by a *component reset* or *system reboot*. However, these are often not the most desirable solutions. Interleaving to prevent errors in consecutive bits [124], using additional circuitry for error detection [125], scrubbing to periodically remove errors to prevent error accumulation [117], [126], adding hardware redundancy and voting mechanisms [127] to rule out the erroneous bits, and using error detection and correction codes [112], [128]–[130] are generally the preferred choices to defend against soft errors in memories and logic. Recently, replicating the complete hardware accelerator and conjoining the accelerators with majority voting are also being used to ensure safety in ML systems. For instance, Tesla's self-driving car computer has two chips deployed to tolerate faults [131].

Numerous approaches are available to handle transient errors; yet, all these approaches provide a tradeoff between error detection and correction capability, area, power consumption, and latency. Redundancy-based approaches can incur large area overhead and cost. A recent work shows that, in a DNN-based system, the bit-flips from 1 to 0 have a more drastic effect on the system's classification accuracy than bit-flips from 0 to 1 [132]. This finding could be used for system design with stronger error-correction mechanisms deployed for more critical bit-flips.

#### Protection against intermittent faults

As system components age at different rates, components in the same chip may require different levels of protection. Protection techniques that are consistent throughout the system, like chip-level guardbanding, may thus not be sufficient. A recent work [133] studies dynamic protection approaches that ensure that the most vulnerable components receive the highest protection in the system. The same work also proposes age-aware workload management to age all components of the system at the same rate. Disturbance errors like Rowhammer can be mitigated via probabilistic mechanisms [113] and various other hardware or software techniques [114]. Online profiling of memory cells [115]–[117], [134]–[136] can also help the system to discover and disable weak cells with intermittent or aging-related errors.

To detect timing errors, several studies propose to use Razor flip-flops [137]–[139]. Once a timing error is detected, error correction is usually employed by either introducing slack in computation, skipping a clock cycle, or scaling voltage to mitigate the error's effect. However, these approaches may introduce a delay in execution as the correct result propagates to the output. Another mitigation approach to defend against timing errors is formal timing analysis [140], [141]. Such timing-verification approaches are intended to ensure that the system behaves correctly within the defined timing bounds.

#### Protection against permanent faults

Hard errors imply irreversible chip damage, for which the most effective solution is usually to replace the faulty chip/component. However, this is a costly solution. A relatively cost-effective alternative to chip replacement is discarding only the erroneous bits/byte of the component [142], [143], which minimizes the

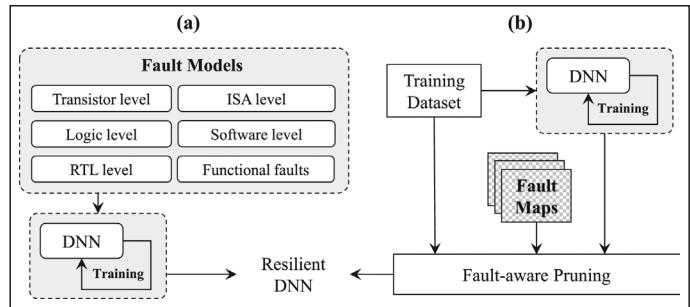
cost incurred. Specific to ML systems, techniques like fault-aware training, pruning, mapping, and activation clipping are often used to address permanent faults [106], [144], [145], [179], [180]. In fault-aware training, the DNN is trained for different faults at multiple levels, like at the transistor and logic levels, as shown in Figure 18a. This is a computationally costly solution. In fault-aware pruning, all the DNN connections and parameters that map to faulty processing elements or nodes are pruned using fault maps of the baseline hardware (e.g., systolic array-based accelerator), as shown in Figure 18b. In fault-aware mapping [179], the saliency of DNN parameters is exploited to define a mapping of different segments of the DNN while retaining the salient parameters. In fault-aware activation clipping [180], the activation values exceeding a pre-defined threshold for fault-free NNs are clipped. This eliminates the need for either pruning or retraining.

#### Mitigation techniques against environmental noise

Similar to defenses against adversarial attacks (the “Defending against adversarial attacks” section), preprocessing filters [31] can reduce the effects of EN in DNNs. Likewise, adversarial training [75] of the DNN with noisy inputs could improve DNN accuracy for certain noise patterns. However, similar to the effect of using adversarial training for adversarial attacks, this solution may not work well because adversarial training overfits the network to adversarial examples but does not ensure better generalization [56]. Since the accuracy of ML systems in the presence of EN and varying input data arises due to the lack of generalization to unseen inputs in the DNN, an alternative solution could be to train the DNN on a larger input data set. However, it is not always possible to obtain a large and diverse input data set. To overcome this limitation, some works propose the generation of synthetic data sets [146]–[149]. Yet, real input domains are mostly very large, multi-dimensional, and of continuous spaces. Hence, it is uncertain if any *finite* number of synthetic input points could be sufficiently representative of the *entire* input domain, allowing the trained DNN to generalize for unseen inputs.

#### Formal verification for robust ML

As briefly highlighted in the “Machine learning: Concepts and terminology” section, testing a trained DNN using a labeled data set is insufficient to ensure



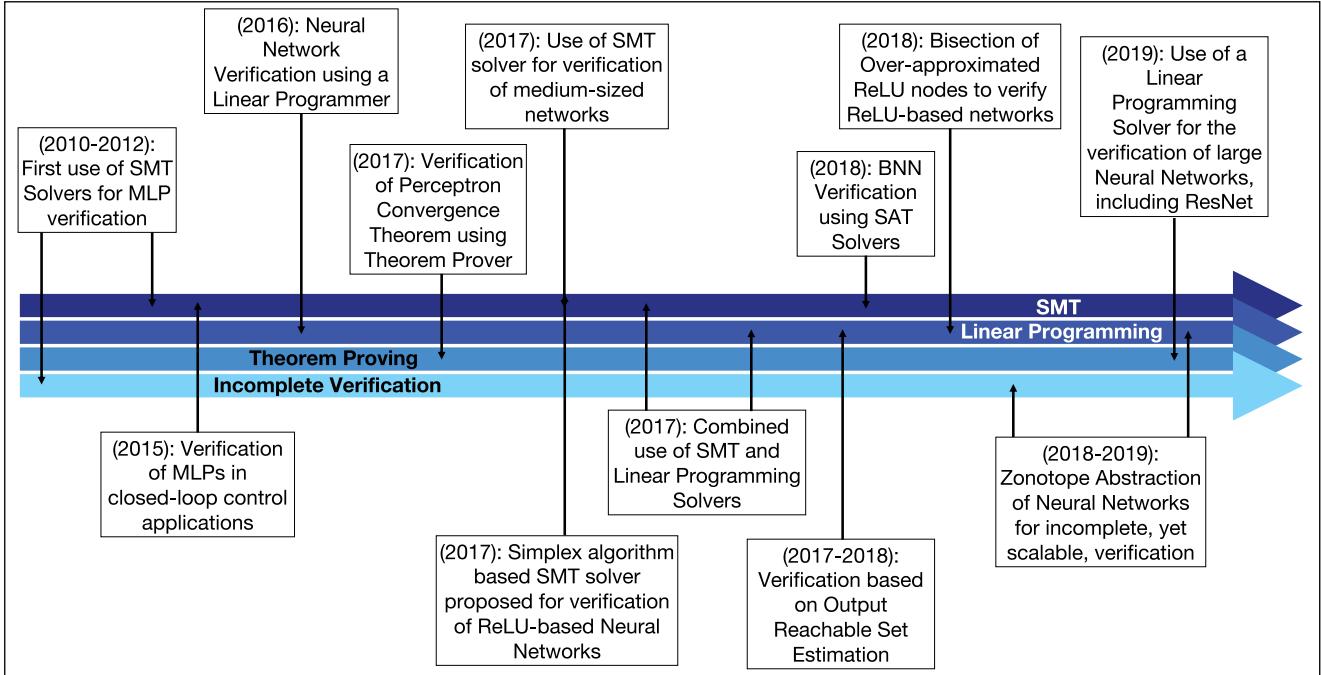
**Figure 18. Mitigation techniques for permanent faults in ML systems. (a) Fault-aware training. (b) Fault-aware pruning.**

reliable DNN inference. This is due to the lack of generalization of DNNs for unseen inputs. Recently, efforts have been made to understand and interpret the decision-making process inside the DNNs, hoping to provide dependable guarantees regarding DNN inference. These include exploring input feature space [150], using saliency maps to understand DNN inference [151], and developing various certifiability criteria for DNN interpretability [152]–[154].

Formal verification provides an orthogonal alternative to testing that provides formal or mathematical guarantees regarding NN performance at the edge. The use of formal verification for hardware and software has existed for a long time [27], [28]. Yet, research on verification of NNs, which forms an essential component of the ML system, has been an active domain of research for only a decade. Figure 19 summarizes the major milestones reached in NN verification over time, according to the four major verification categories: satisfiability (SAT) and satisfiability modulo theories (SMT) solving, linear programming (LP), theorem proving, and incomplete verification.

#### SAT/SMT

SAT checking is the branch of formal verification where the system model and the property to be verified for the system are expressed in a propositional logic, and written into conjunctive normal form (CNF), as shown in Figure 20 (bottom). The formula is then checked by an automatic SAT solver. Having an SAT output implies that a satisfying solution to the negation of the property, i.e., a counterexample, has been found. An UNSAT output implies the absence of any counterexample, and hence indicates that the stated property holds for the system. SMT is a variant of SAT that works similar to SAT solving, as shown



**Figure 19. A decade of verification techniques for NNs.**

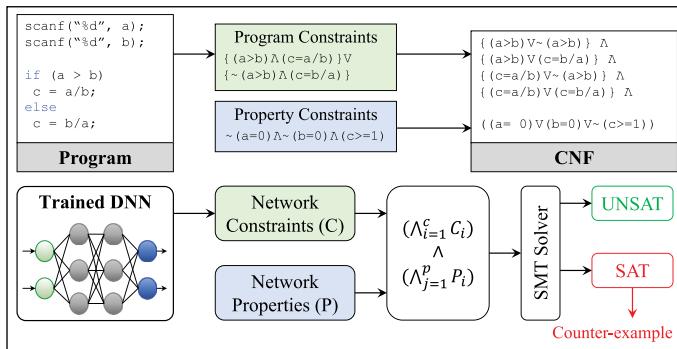
in Figure 20 (top), but allows the use of theories beyond a propositional logic, like linear arithmetic.

Since SAT solving allows the use of only propositional variables (i.e., atoms), it is often the verification approach of choice for binarized NNs (BNNs) [82], [155]. SMT solvers, on the other hand, are preferred for verifying DNNs with real and/or integer network parameters [156], [157]. Another concept often associated with SAT-based verification approaches is counterexample-guided abstraction refinement (CEGAR) [158], which produces more reliable verification results by

iteratively improving the network model using counter-examples. CEGAR and its variants provide an efficient verification solution when the DNN is modeled using overapproximation [159].

However, SAT-based verification suffers from the scalability problem: state-of-the-art techniques are capable of verifying only small networks [160], [161], comprising less than ten neurons, to medium-sized networks [157], comprising of up to 20,000 neurons. Although some works propose optimizations, like  $K$ -factoring [155], to reduce the size of this problem, applying these optimizations can be computationally costly. More rigorous and cost-effective optimizations can improve the scalability problem with SAT-based DNN verification.

Another challenge is to design more efficient SAT/SMT solvers. There has been a tremendous improvement in the state-of-the-art SAT solvers in recent years, with increased computational speed and capability to deal with larger networks. Yet, there is a lack of dedicated tools for DNN verification; existing tools [156] are not scalable to larger networks. More powerful SAT/SMT solvers could be key for the improvement of DNN verification.



**Figure 20. Using an SMT solver for verification. CNF expresses program and property constraints of the C code (top), and SAT/SMT solver for a DNN-based system (bottom).**

#### Linear programming

LP-based verification works by defining the system as a set of linear constraints, and the property

to be verified as an objective function, as shown in Figure 21. The objective function can be either a minimization or a maximization function. The search of the minima or maxima is automatic and involves the use of linear programmers [162], [163].

For DNN verification, LP is generally used to check the robustness of the network against adversarial attacks. The objective is to determine the smallest noise (or noise margin) that satisfies linear constraints of the network but causes misclassification at network output [81], [84].

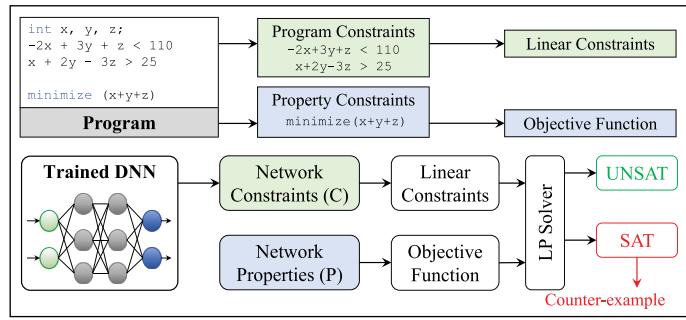
As the name suggests, an inherent limitation of LP is that it requires the constraints to be linear. For DNNs, this poses a problem due to the presence of nonlinear activation functions. Some works [164], [165], as will be discussed in the “Incomplete verification” section, replace nonlinear activation functions by their linear approximations. This yields incomplete verification results since a linear representation is insufficient to fully replicate the behavior of the actual nonlinear activation function. Another approach, proposed for rectified linear unit (ReLU)-based networks, is input bisection for selected network nodes [166]. ReLU is a piecewise linear function that works like a half-way rectifier: output is zero if the input is negative, but the output equals the input for all nonnegative input values. A calculated input bisection splits ReLU into two linear functions, at the cost of a larger size-verification problem.

The use of Big-M encoding<sup>1</sup> is proposed in several recent works [57], [83], [168]–[170]. Although the approach ensures reliable verification results, without a significant increase in the size of the problem, it also suffers from the scalability problem. Reducing the number of constraints by eliminating the inactive neurons [84], and exploiting the sparsity of practical DNNs may allow the effective verification of practical-sized ML systems.

### Interactive theorem proving

Theorem proving is a type of formal verification in which the system and its properties are defined mathematically, and the properties are verified for the system by rules of natural deduction [171]. The verification example demonstrated in Figure 5 shows how natural deduction-based reasoning works. Figure 22 gives

<sup>1</sup>The Big-M technique is used for the verification of ReLU-based networks, where a binary indicator variable  $Y$  is added to the linear constraints to indicate the linear region of the activation function to which the constraint belongs, while  $M$  provides a valid output upper bound that is greater than the maximum output value of every ReLU node in the network. We refer the reader to [167] for details of the technique.

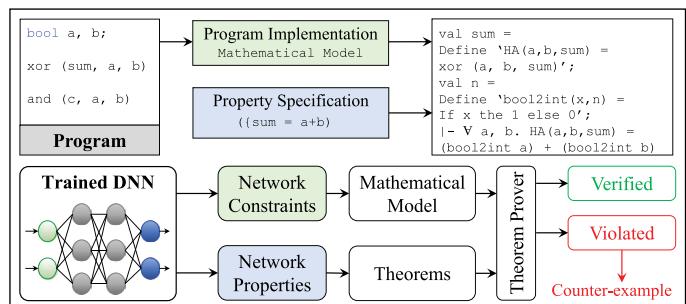


**Figure 21. Using a linear programmer to define the linear constraints and the objective function (top), and verification of a DNN-based system with a linear programmer solver (bottom).**

a more generic view of how theorem proving works. Generally, for the propositional logic and simple circuits, state-of-the-art theorem provers are able to verify the system without human intervention, i.e., these systems can be verified by automatic theorem provers. However, for complex systems, like DNNs, human guidance is essential, and hence the verification of such systems is done via interactive theorem proving.

For verification, the system is represented as a logical model governed by mathematical principles. The property is similarly expressed as a formal proof goal. The objective is to use axioms and rules derived from these axioms to check if the properties, i.e., system specifications, hold for the system model, i.e., the implementation.

As expected from a human-guided verification approach, interactive theorem proving is difficult to execute for two reasons. First, it requires an in-depth knowledge of the underlying system for realistic system modeling. Second, it demands the



**Figure 22. Using a theorem prover for the verification of a half-adder (top), and mathematical model and theorems of theorem proving for a DNN-based system (bottom).**

verifier to have an expert understanding of: 1) why a certain property holds for the system; 2) what are the required assumptions; and 3) how to prove the property on the basis of sound mathematical principles. Hence, it is no wonder that interactive theorem proving has been a scarcely explored research domain for DNN verification. Murphy et al. [172] verify the perceptron convergence theorem, but they focus on a very small subset of DNNs called binary classifiers that may not be easy to adopt for large state-of-the-art DNNs.

For more practical theorem-proving-based DNN verification approaches, the basic need is to understand how DNNs work, why they make certain decisions, and what are the mathematical reasons behind their behavior. The perceptron convergence theorem [2], [173] was proposed almost six decades before it was formally verified by Murphy et al. [172]. Hence, understanding and developing the theory behind DNN operation seem to be a logical step before theorem proving could be successfully used for DNN verification.

### Incomplete verification

Completeness is a notion that decides whether a system model is sufficient to prove everything about the system. Incomplete verification often makes use of abstract interpretation, linear approximation, and other similar approaches to formally model the system [174]–[176]. As a result, the system model is not an exact representation of the actual system but rather an overapproximation. Verification is then performed on this approximate model, as shown in Figure 23. It is important to note that simulation/testing, which also provides incomplete results, must not be confused with incomplete verification. This is because, in testing, the system is considered a black

box, and the tester analyzes the system behavior by feeding the black box with a finite set of inputs and recording the output. In contrast, in incomplete verification, the system is a white box representing the simplified version of the actual system, on which formal verification is performed.

Since incomplete verification involves verifying a simplified version of the actual system model, this makes the approach scalable, even to larger DNNs [164], [165]. To improve the completeness of verification, we can use abstraction refinement approaches like CEGAR [158], [159]. This does not entirely eliminate the problem of incompleteness of verification, but improves the reliability of verification results.

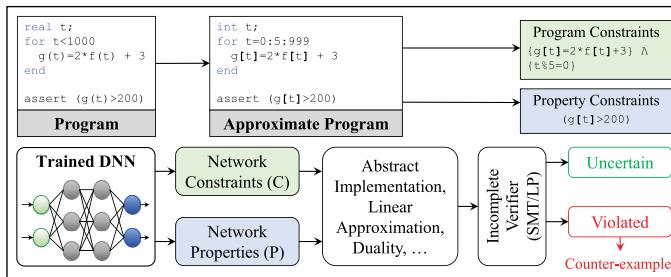
Incomplete verification often leads to false positives [164], [165]. Whenever the incomplete verifier provides counterexamples, they are actual scenarios where the property does not hold for the system. If the verifier provides no counterexamples, the system may still be unsafe or the property being verified may still not hold for some inputs to the system [177].

Incomplete verification is scalable and, hence, is an attractive verification alternative for DNNs. Yet, its inherent incompleteness provides the biggest limitation to its accuracy. A possible solution is to trade off some scalability of incomplete verification with completeness [178]. This can be accomplished by iteratively refining the network model until it matches the exact system model [158], or combining incomplete verification with other complete verification approaches like SMT solvers or LP.

### Open challenges and discussion

Although ML is a rapidly evolving domain, it will probably pass a long time until ML systems are considered robust. In ML systems, similar to other systems, a single vulnerability is sufficient to pose a security or reliability issues that might prevent the system from obtaining accurate results. However, it is very challenging to provide strong robustness guarantees, because we need to deal with a wide range of security and reliability threats, while considering the probabilistic/stochastic nature of the ML algorithms. This section discusses some important (in our view) open challenges for achieving robust ML systems.

ML systems have numerous security issues mainly related to: 1) outsourced training; 2) untrusted fabrication foundries; and 3) attacker access to the environment in which the system is deployed.



**Figure 23: Using incomplete verification for:  
1) verifying a continuous domain program (top)  
and 2) verifying a DNN-based system (bottom).**

Among these security issues, some of the most important are the following.

- *Securing training data sets* before outsourcing them for training. This may involve encrypting the training data set from the cloud servers to ensure IP privacy or minimizing the impact of data poisoning attacks.
- *Obfuscating ML hyper-parameters, algorithms, and IPs*. There are several defenses to successfully obfuscate ML hyper-parameters, algorithms, and IPs using blurring and noise addition. However, as indicated earlier, these techniques do not often work well in practice. Hence, a prospective obfuscation method could be the inclusion of various obfuscation techniques in a single framework, and random switching between these techniques to ensure a more secure ML system.
- *Ensuring fairness of training*, i.e., preventing the bias of the trained NN. Gradient-based adversarial input generation and counterexamples generated via formal verification of NNs can be used to identify the bias in training. This method is based on the observation that adversarial inputs are more likely to identify the output classes to which the trained NN is biased.
- *Validating the functional and behavioral correctness of ML hardware*. Formal verification methods may be required to provide stronger guarantees on the ML hardware operation by performing verification under diverse security and reliability conditions.
- *Minimizing the accessibility to side-channel leakages*. This can be achieved by minimizing the sharing of resources like memory and power, thereby ensuring the hardware isolation of the ML system. However, this may be a costly solution for most ML applications. Another prospective solution to ensure minimal access of an attacker to the side-channel leakages can be the introduction of complementary synthetic noise to nullify the side-channel signatures of the system.

The DNN model and the hardware that run the model are both vulnerable to inconsistencies in performance over their lifetime. Major unresolved reliability challenges in ML systems include:

- *Developing frameworks to emulate ML systems* under diverse operating conditions. This is essential to: 1) study and better understand the

reliability challenges of the systems deployed in the physical environment; 2) assess the performance of the available mitigation techniques; and 3) analyze the tradeoffs between these approaches to identify the solution that ensures the highest system reliability.

- *Providing a fault-safe runtime* in the case of system discrepancies. Currently, such fault-safe techniques include the use of redundancies at the hardware and software levels, which ensure that, in the case of a component malfunctioning, the overall performance of the system is not affected. However, these measures are generally very costly and, hence, there still exists the need for better fault-safe mechanisms for ML systems.
- *Hampering the progression of subsystem failures* to the interconnected components. This requires mitigation approaches that can provide cross-layer reliability to ensure that a failure in one system component does not propagate and affect the results of the next system component(s).

Formal verification is a promising way to provide strong robustness guarantees in ML systems via mathematical proofs. The major challenges for making formal verification a practical tool to ensure robustness include:

- *Formally modeling* the nonlinear, nonconvex behavior of ML systems. Complete verification with existing modeling approaches (e.g., Big-M) is often not the optimal solution due to the large number of generated clauses and/or constraints. Incomplete verification is also not the optimal solution because it may lead to false-positive results due to overapproximation.
- *Incorporating the uncertainties of the real world into the formal system model*. Namely, the verification of the system under different reliability factors, e.g., EN.
- *Inspecting system behavior for all possible inputs*. Formal verification is widely acclaimed due to its rigorous analysis and complete results. However, due to the complexity of NN verification, current approaches rely on applying verification to only a subset of inputs (i.e., seed inputs).

Providing complete guarantees regarding system behavior requires more rigorous verification approaches. Moreover, end-to-end formal verification of the complete system, which is composed of

multiple ML-based subsystems and control subsystems, is a significant research challenge.

- *Optimizing the verification goal* to reduce the computational complexity of the verification problem. As the size of the underlying ML system increases, the size of its formal representation also increases. This requires large computational overhead and time to formally verify ML systems. Hence, simplifying the verification problem prior to the actual verification can reduce the computational complexity of the problem.
- *Improving the timing efficiency of verification*, while ensuring the completeness of verification results. There is a tradeoff between the timing cost of verification and the completeness of the verification results. With the development of efficient verification tools, the bridge between timing efficiency and completeness has been reduced. However, achieving the most optimal tradeoff between timing efficiency and completeness still remains an open challenge.
- *Scaling the verification algorithm* to be applicable to practically sized DNNs. With improvements in verification tools and formalization approaches,

the size of the DNNs that can be formally verified is increasing gradually.

Tackling the previous challenges and research directions is important for providing secure and reliable ML systems. However, as ML is a domain that advances very rapidly, there will probably be new challenges and research directions that will become important with the emergence of new ML models, deeper DNNs, unreliable hardware with reduced technology nodes, and new attack models.

**ML, PARTICULARLY NNs**, forms an essential component of modern CPSs. However, due to outsourced training, compromised foundries, stealthy attackers, system aging, and the harsh operating environment of these systems, both at the system cloud and edge levels, they are vulnerable to numerous security and reliability concerns. This survey highlights: 1) the most prominent security and reliability challenges for ML systems; 2) the mitigation approaches to defend the systems against these challenges; and 3) formal methodologies for verifying trained NNs. This survey also summarizes the most important open challenges that hamper robust ML systems. ■

## Appendix

NN	Description	Pictorial Representation of the Network
Feed-Forward NN	These are the NNs with neurons in every layer impacting only the decision of neurons in the successive layers. Hence, the networks are cycle/loop - free. The feed-forward networks are also called <i>fully connected</i> when every neuron in the preceding layer is connected to every neuron in the successive layer.	
RNN	RNNs comprise of feedback loop(s); hence, neurons in one layer can impact the values of neurons in successive as well as preceding layers. This provides temporal characteristics to the RNNs, i.e., the values of the neurons (or the internal memory of the network) varies temporally.	
CNN	Unlike the earlier fully connected networks, CNNs share network weights via convolution operation. This improves the local spatial correlation of the input, and ensures that only the most prominent input features of the input are carried to the successive network layers.	

*Continued*

NN	Description	Pictorial Representation of the Network
GAN	GANs involve an interplay between a generator and a discriminator for the training of the network. The generator produces synthetic inputs in the same latent space as the training dataset, while the discriminator learns to distinguish the original data from the synthetic data. Hence, the objective of the generator is to maximize the error (i.e., generate more realistic synthetic inputs) while the discriminator minimizes the error by learning to differentiate between real and synthetic input.	<pre> graph LR     RN[Random Noise] --&gt; G[Generator]     G --&gt; TI[Training Inputs]     TI --&gt; D[Discriminator]     TI --&gt; D     D --&gt; EF{Error Function}     EF --&gt; G   </pre>
Capsule Network (CapsNet)	CapsNets are built up of layers that operate on vectors, where each element of the vector represents the instantiation parameter that deduces whether the feature represented in the vector is actually present in the input. The length of the vector, on the other hand, represents the instantiation probability. The connections between two consecutive capsule layers are learned dynamically during inference through the <i>routing-by-agreement</i> algorithm, which iteratively updates the <i>coupling coefficients</i> of the CapsNet. In this way, capsules learn to interpret high level features in a hierarchical manner.	
Spiking NN	All the NNs discussed above assume a normalized firing frequency for the neurons. This neglects the dynamic behavior of the inputs like speech. SNNs make use of spike trains to depict the spatio-temporal characteristics of the input. Hence, SNNs are an important class of NNs particularly for time-dependent applications.	

## References

- [1]. N. Mäkitalo et al., "Safe, secure executions at the network edge: Coordinating cloud, edge, and fog computing," *IEEE Softw.*, vol. 35, no. 1, pp. 30–37, Jan. 2018.
- [2]. F. Rosenblatt, "The perceptron—A perceiving and recognizing automation," Cornell Aeronaut. Lab., Ithaca, NY, USA, Tech. Rep. 85-460-1, 1957.
- [3]. L. Wang et al., "Places205-VGGNet models for scene recognition," Aug. 2015, *arXiv:1508.01667*. [Online]. Available: <https://arxiv.org/abs/1508.01667>
- [4]. G. Huang et al., "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
- [5]. G. Anthes, "Lifelong learning in artificial neural networks," *Commun. ACM*, vol. 62, no. 6, pp. 13–15, May 2019.
- [6]. M. Fink et al., "Deep learning-based multi-scale multi-object detection and classification for autonomous driving," in *Fahrerassistenzsysteme*. Wiesbaden, Germany: Springer, 2019, pp. 233–242.
- [7]. G. Hu et al., "When face recognition meets with deep learning: An evaluation of convolutional neural networks for face recognition," in *Proc. IEEE Int. Conf. Comput. Vis. Workshop (ICCVW)*, Dec. 2015, pp. 142–150.
- [8]. G. Hinton et al., "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
- [9]. T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to Spam filtering," *Expert Syst. Appl.*, vol. 36, no. 7, pp. 10206–10222, Sep. 2009.
- [10]. Z. Yuan et al., "Droid-Sec: Deep learning in Android malware detection," in *Proc. SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4. New York, NY, USA: ACM, 2014, pp. 371–372.
- [11]. V. C. Gungor et al., "Smart grid technologies: Communication technologies and standards," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 529–539, Nov. 2011.
- [12]. M. Bojarski et al., "End to end learning for self-driving cars," Apr. 2016, *arXiv:1604.07316*. [Online]. Available: <https://arxiv.org/abs/1604.07316>
- [13]. Y. Lin, P. Wang, and M. Ma, "Intelligent transportation system (ITS): Concept, challenge and opportunity," in *Proc. IEEE Int. Conf. Big Data Secur. Cloud (BigDataSecurity)*, May 2017, pp. 167–172.

- [14]. A. Esteva et al., "A guide to deep learning in healthcare," *Nature Med.*, vol. 25, no. 1, pp. 24–29, 2019.
- [15]. F. Amato et al., "Artificial neural networks in medical diagnosis," *J. Appl. Biomed.*, vol. 11, no. 2, pp. 47–58, 2013.
- [16]. V. Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
- [17]. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
- [18]. M. I. Jordan, "Serial order: A parallel distributed approach," Institute for Cognitive Science Report, United States, Tech. Rep. 8604, Jun. 1986.
- [19]. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [20]. W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 52, nos. 1–2, pp. 99–115, Jan. 1990.
- [21]. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: MIT Press, 1986, vol. 1, ch. 8, pp. 318–362.
- [22]. Y. LeCun et al., "Handwritten digit recognition with a backpropagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 396–404.
- [23]. G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming autoencoders," in *Proc. Int. Conf. Artif. Neural Netw.* Berlin, Heidelberg: Springer, 2011, pp. 44–51.
- [24]. I. Goodfellow et al., "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
- [25]. J. Vreeken, "Spiking neural networks, an introduction," Utrecht University: Information and Computing Sciences, 2003.
- [26]. C. Szegedy et al., "Intriguing properties of neural networks," Dec. 2013, *arXiv:1312.6199*. [Online]. Available: <https://arxiv.org/abs/1312.6199>
- [27]. A. Camilleri, M. Gordon, and T. Melham, "Hardware verification using higher-order logic," Univ. Cambridge, Comput. Lab., Cambridge, U.K., Tech. Rep. UCAM-CL-TR-91, 1986.
- [28]. V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1165–1178, Jul. 2008.
- [29]. F. Kriebel et al., "Robustness for smart cyber physical systems and Internet-of-Things: From adaptive robustness methods to reliability and security for machine learning," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 581–586.
- [30]. A. Kurakin et al., "Ensemble adversarial training: Attacks and defenses," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–20.
- [31]. F. Khalid et al., "FAdeML: Understanding the impact of pre-processing noise filtering on adversarial machine learning," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2019, pp. 902–907.
- [32]. T. Gu et al., "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.
- [33]. K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2018, pp. 273–294.
- [34]. E. Hesamifard et al., "Privacy-preserving machine learning as a service," in *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 3, pp. 123–142, 2018.
- [35]. E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," Nov. 2017, *arXiv:1711.05189*. [Online]. Available: <https://arxiv.org/abs/1711.05189>
- [36]. R. Gilad-Bachrach et al., "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.
- [37]. T. Graepel, K. Lauter, and M. Naehrig, "MI confidential: Machine learning on encrypted data," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Berlin, Heidelberg: Springer, 2012, pp. 1–21.
- [38]. S. Rehman, M. Shafique, and J. Henkel, *Reliable Software for Unreliable Hardware: A Cross Layer Perspective*. Cham, Switzerland: Springer, 2016.
- [39]. F. Khalid et al., "Security for machine learning-based systems: Attacks and challenges during training and inference," in *Proc. Int. Conf. Frontiers Inf. Technol. (FIT)*, Dec. 2018, pp. 327–332.
- [40]. L. Wei et al., "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proc. Comput. Secur. Appl. Conf.* New York, NY, USA: ACM, 2018, pp. 393–406.
- [41]. V. Duddu et al., "Stealing neural networks via timing side channels," Dec. 2018, *arXiv:1812.11720*. [Online]. Available: <https://arxiv.org/abs/1812.11720>
- [42]. I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–11.
- [43]. F. Michels et al., "On the vulnerability of capsule networks to adversarial attacks," Jun. 2019, *arXiv:1906.03612*. [Online]. Available: <https://arxiv.org/abs/1906.03612>
- [44]. A. Marchisio et al., "Capsattacks: Robust and imperceptible adversarial attacks on capsule

- nets,” Jan. 2019, *arXiv:1901.09878*. [Online]. Available: <https://arxiv.org/abs/1901.09878>
- [45]. A. Salem et al., “MI-leaks: Model and data independent membership inference attacks and defenses on machine learning models,” Jun. 2018, *arXiv:1806.01246*. [Online]. Available: <https://arxiv.org/abs/1806.01246>
- [46]. F. Khalid et al., “TrISec: Training data-unaware imperceptible security attacks on deep neural networks,” in *Proc. IEEE 25th Int. Symp. On-Line Test Robust Syst. Design (IOLTS)*, Jul. 2019, pp. 188–193.
- [47]. B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognit.*, vol. 84, pp. 317–331, Dec. 2018.
- [48]. W. Brendel, J. Rauber, and M. Bethge, “Decision-based adversarial attacks: Reliable attacks against black-box machine learning models,” Dec. 2017, *arXiv:1712.04248*. [Online]. Available: <https://arxiv.org/abs/1712.04248>
- [49]. J. Chen, M. I. Jordan, and M. J. Wainwright, “HopSkipJumpAttack: A query-efficient decision-based attack,” Apr. 2019, *arXiv:1904.02144*. [Online]. Available: <https://arxiv.org/abs/1904.02144>
- [50]. M. Cheng et al., “Query-efficient hard-label black-box attack: An optimization-based approach,” *arXiv preprint arXiv:1807.04457*, 2018.
- [51]. L. Pengcheng, J. Yi, and L. Zhang, “Query-efficient black-box attack by active learning,” in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 1200–1205.
- [52]. Y. Dong et al., “Efficient decision-based black-box adversarial attacks on face recognition,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7714–7722.
- [53]. F. Khalid et al., “RED-Attack: Resource efficient decision based attack for machine learning,” Jan. 2019, *arXiv:1901.10258*. [Online]. Available: <https://arxiv.org/abs/1901.10258>
- [54]. R. Wiyatno and A. Xu, “Maximal Jacobian-based Saliency map attack,” Aug. 2018, *arXiv:1808.07945*. [Online]. Available: <https://arxiv.org/abs/1808.07945>
- [55]. A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–17.
- [56]. N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.
- [57]. J. Steinhardt, P. W. W. Koh, and P. S. Liang, “Certified defenses for data poisoning attacks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3517–3529.
- [58]. Y. Liu, Y. Xie, and A. Srivastava, “Neural Trojans,” in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 45–48.
- [59]. S. Thys, W. V. Ranst, and T. Goedemé, “Fooling automated surveillance cameras: Adversarial patches to attack person detection,” *CoRR*, vol. abs/1904.08653, Apr. 2019. [Online]. Available: <https://arxiv.org/abs/1904.08653>
- [60]. M. Zou et al., “Potrojan: Powerful neural-level Trojan designs in deep learning models,” Feb. 2018, *arXiv:1802.03043*. [Online]. Available: <https://arxiv.org/abs/1802.03043>
- [61]. S. Bhunia et al., “Hardware Trojan attacks: Threat analysis and countermeasures,” *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.
- [62]. J. Clements and Y. Lao, “Hardware Trojan attacks on neural networks,” Jun. 2018, *arXiv:1806.05768*. [Online]. Available: <https://arxiv.org/abs/1806.05768>
- [63]. R. Karri, J. Rajendran, and K. Rosenfeld, “Trojan taxonomy,” in *Introduction to Hardware Security and Trust*. New York, NY, USA: Springer, 2012, pp. 325–338.
- [64]. J. Clements and Y. Lao, “Hardware Trojan design on neural networks,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.
- [65]. Y. Zhao et al., “Memory Trojan attack on neural network accelerators,” in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2019, pp. 1415–1420.
- [66]. I. H. Abbassi et al., “TrojanZero: Switching activity-aware design of undetectable hardware Trojans with zero power and area footprint,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2019, pp. 914–919.
- [67]. L. Batina et al., “CSI neural network: Using side-channels to recover your artificial neural network information,” Oct. 2018, *arXiv:1810.09076*. [Online]. Available: <https://arxiv.org/abs/1810.09076>
- [68]. K. Yoshida et al., “Model-extraction attack against FPGA-DNN accelerator utilizing correlation electromagnetic analysis,” in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2019, p. 318.
- [69]. S. Pal et al., “A framework for the extraction of deep neural networks by leveraging public data,” May 2019, *arXiv:1905.09165*. [Online]. Available: <https://arxiv.org/abs/1905.09165>
- [70]. L. Batina et al., “CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel,” in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.)*, 2019, pp. 515–532.
- [71]. W. Hua, Z. Zhang, and G. E. Suh, “Reverse engineering convolutional neural networks through side-channel information leaks,” in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

- [72]. F. Tramèr et al., "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, Aug. 2016, pp. 601–618.
- [73]. M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: ACM, 2015, pp. 1322–1333.
- [74]. K. Pei et al., "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. 26th Symp. Operating Syst. Princ. (SOSP)*. New York, NY, USA: ACM, 2017, pp. 1–18.
- [75]. K. Pei et al., "Towards practical verification of machine learning: The case of computer vision systems," Dec. 2017, *arXiv:1712.01785*. [Online]. Available: <https://arxiv.org/abs/1712.01785>
- [76]. H. Hosseini, S. Kannan, and R. Poovendran, "Dropping pixels for adversarial robustness," *CoRR*, vol. abs/1905.00180, May 2019. [Online]. Available: <http://arxiv.org/abs/1905.00180>
- [77]. T.-W. Weng et al., "On extensions of clever: A neural network robustness evaluation algorithm," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, Nov. 2018, pp. 1159–1163.
- [78]. I. Goodfellow, "Gradient masking causes CLEVER to overestimate adversarial perturbation size," Aug. 2018, *arXiv:1804.07870*. [Online]. Available: <https://arxiv.org/abs/1804.07870>
- [79]. A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," Feb. 2018, *arXiv:1802.00420*. [Online]. Available: <https://arxiv.org/abs/1802.00420>
- [80]. N. Papernot et al., "Towards the science of security and privacy in machine learning," Nov. 2016, *arXiv:1611.03814*. [Online]. Available: <https://arxiv.org/abs/1611.03814>
- [81]. O. Bastani et al., "Measuring neural net robustness with constraints," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2613–2621.
- [82]. N. Narodytska et al., "Verifying properties of binarized deep neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 6615–6624.
- [83]. S. Dutta et al., "Output range analysis for deep feedforward neural networks," in *Proc. NASA Formal Methods Symp.* Cham, Switzerland: Springer, 2018, pp. 121–138.
- [84]. V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–21.
- [85]. M. A. Siddiqui et al., "Detecting cyber attacks using anomaly detection with explanations and expert feedback," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 2872–2876.
- [86]. D. Agrawal et al., "Trojan detection using IC fingerprinting," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 296–310.
- [87]. S. Adey, "The hunt for the kill switch," *IEEE Spectr.*, vol. 45, no. 5, pp. 34–39, May 2008.
- [88]. Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *Proc. IEEE Int. Workshop Hardw.-Oriented Secur. Trust*, Jun. 2008, pp. 51–57.
- [89]. M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design Test*, vol. 27, no. 1, pp. 10–25, Jan./Feb. 2010.
- [90]. L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*. San Mateo, CA, USA: Morgan Kaufmann, 2009.
- [91]. A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," 2019, *arXiv:1910.13063*. [Online]. Available: <https://arxiv.org/abs/1910.13063>
- [92]. Q. Sun et al., "Natural and effective obfuscation by head inpainting," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 5050–5059.
- [93]. M. Sharif et al., "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.-CCS*. New York, NY, USA: ACM, 2016, pp. 1528–1540.
- [94]. B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 36–52.
- [95]. Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4672–4681.
- [96]. P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.
- [97]. J. Liu et al., "Oblivious neural network predictions via minionn transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.-CCS*. New York, NY, USA: ACM, 2017, pp. 619–631.
- [98]. M. Isakov et al., "Preventing neural network model exfiltration in machine learning hardware accelerators," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2018, pp. 62–67.
- [99]. Y. Liu, D. Dachman-Soled, and A. Srivastava, "Mitigating reverse engineering attacks on deep neural networks," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 657–662.

- [100]. X. Wang et al., "NPUFort: A secure architecture of DNN accelerator against model inversion attack," in *Proc. 16th ACM Int. Conf. Comput. Frontiers*. New York, NY, USA: ACM, 2019, pp. 190–196.
- [101]. T. Hunt et al., "Chiron: Privacy-preserving machine learning as a service," 2018, *arXiv:1803.05961*. [Online]. Available: <https://arxiv.org/abs/1803.05961>
- [102]. A. Tuszyński, "Essential pattern and sequence sensitivity in semiconductor memories," Dept. Elect. Eng., Minnesota Univ., Minneapolis, MN, USA, Tech. Rep. ADA110263, 1980.
- [103]. C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Jul. 2003.
- [104]. R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test. Comput.*, vol. 22, no. 3, pp. 258–266, May 2005.
- [105]. W. D. Greason and G. S. P. Castle, "The effects of electrostatic discharge on microelectronic devices a review," *IEEE Trans. Ind. Appl.*, vol. IA-20, no. 2, pp. 247–252, Mar. 1984.
- [106]. J. J. Zhang et al., "Building robust machine learning systems: Current progress, research challenges, and opportunities," in *Proc. 56th Annu. Design Autom. Conf. (DAC)*, 2019, pp. 1–4.
- [107]. J. Meza et al., "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 415–426.
- [108]. W. Wu and N. Seifert, "MBU-Calc: A compact model for multi-bit upset (MBU) SER estimation," in *Proc. IEEE Int. Rel. Phys. Symp.*, Apr. 2015, pp. SE.2.1–SE.2.6.
- [109]. M. Shafique et al., "The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives," in *Proc. 51st Annu. Design Autom. Conf. (DAC)*. New York, NY, USA: ACM, 2014, pp. 1–6.
- [110]. A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Proc. 41st IEEE/ACM Int. Symp. Microarchit.*, Nov. 2008, pp. 129–140.
- [111]. K. Kang et al., "NBBI induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution?" in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2008, pp. 726–731.
- [112]. M. Shafique, F. Khalid, and S. Rehman, "Intelligent security measures for smart cyber physical systems," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 280–287.
- [113]. Y. Kim et al., "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 361–372.
- [114]. O. Mutlu and J. S. Kim, "RowHammer: A retrospective," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, to be published.
- [115]. S. Khan et al., "The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study," in *Proc. ACM SIGMETRICS Perform. Eval. Rev.* New York, NY, USA: ACM, vol. 42, no. 1, 2014, pp. 519–532.
- [116]. J. Liu et al., "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," in *Proc. 40th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: ACM, 2013, pp. 60–71.
- [117]. M. K. Qureshi et al., "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 427–437.
- [118]. S. Khan et al., "Detecting and mitigating data-dependent DRAM failures by exploiting current memory content," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*. New York, NY, USA: ACM, 2017, pp. 27–40.
- [119]. J. T. Blandford, A. E. Waskiewicz, and J. C. Pickel, "Cosmic ray induced permanent damage in MNOS EAROMs," *IEEE Trans. Nucl. Sci.*, vol. 31, no. 6, pp. 1568–1570, Dec. 1984.
- [120]. A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–14.
- [121]. K. Eykholt et al., "Robust physical-world attacks on deep learning visual classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1625–1634.
- [122]. J. Lu et al., "No need to worry about adversarial examples in object detection in autonomous vehicles," 2017, *arXiv:1707.03501*. [Online]. Available: <https://arxiv.org/abs/1707.03501>
- [123]. *Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam*. Accessed: 14 February 2020. [Online]. Available: <https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html>
- [124]. P. Reviriego et al., "Protection of memories suffering MCUs through the selection of the optimal interleaving distance," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 2124–2128, Aug. 2010.

- [125]. F. Vargas and M. Nicolaidis, "SEU-tolerant SRAM design based on current monitoring," in *Proc. Int. Symp. Fault-Tolerant Comput.*, Jun. 1994, pp. 106–115.
- [126]. G.-C. Yang, "Reliability of semiconductor RAMs with soft-error scrubbing techniques," *IEE Proc. Comput. Digit. Tech.*, vol. 142, no. 5, p. 337, 1995.
- [127]. R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM J. Res. Develop.*, vol. 6, no. 2, pp. 200–209, Apr. 1962.
- [128]. R. W. Hamming, "Error detecting and error correcting codes," *Bell System Tech. J.*, vol. 29, no. 2, pp. 147–160, Apr. 1950.
- [129]. M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 395–401, Jul. 1970.
- [130]. M. Patel et al., "Understanding and modeling on-die error correction in modern DRAM: An experimental study using real devices," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 13–25.
- [131]. *Meet Tesla's Self-Driving Car Computer and Its Two AI Brains*. Accessed: 14 February 2020. [Online]. Available: <https://www.cnet.com/news/meet-tesla-self-driving-car-computer-and-its-two-ai-brains/>
- [132]. M. A. Hanif et al., "Robust machine learning systems: Reliability and security for deep neural networks," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2018, pp. 257–260.
- [133]. H. Lee, M. Shafique, and M. A. Al Faruque, "Aging-aware workload management on embedded GPU under process variation," *IEEE Trans. Comput.*, vol. 67, no. 7, pp. 920–933, Jul. 2018.
- [134]. M. Patel, J. S. Kim, and O. Mutlu, "The reach profiler (REAPER): Enabling the mitigation of DRAM retention failures via profiling at aggressive conditions," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 255–268.
- [135]. D. Lee et al., "Design-induced latency variation in modern DRAM chips: Characterization, analysis, and latency reduction mechanisms," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, pp. 1–36, Jun. 2017.
- [136]. S. Khan, D. Lee, and O. Mutlu, "PARBOR: An efficient system-level technique to detect data-dependent failures in DRAM," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2016, pp. 239–250.
- [137]. J. Zhang et al., "ThUnderVolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*. New York, NY, USA: ACM, Jun. 2018, p. 19.
- [138]. P. N. Whatmough et al., "DNN Engine: A 28-nm timing-error tolerant sparse deep neural network processor for IoT applications," *IEEE J. Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, Sep. 2018.
- [139]. E. Karl, D. Sylvester, and D. Blaauw, "Timing error correction techniques for voltage-scalable on-chip memories," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jul. 2005, pp. 3563–3566.
- [140]. S. Campos et al., "Timing analysis of industrial real-time systems," in *Proc. IEEE Workshop Ind.-Strength Formal Specification Techn.*, Apr. 1995, pp. 97–107.
- [141]. M. Pena et al., "Formal verification of safety properties in timed circuits," in *Proc. 6th Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, Nov. 2000, pp. 2–11.
- [142]. S. Schechter et al., "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. Int. Symp. Comput. Archit.* New York, NY, USA: ACM, 2010, pp. 141–152.
- [143]. J. Wang, X. Dong, and Y. Xie, "Point and discard: A hard-error-tolerant architecture for non-volatile last level caches," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*. New York, NY, USA: ACM, 2012, pp. 253–258.
- [144]. J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *Proc. IEEE 36th VLSI Test Symp. (VTS)*, Apr. 2018, pp. 1–6.
- [145]. S. Koppula et al., "EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.* New York, NY, USA: ACM, Oct. 2019, pp. 166–181.
- [146]. T. Dreossi et al., "Systematic testing of convolutional neural networks for autonomous driving," 2017, *arXiv:1708.03309*. [Online]. Available: <https://arxiv.org/abs/1708.03309>
- [147]. H. K. Ekbatani, O. Pujol, and S. Segui, "Synthetic data generation for deep learning in counting pedestrians," in *Proc. ICPRAM*, 2017, pp. 318–323.
- [148]. G. J. Stein and N. Roy, "GeneSIS-Rt: Generating synthetic images for training secondary real-world tasks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 7151–7158.
- [149]. Y. Tian et al., "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. 40th Int. Conf. Softw. Eng. (ICSE)*. New York, NY, USA: ACM, 2018, pp. 303–314.

- [150]. R. R. Selvaraju et al., "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proc. Int. Conf. Comput. Vis.*, Oct. 2017, pp. 618–626.
- [151]. J. Adebayo et al., "Sanity checks for saliency maps," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9505–9515.
- [152]. D. Alvarez-Melis and T. S. Jaakkola, "Towards robust interpretability with self-explaining neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7775–7784.
- [153]. M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proc. Int. Conf. Mach. Learn. (JMLR)*. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3319–3328.
- [154]. A. Levine, S. Singla, and S. Feizi, "Certifiably robust interpretation in deep learning," 2019, *arXiv:1905.12105*. [Online]. Available: <https://arxiv.org/abs/1905.12105>
- [155]. C.-H. Cheng et al., "Verification of Binarized Neural Networks via Inter-neuron Factoring," in *Proc. Work. Conf. Verified Softw., Theories, Tools, Exp.* Cham, Switzerland: Springer, 2018, pp. 279–290.
- [156]. G. Katz et al., "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proc. Int. Conf. Comput.-Aided Verification*. Springer, 2017, pp. 97–117.
- [157]. X. Huang et al., "Safety verification of deep neural networks," in *Proc. Int. Conf. Comput.-Aided Verification*. Cham, Switzerland: Springer, 2017, pp. 3–29.
- [158]. E. Clarke et al., "Counterexample-guided abstraction refinement for symbolic model checking," *J. ACM*, vol. 50, no. 5, pp. 752–794, Sep. 2003.
- [159]. L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Proc. Int. Conf. Comput.-Aided Verification*. Berlin, Heidelberg: Springer, 2010, pp. 243–257.
- [160]. L. Pulina and A. Tacchella, "Challenging SMT solvers to verify neural networks," *AI Commun.*, vol. 25, no. 2, pp. 117–135, 2012.
- [161]. K. Scheibler et al., "Towards verification of artificial neural networks," in *Proc. MBMV*, 2015, pp. 30–40.
- [162]. *Gurobi Optimizer*. Accessed: 14 February 2020. [Online]. Available: <https://www.gurobi.com/>
- [163]. *IBM ILOG CPLEX Optimization Studio*. Accessed: 14 February 2020. [Online]. Available: <https://www.ibm.com/analytics/cplex-optimizer>
- [164]. G. Singh et al., "Fast and effective robustness certification," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 10802–10813.
- [165]. G. Singh et al., "An abstract domain for certifying neural networks," in *Proc. ACM Program. Lang.*, vol. 3, 2019, pp. 1–30.
- [166]. S. Wang et al., "Efficient formal safety analysis of neural networks," in *Proc. Adv. Neural Inf. Process. Syst. Montréal, Canada: Curran Associates, Inc.*, 2018, pp. 6367–6377.
- [167]. I. E. Grossmann, "Review of nonlinear mixed-integer and disjunctive programming techniques," *Optim. Eng.*, vol. 3, no. 3, pp. 227–252, 2002.
- [168]. C.-H. Cheng, G. Nuhrenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *Proc. Int. Symp. Automated Technol. Verification Anal.* Springer, 2017, pp. 251–268.
- [169]. A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward ReLU neural networks," 2017, *arXiv:1706.07351*. [Online]. Available: <https://arxiv.org/abs/1706.07351>
- [170]. P. Kouvaros and A. Lomuscio, "Formal verification of CNN-based perception systems," 2018, *arXiv:1811.11373*. [Online]. Available: <https://arxiv.org/abs/1811.11373>
- [171]. E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Comput. Surv.*, vol. 28, no. 4, pp. 626–643, Dec. 1996.
- [172]. C. Murphy, P. Gray, and G. Stewart, "Verified perceptron convergence theorem," in *Proc. 1st ACM SIGPLAN Int. Workshop Mach. Learn. Program. Lang. (MAPL)*. New York, NY, USA: ACM, 2017, pp. 43–50.
- [173]. F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," Cornell Aeronaut. Lab, Buffalo, NY, USA, Tech. Rep. AD0256582, 1961.
- [174]. T. Gehr et al., "AI2: Safety and robustness certification of neural networks with abstract interpretation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 3–18.
- [175]. W. Xiang, H.-D. Tran, and T. T. Johnson, "Specification-guided safety verification for feedforward neural networks," 2018, *arXiv:1812.06161*. [Online]. Available: <https://arxiv.org/abs/1812.06161>
- [176]. W. Xiang and T. T. Johnson, "Reachability analysis and safety verification for neural network control systems," 2018, *arXiv:1805.09944*. [Online]. Available: <https://arxiv.org/abs/1805.09944>
- [177]. W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5777–5783, Nov. 2018.

- [178]. G. Singh et al., "Boosting robustness certification of neural networks," in *Proc. Int. Conf. Learn. Represent.*, pp. 1–12, 2018.
- [179]. M. Abdullah Hanif and M. Shafique, "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philos. Trans. Roy. Soc. A*, vol. 378, no. 2164, pp. 1–23, 2020
- [180]. L.-H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," 2019, arXiv preprint arXiv:1912.00941 [Online].

**Muhammad Shafique** has been a Professor of Computer Architecture and Robust Energy-Efficient Technologies (CARE-Tech.), Institute of Computer Engineering, Technische Universität Wien (TU Wien), Vienna, Austria, since November 2016. His research interests are in computer architecture, power-/energy-efficient systems, robust computing, hardware security, brain-inspired computing trends like neuromorphic and approximate computing, hardware and system-level design for machine learning and artificial intelligence (AI), emerging technologies and nanosystems, FPGAs, MPSoCs, and embedded systems. His research has a special focus on crosslayer modeling, design, and optimization of computing and memory systems, and their deployment in use cases from the Internet-of-Things (IoT), cyber-physical systems (CPSs), and ICT for development (ICT4D) domains. Shafique has a PhD in computer science from the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany (2011). He is a Senior Member of the IEEE.

**Mahum Naseer** is currently pursuing a PhD in formal methods for resilient embedded systems at Technische Universität Wien (TU Wien), Vienna, Austria. Her research interests include reliability analysis of systems, error control coding, resilient machine learning systems, and formal methods for system verification. Naseer has a BE in electronics engineering from the NED University of Engineering and Technology, Karachi, Pakistan (2016), and an MS in electrical engineering from the National University of Sciences and Technology (NUST), Islamabad, Pakistan (2018). She is a Student Member of the IEEE.

**Theocharis Theocharides** has been an Associate Professor at the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus, since 2006, where he directs the Embedded and Application-Specific

Systems-on-Chip Laboratory. He has also been a Faculty Member of the KIOS Research and Innovation Center of Excellence, University of Cyprus, since the Center's inception in 2008. His research focuses on the design, development, implementation, and deployment of low-power and reliable on-chip application-specific architectures, low-power VLSI design, real-time embedded systems design, and exploration of energy-reliability tradeoffs for systems on chip and embedded systems. His focus lies on acceleration of computer vision and artificial intelligence algorithms in hardware, geared toward edge computing, and in utilizing reconfigurable hardware toward self-aware, evolvable edge computing systems. Theocharides has a PhD in computer engineering from Penn State University, State College, PA, working in the areas of low-power computer architectures and reliable system design with emphasis on computer vision and machine learning applications. He is a Senior Member of the IEEE and a member of CEDA and the ACM.

**Christos Kyrkou** is a Research Associate with the KIOS Research Center for Intelligent Systems and Networks, University of Cyprus, Nicosia, Cyprus. His research interests include real-time embedded systems, field-programmable gate arrays and reconfigurable hardware, computer vision, machine learning, and smart camera networks. Kyrkou has a BSc, an MSc, and a PhD in computer engineering from the University of Cyprus (2008, 2010, and 2014, respectively). He is a Member of the IEEE.

**Onur Mutlu** is a Professor of computer science at ETH Zürich, Zürich, Switzerland. He is also a Faculty Member at Carnegie Mellon University, Pittsburgh, PA, where he previously held the William D. and Nancy W. Strecker Early Career Professorship. His current broader research interests are in computer architecture, computing systems, hardware security, robust systems, and bioinformatics. He is especially interested in interactions across domains and between applications, system software, compilers, and microarchitecture, with a major current focus on memory and storage systems. A variety of techniques he, together with his group and collaborators, has invented over the years have influenced industry and have been employed in commercial microprocessors and memory/storage systems. Mutlu has a BS in computer engineering and psychology from the University of Michigan, Ann Arbor, MI, and an MS and a PhD in electrical and computer engineering from the

University of Texas at Austin, Austin, TX. He is a Fellow of the ACM and the IEEE.

**Lois Orosa** held a postdoctoral position in the SAFARI Research Group, ETH Zürich, Zürich, Switzerland. His current research interests are in computer architecture, hardware security, memory systems, and machine learning (ML) accelerators. Orosa has a PhD from the University of Santiago de Compostela, Santiago, Spain.

**Jungwook Choi** is currently an Assistant Professor at Hanyang University, Seoul, South Korea. His research

interests include high-performance, energy-efficient, and reliable implementation of machine learning and deep learning algorithms. Choi has a PhD in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Champaign, IL.

■ Direct questions and comments about this article to Muhammad Shafique, Technische Universität Wien (TU Wien), 1040 Vienna, Austria; muhammad.shafique@tuwien.ac.at and Theocharis Theocharides, University of Cyprus, 1678 Nicosia, Cyprus; ttheocharides@ucy.ac.cy.