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

Muhammad Shafique[1], Mahum Naseer[1], Theocharis Theocharides[2], Christos Kyrkou[2],
Onur Mutlu[3], Lois Orosa[3], Jungwook Choi[4]
[1]Technische Universität Wien (TU Wien), Vienna, Austria,
[2]University of Cyprus, Cyprus
[3]ETH Zürich, Zürich, Switzerland,
[4]Hanyang University, South Korea
Corresponding Authors' Email: muhammad.shafique@tuwien.ac.at, ttheocharides@ucy.ac.cy

***Abstract*—Machine Learning (ML) techniques have been rapidly adopted by smart Cyber-Physical Systems (CPS) and Internet-of-Things (IoT) due to their powerful decision-making capabilities. However, they are vulnerable to various security and reliability threats, at both hardware and software levels, that compromise their accuracy. These threats get aggravated in emerging edge ML devices that have stringent constraints in terms of resources (e.g., compute, memory, power/energy), and that therefore cannot employ costly security and reliability measures. Security, reliability, and vulnerability mitigation techniques span from network security measures to hardware protection, with an increased interest towards formal verification of trained ML models.**

This paper summarizes the prominent vulnerabilities of modern ML systems, highlights successful defenses and mitigation techniques against these vulnerabilities, both at the cloud (i.e., during the ML training phase) and edge (i.e., during the ML inference stage), discusses the implications of a resource-constrained design on the reliability and security of the system, identifies verification methodologies to ensure correct system behavior, and describes open research challenges for building secure and reliable ML systems at both the edge and the cloud.

I. Introduction

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 CPS 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 cyber-physical systems.

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 cyber-physical systems is 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 Machine Learning (ML), especially in Deep Learning, evolved from the concept of a single-layer neural network, the perceptron [2], to the Multi-Layer Perceptron (MLP) [2] and the current intricate multi-layer Deep Neural Networks (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 re-learning [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 like autonomous driving [12], intelligent transportation [13] and health-care [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-labelled 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, 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. Fig. 1 summarizes both the

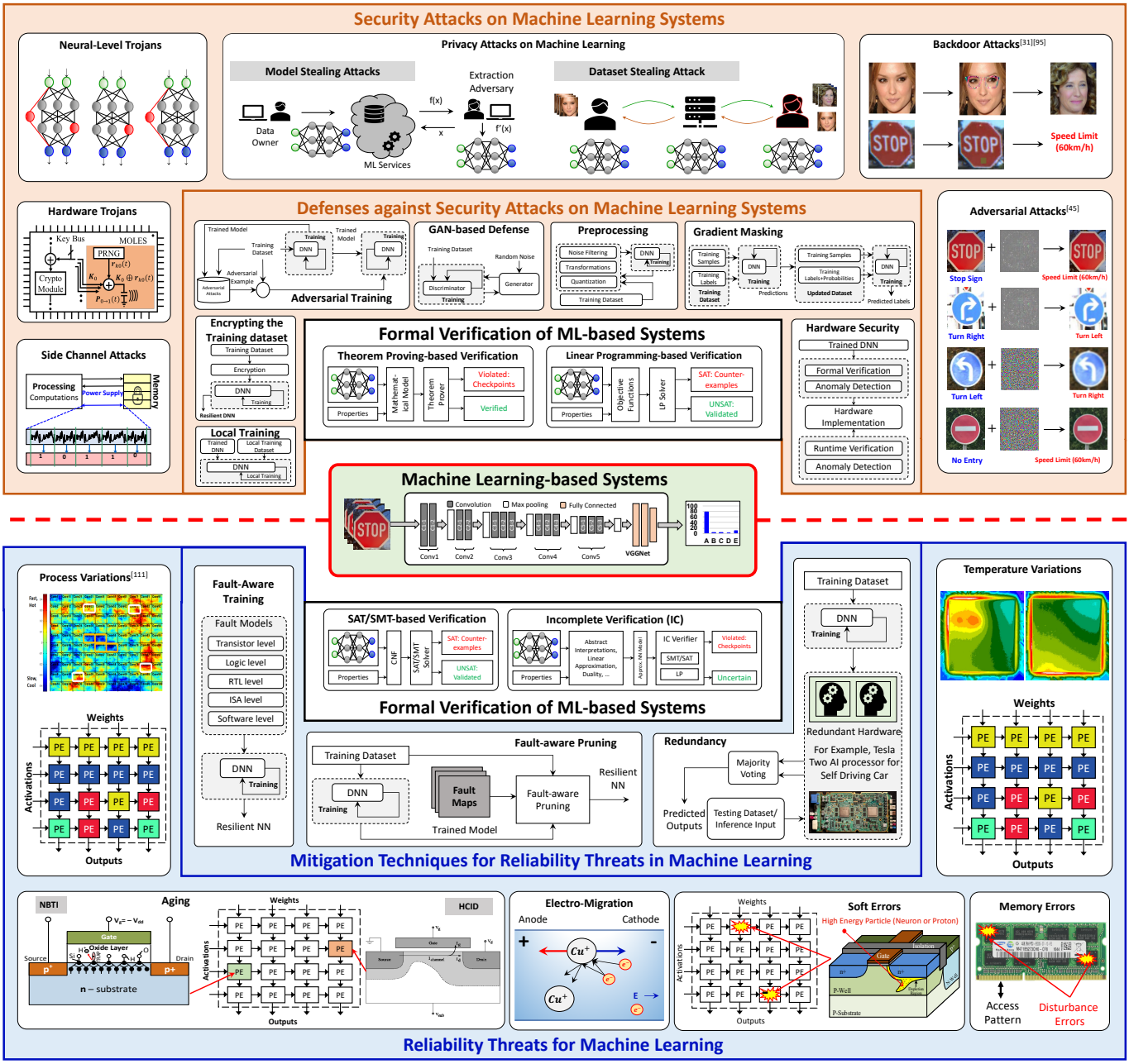


Fig. 1: Overview of threats and challenges associated with ML-based systems: reliability threats and corresponding mitigation techniques (bottom), and security attacks and corresponding defenses (top).

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 I.

The rest of the article is organized as follows (see Fig. 2):

- Section II provides a taxonomy of ML that categorizes different network types, explains the ML design cycle, and introduces basic concepts about security and reliability.
- Section III highlights the various attack strategies that jeopardize the integrity of ML systems, particularly at the cloud and edge levels, explains how these attacks are implemented, and identifies mechanisms that can lead to the mitigation of the attacks.
- Section IV describes the defenses against the security attacks to secure ML systems, and states the shortcomings of these defenses.
- Section V elaborates on the reliability concerns that reduce

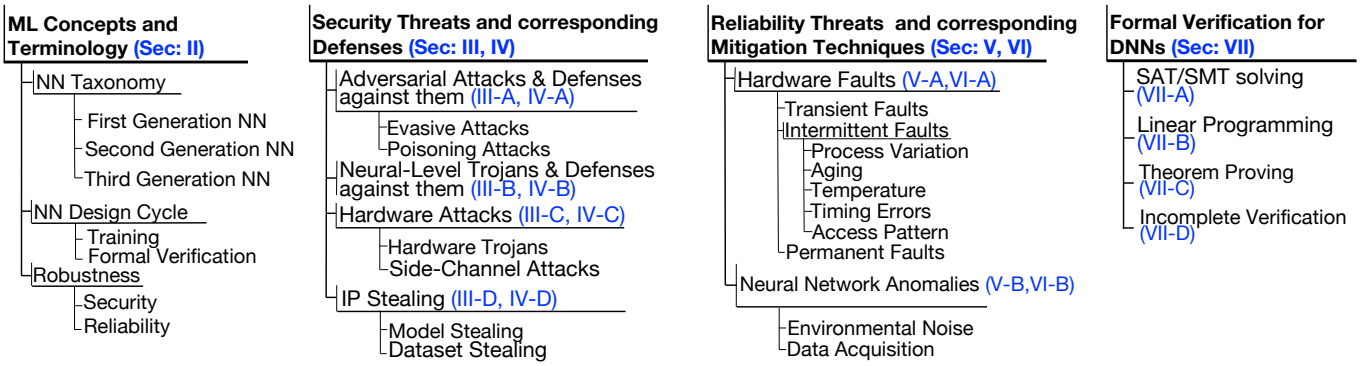


Fig. 2: Organization of the paper.

TABLE I: 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
Neural Network	NN
Recurrent Neural Network	RNN
Satisfiability Modulo Theories	SMT
Satisfiability solving	SAT solving
Spiking Neural Network	SNN

the accuracy of ML systems, specifically when deployed at the edge, in the absence of an explicit attacker.

- Section VI discusses the state-of-the-art techniques to mitigate the impact of the reliability issues in ML, and their corresponding limitations.
- Section VII elaborates on the use of formal methods for Neural Network (NN) verification, presents the various types of verification techniques and their use for DNN verification in state-of-the-art, and explains the reasons for their limited success.
- Section VIII discusses the open research challenges and perspectives towards designing secure and reliable ML systems.

II. Machine Learning: 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 of 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 of *neurons/nodes*, which connect to other neurons in the corresponding layers via a non-linear *activation* function. Each neuron has its associated parameters i.e., weight, bias, and/or filter coefficient. A detailed overview of neural networks can be found in [16][17].

A. Neural Network Taxonomy

If the input propagates through the network in only one direction, the network is said to be *feed-forward*. If there are feedback loops in the network, the network is called a *recurrent* neural network (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 (see Fig. 3). The details of the NN types of each generation can be found in Appendix IX.

First Generation of NNs: The first generation of NNs [20] is comprised of single-layer and multi-layer perceptrons (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 neural networks (CNN)*, which make use of convolutional layers comprising of convolutional filters to extract important features from the input, while providing a certain degree of shift invariance to the network [22]. A convolutional layer typically uses continuous non-linear 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. This generation of NNs are increasingly being deployed in practical ML systems.

A relatively new approach to solve the problem of information loss in CNNs is the use of *capsule networks (CapsNets)*

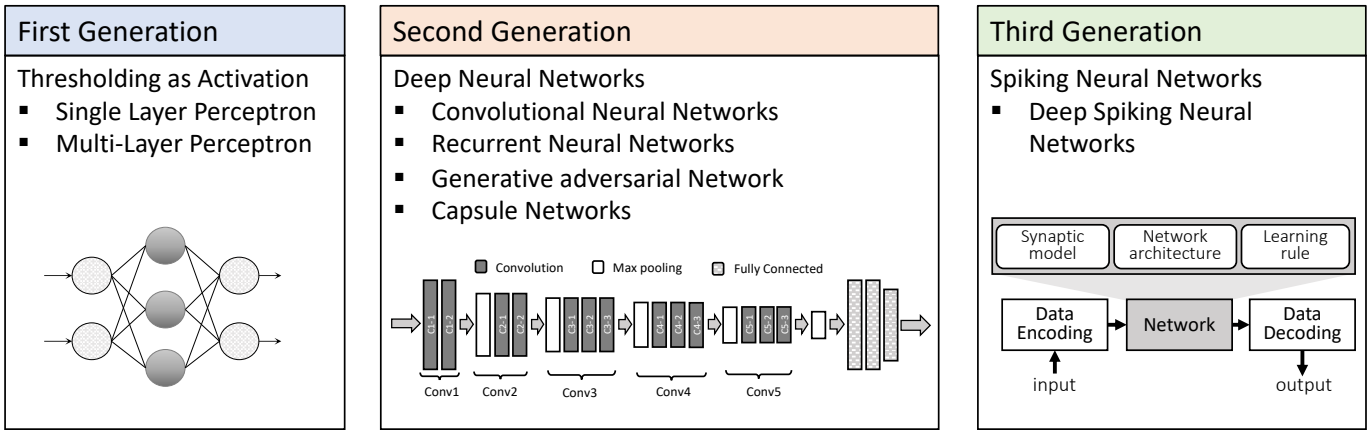


Fig. 3: Summary of Neural Network models proposed over time.

[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 towards 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 Neural Networks (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 spatio-temporal characteristics of the biological neurons.

B. Neural Network Design Cycle

Fig.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, while inference is typically performed at the edge in real-world Smart CPS systems (e.g., autonomous vehicles and wearable healthcare devices). In certain IoT/CPS systems thT 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 dataset* to find suitable values for the network parameters. Training is composed of a forward-pass and a backward-pass. The forward-pass calculates 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 dataset, is called an *epoch*. The overall training process of a 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 dataset, i.e., the validation dataset. 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 parameters are again updated using the training dataset until either the process reaches the maximum number of epochs (or cycles), or the network reaches the desired level of accuracy with the validation dataset.

The most common way to check the final inference accuracy of a trained network is to use a *testing dataset*. 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 Fig. 5, the guarantees provided by verification are valid for the entire (infinite) input domain whereas those provided by testing

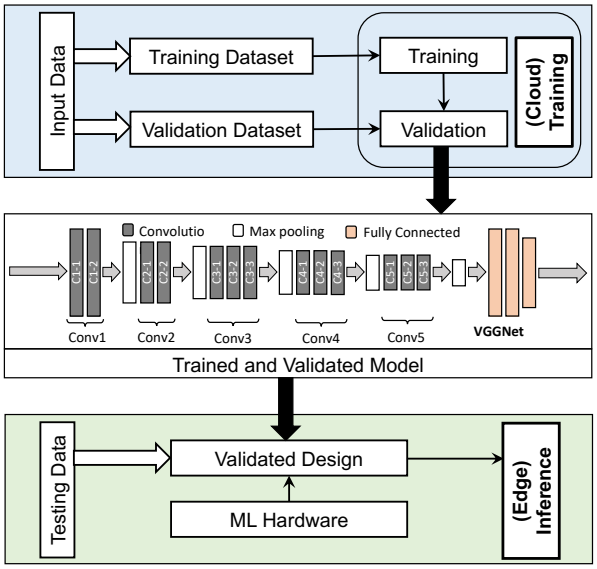


Fig. 4: Design cycle of a NN-based ML system.

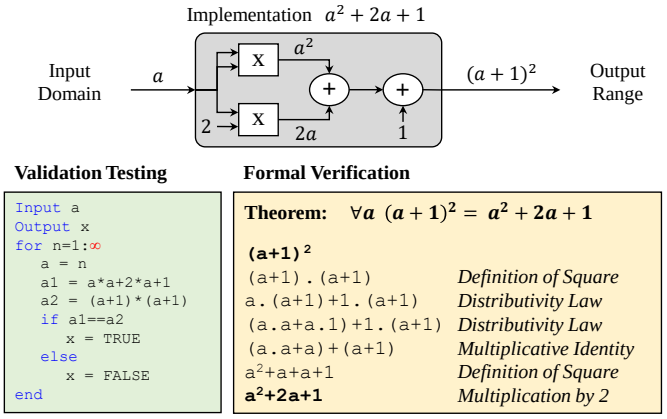


Fig. 5: Comparison between testing and verification for a small hypothetical system: ensuring behavioral correctness of the system for all possible inputs is not always feasible with testing.

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 Fig. 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 carried out at the edge of the IoT/CPS system, hence exposing the system to numerous security and reliability concerns during the operations under varying scenarios and harsh environmental conditions.

C. 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

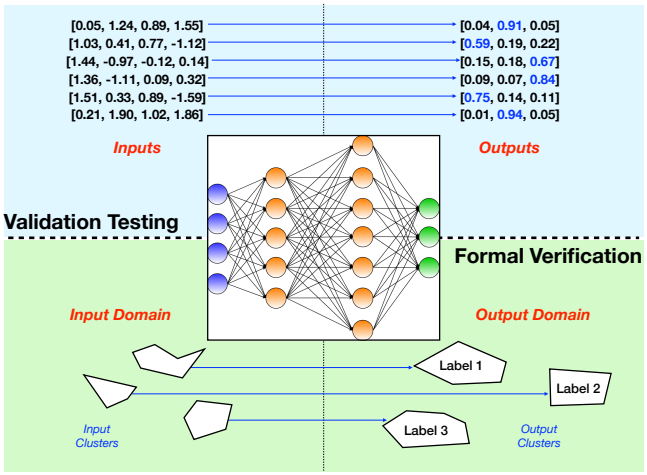


Fig. 6: Comparison between testing and verification for a neural network based system: Verification is intended to determine whether the bounded inputs are reachable to the correct output bounds.

conditions, and the accuracy of DNN outputs in the presence/absence of input or network alterations. This can be divided into two sub-properties: security and reliability [29]. The DNN is said to be *secure* against an attack if the attacker cannot steal information (via 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 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.

III. 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 II).

A. Adversarial Attack

Since their discovery, adversarial attacks [26] have been a widely studied DNN security threat [42][43][44]. In an adversarial attack, the known DNN parameters are exploited to minimize the cost function corresponding to noise patterns δx, which, when added to the input x, can cause *misclassification*, as shown in Fig. 7. The noise added is usually imperceptible, making the task of distinguishing between clean and malignant inputs nearly impossible. This can be represented formally as:

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

TABLE II: 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]

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*.

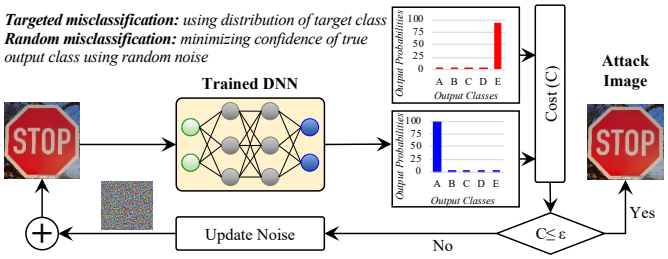


Fig. 7: Adversarial Attack on a trained DNN: an adversarial attack can result in the misclassification (either targeted or random) of traffic sign boards, which is a concern in autonomous driving [46].

Adversarial attacks can be categorized as either *evasive* or *poisoning* [47], depending on the access of the attacker to the DNN design cycle. In **evasive attacks**, the attacker has no access to the DNN training process and training dataset. The attack is solely configured during the DNN inference stage, using either input gradients, output probability vectors, or the output decision [48]–[54]. For instance, the Fast Sign Gradient Method (FSGM) [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) [55], 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 [56][57] make use of input gradients to propose adversarial attacks. TrISec [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 structural similarity index) in the DNN training algorithm. Works like [58][48] make use of output labels to determine attacks in close proximity to the classification boundary.

In **poisoning attacks** [59], the attacker has access to the training dataset/training procedure. The attack is implanted in the DNN during training by feeding the network with malicious training data. Fig. 8 shows two example poisoning attacks that increase the probability of misclassification of an stop signal (red bars). The data could be poisoned with tailored noise [32][60], 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 signify that the network has the capacity to learn more. Hence, such networks can be trained on poisoned data (as shown in Fig. 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 [61].

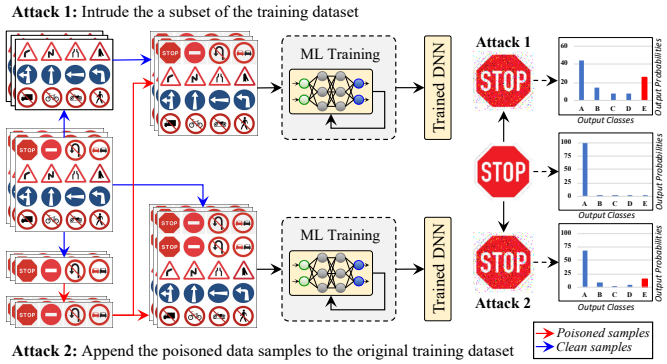


Fig. 8: Classification accuracy of DNN trained on a poisoned dataset.

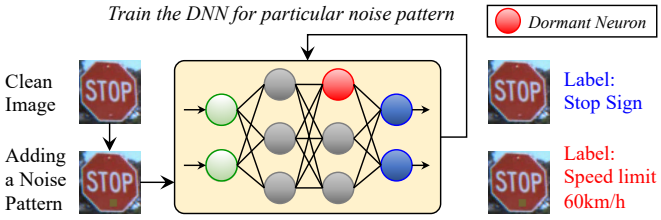


Fig. 9: Effect of a backdoor on DNN accuracy. The dormant neurons (red) learn to associate the backdoor with a targeted misclassification label.

For most of the adversarial attacks, a common inadequacy is to ignore the pre-processing filtering stage in an ML system

[31]. The pre-processing 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.

B. Neural-Level Trojans

Another class of attacks, the neural-level trojans [62], involves the insertion of additional neurons into a pre-trained 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 re-train the network and use complex internal triggering mechanisms.

C. Hardware Attacks

Hardware trojans [63][64][65][66][67] are malicious components implanted into the system hardware, which compromise the security of a 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, thus 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 [68], to ensure a successful stealthy attack.

Side channel attacks, as shown in Fig. 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], [69]–[72]. 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 [73], [74].

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

D. IP Stealing

Attacks to steal Intellectual Property (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 [75], or

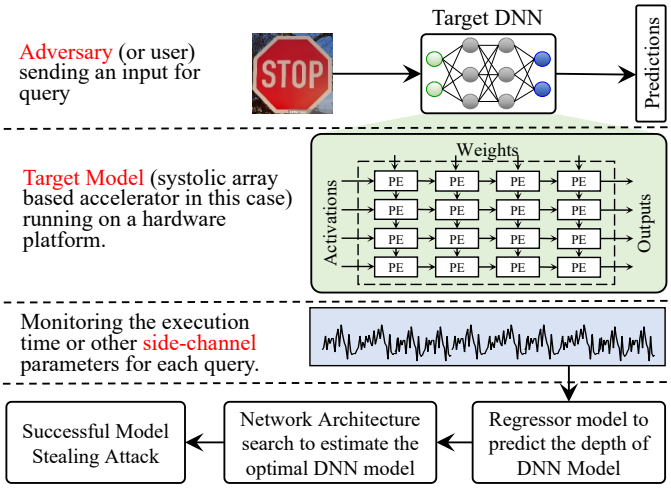


Fig. 10: Side-channel attack based on the execution time of individual input queries, which can be used to decipher the depth of the DNN model and estimate the network parameters/model.

predicting the *data* the DNN was trained on using the available model description (**Dataset Stealing Attack**) [76]. Both types of attacks are shown in Fig. 11. Leaking side-channels of the model, responses of queries to the system, and similar behavioral network characteristics can be exploited, analyzed, and reverse engineered to obtain the underlying IP.

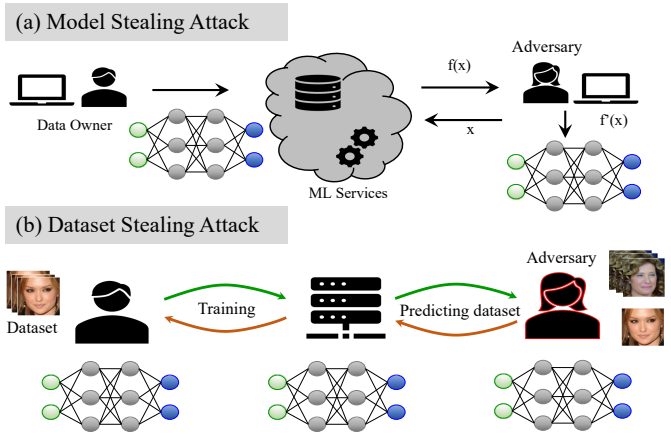


Fig. 11: IP Stealing from a trained DNN: the objective of a stealing attack can either be: (a) to estimate the underlying DNN model, or (b) predict the dataset used for DNN training, using multiple queries.

IV. 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.

A. Defending Against Adversarial Attacks

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

As shown in Fig. 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.

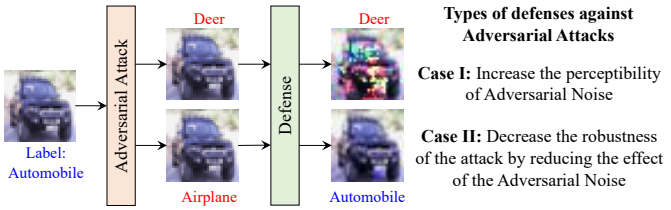


Fig. 12: Defenses against adversarial attacks either increase the perceptibility of adversarial noise (Case I) or decrease the effect of the adversarial noise (Case II).

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 Fig. 13, reduces the dependability of output classification by retraining the DNN with the output probability vector. **Adversarial training**, as shown in Fig. 14(a), is another commonly used defense [77][78], 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 pre-processing** [31]. This defense smooths out, transforms and truncates the noise before it is even fed to the DNN. As shown in Fig. 14(b), 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** [79]. 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 datasets is robust against adversarial attacks.

Most of the above defenses may work against a naive attack. However, for a strong attack, the defenses may fail. Many studies show that gradient masking does not increase the robustness of a DNN [80]–[82], and hence can be broken with the use of a substitute model to identify the approximate gradient direction [83]. Pre-processing aware attacks [31] can break the filtering defense. Likewise, as studied by several works [84][85], 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.

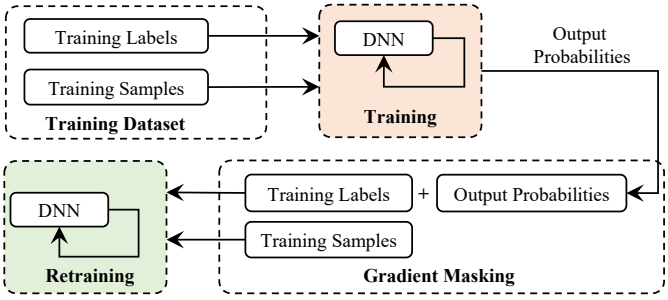


Fig. 13: Using Gradient Masking to hide the input gradients that might be used by the attacker to determine the perturbations that need to be inserted to perform the Adversarial Attack.

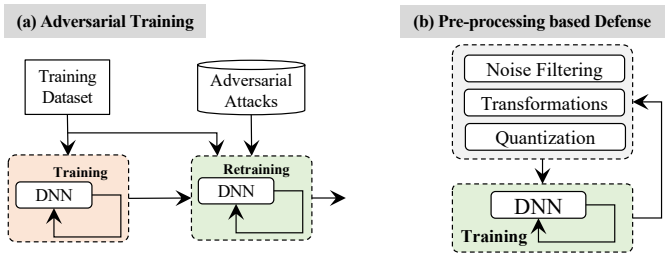


Fig. 14: (a) Improving a DNN's accuracy in the presence of a known attack by training the dataset with Adversarial Examples obtained from known adversarial attacks, i.e., Adversarial Training. (b) Reducing the effects of adversarial noise added to the input via input pre-processing techniques such as noise filtering, quantization and other input transformations.

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, 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 [86]–[88]. 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 not be vulnerable to the adversarial attacks anymore. However, the question of how the knowledge of these bounds can be used to actually prevent adversarial attacks is yet to be answered.

B. 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 pre-processing** [31] to smooth out the input trigger, input **anomaly detection** [89] to identify suspicious input patterns, and **prediction distribution** [60] to identify the bias of DNN towards the targeted output. Since trojans are inserted into pre-trained 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.

C. Defending Against Hardware Attacks

Hardware trojans [63], [90]–[93] are a hardware-related security problem in ML systems. A hardware trojan is a malicious modification of a 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. Fig. 15 demonstrates the use of Binary Decision Diagrams (BDDs) for equivalence checking [94] 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 [63][64].

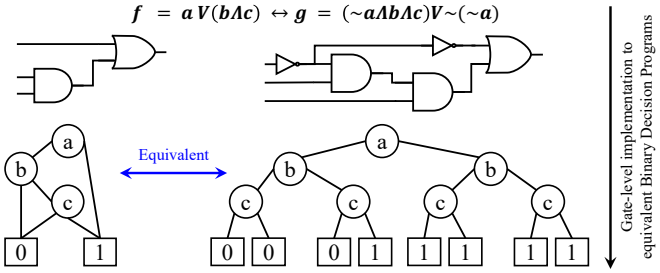


Fig. 15: Using Binary Decision Diagrams (BDDs) for hardware equivalence checking.

Other potential defenses against hardware trojans include **side-channel analysis** [39] for anomaly detection, and cross-layer attack modeling via bridging the gap between the hardware and software [38]. 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 [68].

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 rely 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 a NN, could potentially make the inference of useful knowledge from side-channels more difficult [95][40][41].

D. 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 [96][97]. In practice, these approaches may not work well as they may not be strong enough [98]. 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 dataset [99], verifying the training procedure [100], ensuring privacy after training by network transformation [101], obfuscating defenses against reverse engineering-based attacks [102], [103], and isolating the hardware accelerators [104].

To protect IP against *piracy*, the **rounding** approach [105] 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].

V. 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, which lead to reduced ML system reliability.

A. Hardware Faults

Errors in the hardware components that build up a system are generally classified into transient, intermittent, and permanent faults [106][107]. 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.

1) Transient Faults: The nature of applications where the ML systems are deployed expose 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 [108]. Transient faults can be random, i.e., occurring unpredictably, or non-random, i.e., can be reproduced under certain circumstances [106]. Electrostatic discharge, electromagnetic radiation, noise in hardware interconnections, or flaw(s) in fabrication are among the leading factors contributing to transient faults [109][107].

Soft errors [108] are a type of transient fault, mostly caused either by 1) a high-speed proton strike from cosmic rays, or 2)

the emission of an alpha particle from impurities in IC packaging. Both particles generate a charge Q_{rad} in the chip, and if this charge exceeds a certain threshold value Q_{th}, it is likely to change the state of the chip, 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 [110], [111]. With increasing technology miniaturization, such bit-flips can extend to multiple bits within a single data word [111], [112]. This phenomenon, termed as a *Multiple Bit Upset (MBU)*, poses a challenge to robust machine learning. These effects may lead to misclassification in ML systems, as shown in Fig. 16(a).

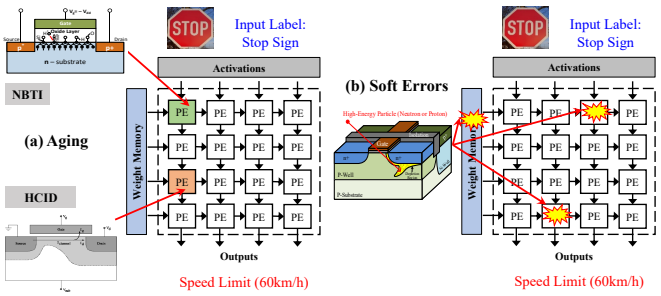


Fig. 16: Effects of reliability threats, i.e., (a) soft errors, and (b) aging on ML systems.

2) Intermittent Faults: Such faults are intermittent and relatively unpredictable, which make them difficult to repeat, analyze, and understand. **Process variation** [29][113] 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 [106]. Similarly, **aging** [114] (see Fig. 16(b)) 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 [115] 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, it may also make the ML model vulnerable to serious security concerns [116].

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 [117], [118], which is the phenomenon that repeatedly accessing a row in a modern DRAM chip causes disturbance errors in physically-adjacent rows. DRAM data **retention failures** [119]–[122] can also cause intermittent and unpredictable faults due to DRAM variable retention time and data pattern dependence.

3) Permanent Faults: These faults are irreparable, where the system portrays fixed/repetitive errors like *stuck-at* faults. Factors contributing to permanent faults include **cosmic radiation**, **electrostatic discharge (ESD)** in device, **fabrication flaws** [106], [109], [123], or recurring intermittent faults.

B. Neural Network Anomalies

Environmental noise (EN) [124][125] has a similar impact on edge devices as adversarial attacks have on DNNs.

$$f(x) \neq f(x + EN) \tag{2}$$

For instance, for an object classification system, possible environmental noise could be due to fog or pollution in 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* [77][126] can impact the decision-making of the vehicle and may lead to serious consequences [127].

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 2D space in Fig. 17 (top). The inputs closer to these boundaries are vulnerable, and slight input changes, even in the absence of a malicious attacker, may lead to misclassification.

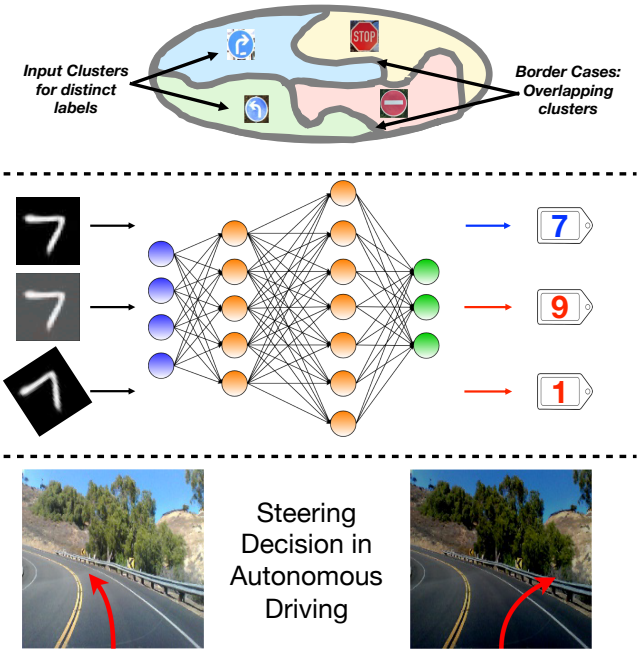


Fig. 17: Inputs close to cluster boundaries (top) in hyperspace are most vulnerable to environmental adversarial transformations. Variation during data acquisition (middle) can cause misclassification, which can lead to drastic effects in ML systems (bottom) [77].

VI. Mitigation techniques for reliability threats in ML systems

This section discusses several mitigation techniques for the reliability threats in ML systems discussed in Section V.

A. Mitigation Techniques for Hardware Faults

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

1) 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 [128], using additional circuitry for **error detection** [129], **scrubbing** to periodically remove errors to prevent error accumulation [121], [130], adding **hardware redundancy** and voting mechanisms [131] to rule out the erroneous bits, and using **error detection and correction codes** [108], [132]–[134], 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 is also being used to ensure safety in ML systems. For instance, Tesla's self-driving car computer has two chips deployed to tolerate faults [135].

Numerous approaches are available to handle transient errors; yet, all these approaches provide a trade-off between error detection/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 [136]. This finding could be used for system design with stronger error correction mechanisms deployed for more critical bit flips.

2) 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 [137] 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 [117] and various other hardware or software techniques [118]. Online profiling of memory cells [119]–[121], [138]–[140] 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** [141][142][143]. 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** [144][145]. Such timing verification approaches are intended to ensure that the system behaves correctly within the defined timing bounds.

3) 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 [146], [147], 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 [110], [148]–[151]. In **fault-aware training**, the DNN is trained for different faults at multiple levels, like transistor level and logic level, as shown in Fig. 18(a). This is a computational greedy solution. In **fault-aware pruning**, all the DNN connections and parameters that map to faulty processing elements/nodes are pruned using fault maps of the baseline hardware (e.g., systolic array-based accelerator), as shown in Fig. 18(b). In **fault-aware mapping**, the saliency of DNN parameters is exploited to define a mapping of different segments of the DNN. This mapping is then used to prune the ineffectual parameters of the DNN while retaining the salient parameters. In **activation clipping**, the activation values exceeding the predefined threshold for a fault-free neural networks are clipped. This eliminates the need for either pruning or retraining.

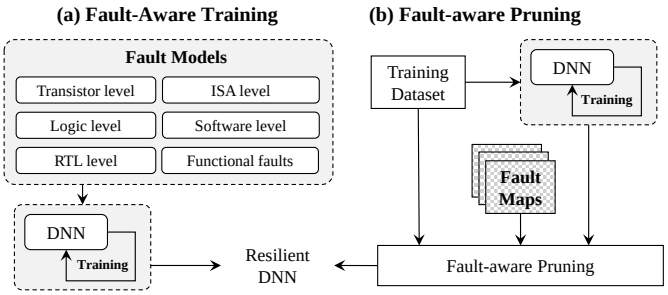


Fig. 18: Mitigation techniques for permanent faults in ML systems

B. Mitigation Techniques against Environmental Noise

Similar to defenses against adversarial attacks (Section IV-A), **pre-processing filters** [31] can reduce the effects of environmental noise in DNNs. Likewise, **adversarial training** [78] 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 [84]. Since the accuracy of ML systems in the presence of environmental noise and varying input data arise 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 dataset. However, it is not always possible to obtain a large and diverse input dataset. To overcome this limitation, some works propose the generation of **synthetic datasets** [152][153][154][155]. Yet, real input domains are

mostly very large, multi-dimensional, and 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.

VII. Formal Verification for Robust ML

As briefly highlighted in Section II, testing a trained DNN using a labelled dataset is insufficient to ensure 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[156], using saliency maps to understand DNN inference [157], and developing various certifiability criteria for DNN interpretability [158][159][160].

Formal verification provides an orthogonal alternative to testing that provides formal/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 neural networks, which forms an essential component of the ML system, has been an active domain of research for only a decade. Fig. 19 summarizes the major milestones reached in NN verification over time, according to the four major verification categories: SAT/SMT solving, linear programming, theorem proving, and incomplete verification.

A. SAT/SMT

Satisfiability (SAT) checking is the branch of formal verification where the system model and the property to be verified for the system are expressed in propositional logic, and written into Conjunctive Normal Form (CNF), as shown in Fig. 20 (bottom). The formula is then checked by an automatic SAT solver. Having a *SAT* output implies that a satisfying solution to the negation of the property, i.e., a counter-example, has been found. An *UNSAT* output implies the absence of any counter-example, and hence indicates that the stated property holds for the system. Satisfiability Modulo Theories (SMT) is a variant of SAT that works similar to SAT solving, as shown in Fig. 20 (top), but allows the use of theories beyond 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 Neural Networks (BNNs) [86][161]. SMT solvers, on the other hand, are preferred for verifying DNNs with real and/or integer network parameters [162][163]. Another concept often associated with SAT-based verification approaches is Counter-Example-Guided based Abstraction Refinement (CEGAR) [164], 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 over-approximation [165].

However, SAT-based verification suffers from the scalability problem: state-of-the-art techniques are capable of verifying only small networks [166][167], comprising of less than 10 neurons, to medium-sized networks [163], comprising of up to $20,000$ neurons. Although some works propose optimizations, like K-factoring [161], 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 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 [162] are not scalable to larger networks. More powerful SAT/SMT solvers could be key for the improvement of DNN verification.

B. Linear Programming

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 Fig. 21. The objective function can be either a minimization or a maximization function. The search of the minima/maxima is automatic, and involves the use of linear programmers [168][169].

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 [85][88].

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 non-linear activation functions. Some works [170][171], as will be discussed in Section VII-D, replace non-linear 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 non-linear activation function. Another approach, proposed for ReLU-based networks, is input bisection for selected network nodes [172]. ReLU is a piecewise linear function that works like a half-way rectifier: output is zero if the input is negative but output equals the input for all non-negative 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[1] is proposed in several recent works [59], [87], [174]–[176]. Although the approach ensures reliable verification results, without a significant increase in the size of the problem, it also suffers from the scalability

[1]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 [173] for details of the technique.

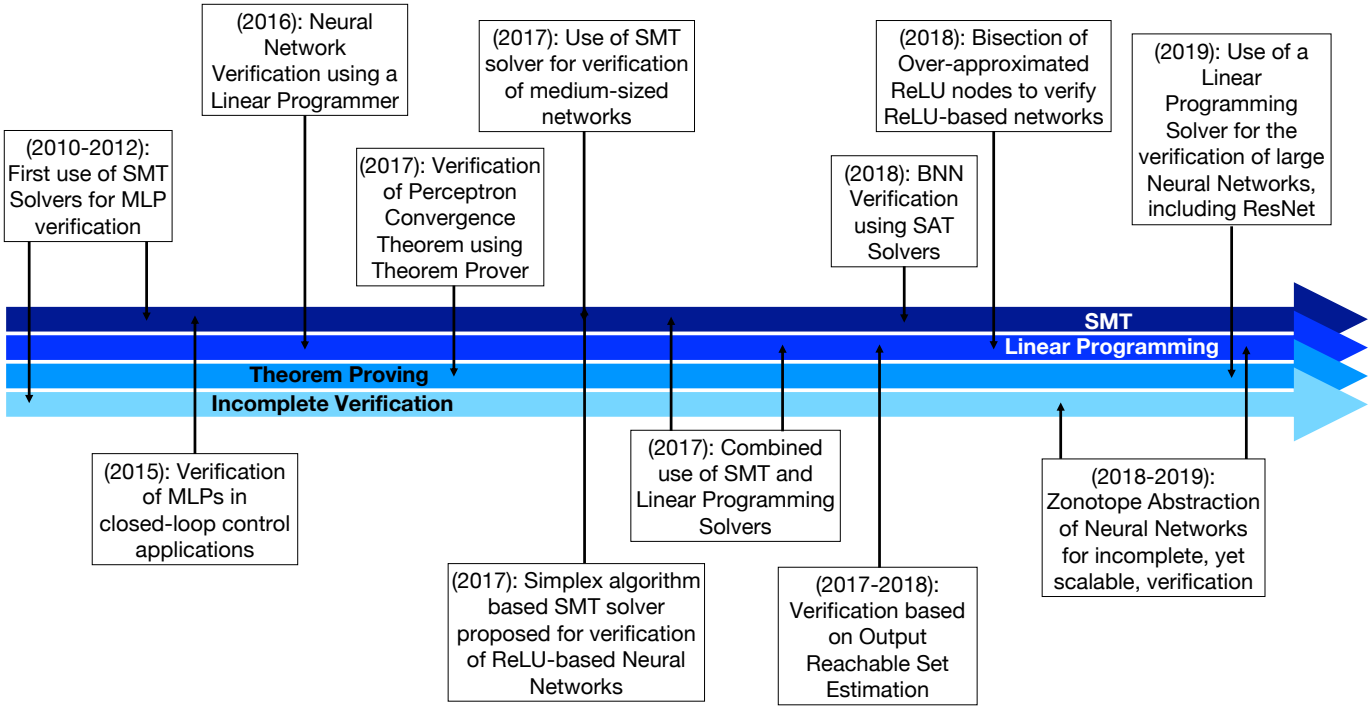


Fig. 19: A decade of Verification Techniques for Neural Networks

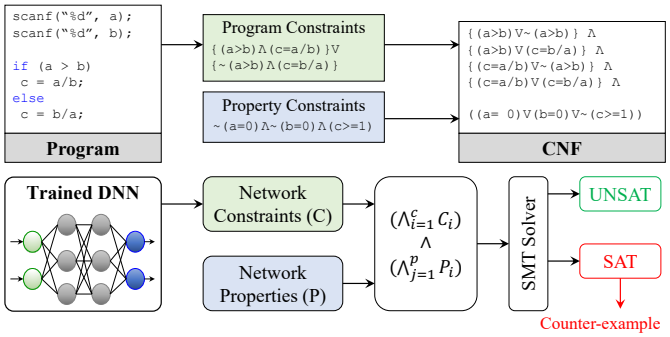


Fig. 20: Using an SMT solver for verification. CNF expresses program and property constraints of C code (top), and SAT/SMT solver for a DNN-based system (bottom).

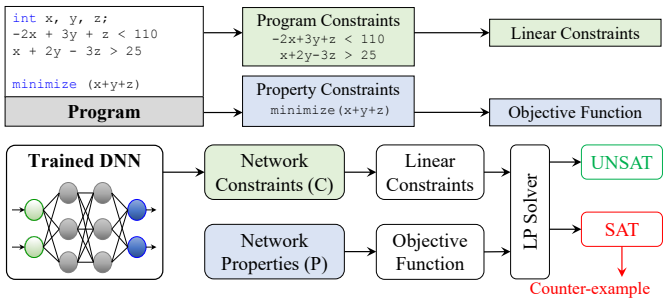


Fig. 21: Using a Linear Programmer to define the linear constraints and the objective function (top), and verification of a DNN-based system with a Linear Programmer Solver (bottom)

problem. Reducing the number of constraints by eliminating the inactive neurons [88], and exploiting the sparsity of practical DNNs may allow effective verification of practical-sized ML systems.

C. 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 [177]. The verification example demonstrated in Fig. 5 shows how natural deduction based reasoning works. Fig. 22 gives a more generic view of how theorem proving works. Generally, for 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 verification of such systems is done via interactive theorem proving.

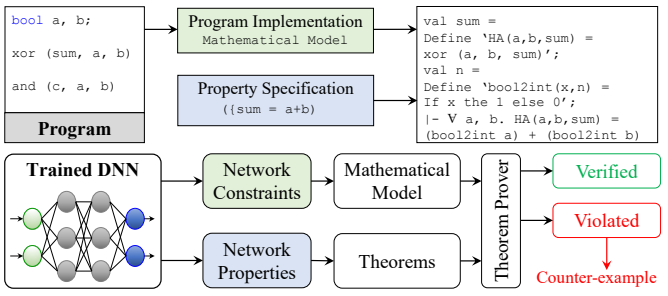


Fig. 22: Using Theorem prover for verification of a half-adder (top), and mathematical model and theorems of Theorem Proving for a DNN-based system (bottom).

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 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. [178] verifies the perceptron convergence theorem, but the work focuses on a very small subset of DNNs called binary classifiers, and hence 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][179] was proposed almost six decades before it was formally verified by [178]. Hence, understanding and developing the theory behind DNN operation seems to be a logical step before theorem proving could be successfully employed for DNN verification.

D. 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 [180]–[182]. As a result, the system model is not an exact representation of the actual system but rather an over-approximation. Verification is then performed on this approximate model, as shown in Fig. 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 [170][171]. To improve the completeness of verification, we can use abstraction refinement approaches like CEGAR [164][165]. 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 [170][171]. Whenever the incomplete verifier provides counter-examples, they are actual scenarios where the property

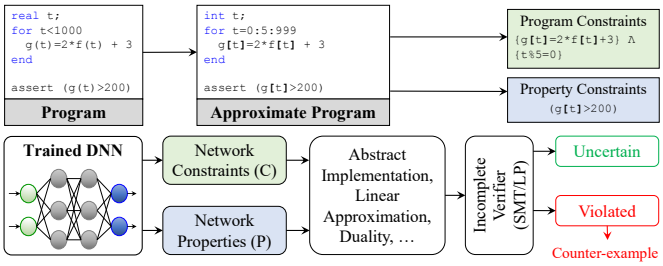


Fig. 23: Using Incomplete Verification for 1) verifying a continuous-domain program (top), and 2) verifying a DNN-based system (bottom).

does not hold for the system. If the verifier provides no counter-examples, the system may still be unsafe or the property being verified may still not hold for some inputs to the system [183].

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 [184]. This can be accomplished by iteratively refining the network model until it matches the exact system model [164], or combining incomplete verification with other complete verification approaches like SMT solvers or LP.

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

1) *Securing training datasets* before outsourcing them for training. This may involve encrypting the training dataset from the cloud servers to ensure IP privacy, or minimizing the impact of data poisoning attacks.
2) *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.
3) *Ensuring fairness of training*, i.e., preventing bias of the trained NN. Gradient-based adversarial input generation

and counter-examples generated via formal verification of NNs can be used to identify 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 towards.

4) *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.
5) *Minimizing the accessibility to side-channel leakages*. This can be achieved by minimizing the sharing of resources like memory and power, hence 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 runs the model are both vulnerable to inconsistencies in performance over their life time. Major unresolved reliability challenges in ML systems include:

1) *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 physical environment, 2) assess the performance of the available mitigation techniques, and 3) analyze the trade-offs between these approaches to identify the solution that ensures the highest system reliability.
2) *Providing a fault-safe runtime* in case of system discrepancies. Currently, such fault-safe techniques include the use of redundancies at hardware and software levels, which ensure that, in 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.
3) *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:

1) *Formally modeling* the non-linear, non-convex behavior of ML systems. Complete verification with existing modelling approaches (e.g., Big-M) is often not the optimal solution due to the large number of generated clauses/constraints. Incomplete verification is also not the optimal solution because it may lead to false positive results due to over-approximation.
2) *Incorporating the uncertainties of the real world into the formal system model*. Namely, the verification of the system under different reliability factors, e.g., environmental noise.
3) *Inspecting system behavior for all possible inputs*. Formal verification is widely acclaimed due 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.
4) *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.
5) *Improving the timing efficiency of verification*, while ensuring the completeness of verification results. There is a trade-off 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 trade-off between timing efficiency and completeness still remains an open challenge.
6) *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.

IX. Conclusion

Machine Learning (ML), particularly Neural Networks (NNs), forms an essential component of modern Cyber-Physical Systems (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, they are vulnerable to numerous security and reliability concerns. This survey highlight 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.

References

[1] N. Mäkitalo, A. Ometov, J. Kannisto, S. Andreev, Y. Koucheryavy, and T. Mikkonen, "Safe, secure executions at the network edge:

coordinating cloud, edge, and fog computing," *IEEE Software*, vol. 35, no. 1, pp. 30–37, 2017.

[2] F. Rosenblatt, "The perceptron–a perceiving and recognizing automation," *Report 85-460-1 Cornell Aeronautical Laboratory, Ithaca, Tech. Rep.*, 1957.

[3] L. Wang, S. Guo, W. Huang, and Y. Qiao, "Places205-vggnet models for scene recognition," *arXiv preprint arXiv:1508.01667*, 2015.

[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[5] G. Anthes, "Lifelong learning in artificial neural networks," *Communications of the ACM*, vol. 62, no. 6, pp. 13–15, 2019.

[6] M. Fink, Y. Liu, A. Engstle, and S.-A. Schneider, "Deep Learning-Based Multi-scale Multi-object Detection and Classification for Autonomous Driving," in *Fahrerassistenzsysteme*. Springer, 2019, pp. 233–242.

[7] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales, "When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition," in *International conference on computer vision workshops*. IEEE, 2015, pp. 142–150.

[8] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *Signal processing magazine*, vol. 29, 2012.

[9] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 206–10 222, 2009.

[10] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," in *SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 371–372.

[11] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "Smart grid technologies: Communication technologies and standards," *Transactions on Industrial informatics*, vol. 7, no. 4, pp. 529–539, 2011.

[12] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[13] Y. Lin, P. Wang, and M. Ma, "Intelligent transportation system (its): Concept, challenge and opportunity," in *International conference on big data security on cloud (bigdatasecurity)*. IEEE, 2017, pp. 167–172.

[14] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature Medicine*, vol. 25, no. 1, p. 24, 2019.

[15] F. Amato, A. López, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel, "Artificial neural networks in medical diagnosis," *Journal of Applied Biomedicine*, 2013.

[16] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 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*, vol. 8604, 1986.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 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," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[22] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, 1990, pp. 396–404.

[23] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 44–51.

[24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[25] J. Vreeken, "Spiking neural networks, an introduction," 2003.

[26] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[27] A. Camilleri, M. Gordon, and T. Melham, "Hardware verification using Higher-Order Logic," University of Cambridge, Computer Laboratory, Tech. Rep., 1986.

[28] V. D'silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008.

[29] F. Kriebel, S. Rehman, M. A. Hanif, F. Khalid, and M. Shafique, "Robustness for Smart Cyber Physical Systems and Internet-of-Things: From Adaptive Robustness Methods to Reliability and Security for Machine Learning," in *Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 581–586.

[30] A. Kurakin, D. Boneh, F. Tramr, I. Goodfellow, N. Papernot, and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," in *International Conference on Learning Representations (ICLR)*, 2018.

[31] F. Khalid, M. A. Hanif, S. Rehman, J. Qadir, and M. Shafique, "FAdeML: Understanding the Impact of Pre-Processing Noise Filtering on Adversarial Machine Learning," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 902–907.

[32] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[33] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 273–294.

[34] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 123–142, 2018.

[35] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[36] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[37] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in *International Conference on Information Security and Cryptology*. Springer, 2012, pp. 1–21.

[38] S. Rehman, M. Shafique, and J. Henkel, *Reliable Software for Unreliable Hardware: A Cross Layer Perspective*. Springer Publishing Company, Incorporated, 2016.

[39] F. Khalid, M. A. Hanif, S. Rehman, and M. Shafique, "Security for Machine Learning-based Systems: Attacks and Challenges during Training and Inference," in *Frontiers of Information Technology (FIT)*. IEEE, 2018, pp. 327–332.

[40] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Computer Security Applications Conference*. ACM, 2018, pp. 393–406.

[41] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[42] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[43] F. Michels, T. Uelwer, E. Upschulte, and S. Harmeling, "On the Vulnerability of Capsule Networks to Adversarial Attacks," *arXiv preprint arXiv:1906.03612*, 2019.

[44] A. Marchisio, G. Nanfa, F. Khalid, M. A. Hanif, M. Martina, and M. Shafique, "Capsattacks: Robust and imperceptible adversarial attacks on capsule networks," *arXiv preprint arXiv:1901.09878*, 2019.

[45] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," *arXiv preprint arXiv:1806.01246*, 2018.

[46] F. Khalid, M. A. Hanif, S. Rehman, R. Ahmed, and M. Shafique, "TrISec: Training Data-Unaware Imperceptible Security Attacks on Deep Neural Networks," in *On-Line Testing and Robust System Design (IOLTS)*. IEEE/ACM, 2019.

[47] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[48] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," *arXiv preprint arXiv:1712.04248*, 2017.

[49] ——, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," *arXiv preprint arXiv:1712.04248*, 2017.

[50] J. Chen and M. I. Jordan, "Boundary attack++: Query-efficient decision-based adversarial attack," *arXiv preprint arXiv:1904.02144*, 2019.

[51] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh, "Query-efficient hard-label black-box attack: An optimization-based approach," *arXiv preprint arXiv:1807.04457*, 2018.

[52] L. Pengcheng, J. Yi, and L. Zhang, "Query-efficient black-box attack by active learning," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 1200–1205.

[53] Y. Dong, H. Su, B. Wu, Z. Li, W. Liu, T. Zhang, and J. Zhu, "Efficient decision-based black-box adversarial attacks on face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7714–7722.

[54] F. Khalid, H. Ali, M. A. Hanif, S. Rehman, R. Ahmed, and M. Shafique, "RED-Attack: Resource Efficient Decision based Attack for Machine Learning," *arXiv preprint arXiv:1901.10258*, 2019.

[55] R. Wiyatno and A. Xu, "Maximal Jacobian-based Saliency Map Attack," *arXiv preprint arXiv:1808.07945*, 2018.

[56] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *International Conference on Learning Representations, ICLR*, 2017, pp. 1–17.

[57] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[58] F. Khalid, H. Ali, M. A. Hanif, S. Rehman, R. Ahmed, and M. Shafique, "RED-Attack: Resource Efficient Decision based Attack for Machine Learning," *arXiv preprint arXiv:1901.10258*, 2019.

[59] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in neural information processing systems*, 2017, pp. 3517–3529.

[60] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 45–48.

[61] S. Thys, W. V. Ranst, and T. Goedemé, "Fooling automated surveillance cameras: adversarial patches to attack person detection," *CoRR*, vol. abs/1904.08653, 2019. [Online]. Available: http://arxiv.org/abs/1904.08653

[62] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, "Potrojan: powerful neural-level trojan designs in deep learning models," *arXiv preprint arXiv:1802.03043*, 2018.

[63] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[64] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," *arXiv preprint arXiv:1806.05768*, 2018.

[65] R. Karri, J. Rajendran, and K. Rosenfeld, "Trojan taxonomy," in *Introduction to Hardware Security and Trust*. Springer, 2012, pp. 325–338.

[66] J. Clements and Y. Lao, "Hardware trojan design on neural networks," in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[67] Y. Zhao, X. Hu, S. Li, J. Ye, L. Deng, Y. Ji, J. Xu, D. Wu, and Y. Xie, "Memory trojan attack on neural network accelerators," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1415–1420.

[68] I. H. Abbassi, F. Khalid, S. Rehman, A. M. Kamboh, A. Jantsch, S. Garg, and M. Shafique, "TrojanZero: Switching Activity-Aware Design of Undetectable Hardware Trojans with Zero Power and Area Footprint," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 914–919.

[69] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[70] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Csi neural network: Using side-channels to recover your artificial neural network information," *arXiv preprint arXiv:1810.09076*, 2018.

[71] K. Yoshida, T. Kubota, M. Shiozaki, and T. Fujino, "Model-extraction attack against fpga-dnn accelerator utilizing correlation electromagnetic analysis," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 318–318.

[72] S. Pal, Y. Gupta, A. Shukla, A. Kanade, S. Shevade, and V. Ganapathy, "A framework for the extraction of deep neural networks by leveraging public data," *arXiv preprint arXiv:1905.09165*, 2019.

[73] L. Batina, S. Bhasin, D. Jap, and S. Picek, "{CSI} {NN}: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 515–532.

[74] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[75] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 601–618.

[76] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[77] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18.

[78] ——, "Towards practical verification of machine learning: The case of computer vision systems," *arXiv preprint arXiv:1712.01785*, 2017.

[79] H. Hosseini, S. Kannan, and R. Poovendran, "Dropping pixels for adversarial robustness," *CoRR*, vol. abs/1905.00180, 2019. [Online]. Available: http://arxiv.org/abs/1905.00180

[80] T.-W. Weng, H. Zhang, P.-Y. Chen, A. Lozano, C.-J. Hsieh, and L. Daniel, "On Extensions of CLEVER: A Neural Network Robustness Evaluation Algorithm," in *Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2018, pp. 1159–1163.

[81] I. Goodfellow, "Gradient masking causes clever to overestimate adversarial perturbation size," *arXiv preprint arXiv:1804.07870*, 2018.

[82] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[83] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[84] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[85] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Advances in neural information processing systems*, 2016, pp. 2613–2621.

[86] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," in *AAAI Conference on Artificial Intelligence*, 2018.

[87] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods Symposium*. Springer, 2018, pp. 121–138.

[88] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *International Conference on Learning Representations (ICLR)*, 2019.

[89] M. A. Siddiqui, J. W. Stokes, C. Seifert, E. Argyle, R. McCann, J. Neil, and J. Carroll, "Detecting Cyber Attacks Using Anomaly Detection with Explanations and Expert Feedback," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2872–2876.

[90] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using IC fingerprinting," in *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 2007, pp. 296–310.

[91] S. Adee, "The hunt for the kill switch," *iEEE SpEctrum*, vol. 45, no. 5, pp. 34–39, 2008.

[92] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *2008 IEEE International workshop on hardware-oriented security and trust*. IEEE, 2008, pp. 51–57.

[93] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, pp. 10–25, 2010.

[94] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[95] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," *arXiv preprint arXiv:1910.13063*, 2010.

[96] Q. Sun, L. Ma, S. Joon Oh, L. Van Gool, B. Schiele, and M. Fritz, "Natural and effective obfuscation by head inpainting," in *Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5050–5059.

[97] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1528–1540.

[98] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 36–52.

[99] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *Advances in Neural Information Processing Systems*, 2017, pp. 4672–4681.

[100] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[101] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 619–631.

[102] M. Isakov, L. Bu, H. Cheng, and M. A. Kinsy, "Preventing neural network model exfiltration in machine learning hardware accelerators," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2018, pp. 62–67.

[103] Y. Liu, D. Dachman-Soled, and A. Srivastava, "Mitigating reverse engineering attacks on deep neural networks."

[104] X. Wang, R. Hou, Y. Zhu, J. Zhang, and D. Meng, "Npufort: a secure architecture of dnn accelerator against model inversion attack," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*. ACM, 2019, pp. 190–196.

[105] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *arXiv preprint arXiv:1803.05961*, 2018.

[106] A. Tuszynski, "Essential Pattern and Sequence Sensitivity in Semiconductor Memories," Minnesota Univ., Minneapolis (USA). Dept. of Electrical Engineering, Tech. Rep., 1980.

[107] C. Constantinescu, "Trends and challenges in vlsi circuit reliability," *IEEE micro*, vol. 23, no. 4, pp. 14–19, 2003.

[108] R. Baumann, "Soft errors in advanced computer systems," *Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[109] W. D. Greason and G. P. Castle, "The Effects of Electrostatic Discharge on Microelectronic Devices A Review," *Transactions on industry applications*, no. 2, pp. 247–252, 1984.

[110] J. Zhang, K. Liu, F. Khalid, M. Hanif, S. Rehman, T. Theocharides, A. Artussi, M. Shafique, and S. Garg, "INVITED: Building Robust Machine Learning Systems: Current Progress, Research Challenges, and Opportunities," in *Design Automation Conference*, 06 2019, pp. 1–4.

[111] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 415–426.

[112] W. Wu and N. Seifert, "MBU-Calc: A compact model for multi-bit upset (MBU) SER estimation," in *International Reliability Physics Symposium*. IEEE, 2015, pp. SE–2.

[113] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives," in *Design Automation Conference*. ACM, 2014, pp. 1–6.

[114] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *International Symposium on Microarchitecture*. IEEE Computer Society, 2008, pp. 129–140.

[115] K. Kang, S. Gangwal, S. P. Park, and K. Roy, "NBTI induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution?" in *Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, 2008, pp. 726–731.

[116] M. Shafique, F. Khalid, and S. Rehman, "Intelligent security measures for smart cyber physical systems," in *Euromicro Conference on Digital System Design (DSD)*. IEEE, 2018, pp. 280–287.

[117] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14, 2014, pp. 361–372.

[118] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[119] S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, "The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1. ACM, 2014, pp. 519–532.

[120] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 60–71.

[121] M. K. Qureshi, D.-H. Kim, S. Khan, P. J. Nair, and O. Mutlu, "AVATAR: A variable-retention-time (VRT) aware refresh for DRAM systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 427–437.

[122] S. Khan, C. Wilkerson, Z. Wang, A. R. Alameldeen, D. Lee, and O. Mutlu, "Detecting and mitigating data-dependent dram failures by exploiting current memory content," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 27–40.

[123] J. Blandford, A. Waskiewicz, and J. Pickel, "Cosmic ray induced permanent damage in MNOS EAROMs," *Transactions on Nuclear Science*, vol. 31, no. 6, pp. 1568–1570, 1984.

[124] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *International Conference on Learning Representations, ICLR*, 2017, pp. 1–14.

[125] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[126] J. Lu, H. Sibai, E. Fabry, and D. Forsyth, "No need to worry about adversarial examples in object detection in autonomous vehicles," *arXiv preprint arXiv:1707.03501*, 2017.

[127] "Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam," https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html.

[128] P. Reviriego, J. A. Maestro, S. Baeg, S. Wen, and R. Wong, "Protection of memories suffering MCUs through the selection of the optimal interleaving distance," *Transactions on Nuclear Science*, vol. 57, no. 4, pp. 2124–2128, 2010.

[129] F. Vargas and M. Nicolaidis, "SEU-tolerant SRAM design based on current monitoring," in *International Symposium on Fault-Tolerant Computing*. IEEE, 1994, pp. 106–115.

[130] G.-C. Yang, "Reliability of semiconductor RAMs with soft-error scrubbing techniques," *IEE Proceedings-Computers and Digital Techniques*, vol. 142, no. 5, pp. 337–344, 1995.

[131] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM journal of research and development*, vol. 6, no. 2, pp. 200–209, 1962.

[132] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[133] M.-Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395–401, 1970.

[134] M. Patel, J. S. Kim, H. Hassan, and O. Mutlu, "Understanding and modeling on-die error correction in modern DRAM: An experimental study using real devices," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 13–25.

[135] "Meet Tesla's self-driving car computer and its two AI brains," https://www.cnet.com/news/meet-tesla-self-driving-car-computer-and-its-two-ai-brains/.

[136] M. A. Hanif, F. Khalid, R. V. W. Putra, S. Rehman, and M. Shafique, "Robust machine learning systems: Reliability and security for deep neural networks," in *International Symposium on On-Line Testing And Robust System Design (IOLTS)*. IEEE, 2018, pp. 257–260.

[137] H. Lee, M. Shafique, and M. A. Al Faruque, "Aging-aware workload management on embedded GPU under process variation," *Transactions on Computers*, vol. 67, no. 7, pp. 920–933, 2018.

[138] 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 *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 255–268.

[139] D. Lee, S. Khan, L. Subramanian, S. Ghose, R. Ausavarungnirun, G. Pekhimenko, V. Seshadri, and O. Mutlu, "Design-induced latency variation in modern DRAM chips: Characterization, analysis, and latency reduction mechanisms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, p. 26, 2017.

[140] S. Khan, D. Lee, and O. Mutlu, "PARBOR: An efficient system-level technique to detect data-dependent failures in DRAM," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 239–250.

[141] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *Design Automation Conference*. ACM, 2018, p. 19.

[142] P. N. Whatmough, S. K. Lee, D. Brooks, and G.-Y. Wei, "DNN engine: A 28-nm timing-error tolerant sparse deep neural network processor for IoT applications," *Journal of Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, 2018.

[143] E. Karl, D. Sylvester, and D. Blaauw, "Timing error correction techniques for voltage-scalable on-chip memories," in *International Symposium on Circuits and Systems*. IEEE, 2005, pp. 3563–3566.

[144] S. Campos, E. Clarke, W. Marrero, and M. Minea, "Timing analysis of industrial real-time systems," in *Proceedings of 1995 IEEE Workshop on Industrial-Strength Formal Specification Techniques*. IEEE, 1995, pp. 97–107.

[145] M. A. Pena, J. Cortadella, A. Kondratyev, and E. Pastor, "Formal verification of safety properties in timed circuits," in *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)(Cat. No. PR00586)*. IEEE, 2000, pp. 2–11.

[146] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, Not ECC, for Hard Failures in Resistive Memories," in *International Symposium on Computer Architecture*. ACM, 2010, pp. 141–152.

[147] J. Wang, X. Dong, and Y. Xie, "Point and Discard: A Hard-error-tolerant Architecture for Non-volatile Last Level Caches," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 253–258.

[148] J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in ***VTS***. IEEE, 2018, pp. 1–6.

[149] S. Koppula, L. Orosa, A. G. Yağlıkçı, R. Azizi, T. Shahroodi, K. Kanellopoulos, and O. Mutlu, "EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2019, pp. 166–181.

[150] M. Abdullah Hanif and M. Shafique, "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, pp. 1–23, 2020.

[151] 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," *arXiv preprint arXiv:1912.00941*, 2019.

[152] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Systematic testing of convolutional neural networks for autonomous driving," *arXiv preprint arXiv:1708.03309*, 2017.

[153] H. K. Ekbatani, O. Pujol, and S. Segui, "Synthetic data generation for deep learning in counting pedestrians." in *ICPRAM*, 2017, pp. 318–323.

[154] G. J. Stein and N. Roy, "Genesis-rt: Generating synthetic images for training secondary real-world tasks," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7151–7158.

[155] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *International conference on software engineering*. ACM, 2018, pp. 303–314.

[156] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *International Conference on Computer Vision*, 2017, pp. 618–626.

[157] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity Checks for Saliency Maps," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 9505–9515.

[158] D. Alvarez-Melis and T. S. Jaakkola, "Towards robust interpretability with self-explaining neural networks," in *International Conference on Neural Information Processing Systems*, 2018, pp. 7786–7795.

[159] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International Conference on Machine Learning*. JMLR. org, 2017, pp. 3319–3328.

[160] A. Levine, S. Singla, and S. Feizi, "Certifiably robust interpretation in deep learning," *arXiv preprint arXiv:1905.12105*, 2019.

[161] C.-H. Cheng, G. Nührenberg, C.-H. Huang, and H. Ruess, "Verification of Binarized Neural Networks via Inter-Neuron Factoring," in *Working Conference on Verified Software: Theories, Tools, and Experiments*. Springer, 2018, pp. 279–290.

[162] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.

[163] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[164] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *Journal of the ACM (JACM)*, vol. 50, no. 5, pp. 752–794, 2003.

[165] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 243–257.

[166] ——, "Challenging SMT solvers to verify neural networks," *Ai Communications*, vol. 25, no. 2, pp. 117–135, 2012.

[167] K. Scheibler, L. Winterer, R. Wimmer, and B. Becker, "Towards Verification of Artificial Neural Networks," in *MBMV*, 2015, pp. 30–40.

[168] "Gurobi Optimizer," https://www.gurobi.com/.

[169] "IBM ILOG CPLEX Optimization Studio," https://www.ibm.com/analytics/cplex-optimizer.

[170] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and Effective Robustness Certification," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 802–10 813.

[171] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An Abstract Domain for Certifying Neural Networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, p. 41, 2019.

[172] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6367–6377.

[173] I. E. Grossmann, "Review of nonlinear mixed-integer and disjunctive programming techniques," *Optimization and engineering*, vol. 3, no. 3, pp. 227–252, 2002.

[174] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 251–268.

[175] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward ReLU neural networks," *arXiv preprint arXiv:1706.07351*, 2017.

[176] P. Kouvaros and A. Lomuscio, "Formal verification of CNN-based perception systems," *arXiv preprint arXiv:1811.11373*, 2018.

[177] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[178] C. Murphy, P. Gray, and G. Stewart, "Verified perceptron convergence theorem," in *SIGPLAN International Workshop on Machine Learning and Programming Languages*. ACM, 2017, pp. 43–50.

[179] F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

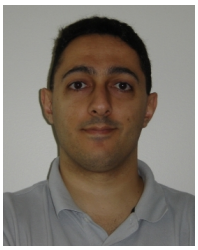
[180] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI2: Safety and Robustness Certification of Neural

Networks with Abstract Interpretation," in *Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.
[181] W. Xiang, H.-D. Tran, and T. T. Johnson, "Specification-guided safety verification for feedforward neural networks," *arXiv preprint arXiv:1812.06161*, 2018.
[182] W. Xiang and T. T. Johnson, "Reachability analysis and safety verification for neural network control systems," *arXiv preprint arXiv:1805.09944*, 2018.
[183] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *Transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5777–5783, 2018.
[184] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting Robustness Certification of Neural Networks," in *International Conference on Learning Representations*, 2018.

Muhammad Shafique (M'11 - SM'16) is a full professor of Computer Architecture and Robust Energy-Efficient Technologies (CARE-Tech.) at the Institute of Computer Engineering, TU Wien, Austria since Nov. 2016. He received his Ph.D. in Computer Science from Karlsruhe Institute of Technology (KIT), Germany, in Jan.2011. Before, he was with Streaming Networks Pvt. Ltd. where he was involved in research and development of video coding systems for several years. 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 AI, emerging technologies & nanosystems, FPGAs, MPSoCs, and embedded systems. His research has a special focus on cross-layer modeling, design, and optimization of computing and memory systems, as well as their deployment in use cases from Internet-of-Things (IoT), Cyber-Physical Systems (CPS), and ICT for Development (ICT4D) domains.

Dr. Shafique has given several Keynote, Invited Talks, and Tutorials. He has also organized many special sessions at premier venues and served as the Guest Editor for IEEE Design and Test Magazine and IEEE Transactions on Sustainable Computing. He has served on the TPC of numerous prestigious IEEE/ACM conferences. Dr. Shafique received the 2015 ACM/SIGDA Outstanding New Faculty Award. six gold medals in his educational career, and several best paper awards and nominations at prestigious conferences like CODES+ISSS, DATE, DAC and ICCAD, Best Master Thesis Award, DAC'14 Designer Track Best Poster Award, IEEE Transactions of Computer "Feature Paper of the Month" Awards, and Best Lecturer Award. Dr. Shafique holds one US patent and has (co-)authored 6 Books, 10+ Book Chapters, and over 200 papers in premier journals and conferences. He is a senior member of the IEEE and IEEE Signal Processing Society (SPS), and a member of the ACM, SIGARCH, SIGDA, SIGBED, and HIPEAC.

Mahum Naseer (S'19) received her B.E. in Electronics Engineering degree from NED University of Engineering and Technology, and M.S. in Electrical Engineering degree from National University of Sciences and Technology (NUST), Pakistan, in 2016 and 2018 respectively. She is currently pursuing her Ph.D. degree in Formal Methods for Resilient Embedded Systems from Technische Universität Wien (TU Wien), Austria. Her research interests include reliability analysis of systems, error control coding, resilient machine learning systems, and formal methods for system verification.



Theocharis (Theo) Theocharides (S'01-M'05-SM'11) is an Associate Professor in the Department of Electrical and Computer Engineering, at the University of Cyprus. Theocharis received his Ph.D. in Computer Engineering from Penn State University, working in the areas of low-power computer architectures and reliable system design with emphasis on computer vision and machine learning applications. Theocharis was honored with the Robert M. Owens Memorial Scholarship in May 2005. He has been with the Electrical and Computer Engineering department at the University of Cyprus since 2006, where he directs the Embedded and Application-Specific Systems-on-Chip Laboratory. He is also a Faculty Member of the KIOS Research and Innovation Center of Excellence 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 trade-offs for Systems on Chip and Embedded Systems. His focus lies on acceleration of computer vision and artificial intelligence algorithms in hardware, geared towards edge computing, and in utilizing reconfigurable hardware towards self-aware, evolvable edge computing systems. He serves on several organizing and technical program committees of various IEEE and ACM conferences, is a Senior Member of the IEEE, and a member of CEDA, is a member of the ACM, and an Associate Editor for IEEE Consumer Elect ronics magazine, the IET Computers and Digital Techniques, and the ETRI journal. He also serves on the Editorial Boards of IEEE Design & Test magazine, and ACM Transactions on Embedded Computing Systems.

Christos Kyrkou (S'09-M'14) received the B.Sc., M.Sc., and Ph.D. degrees in computer engineering from University of Cyprus, Nicosia, Cyprus, in 2008, 2010, and 2014, respectively. He is a Research Associate with the KIOS Research Center for Intelligent Systems and Networks, University of Cyprus. His research interests include real-time embedded systems, fieldprogrammable gate arrays and reconfigurable hardware, computer vision, machine learning, and smart camera networks.

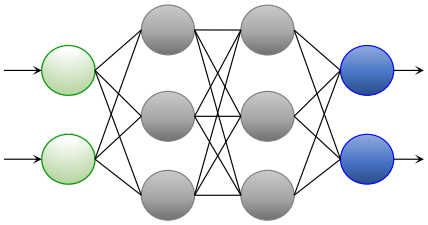
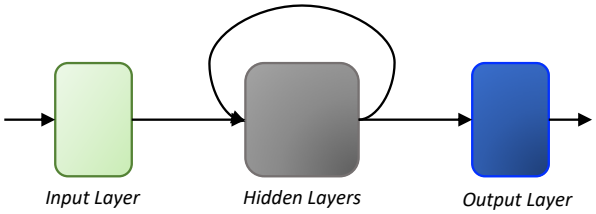
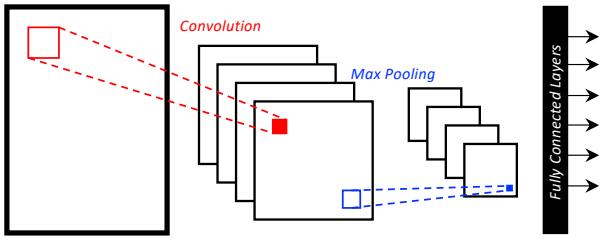
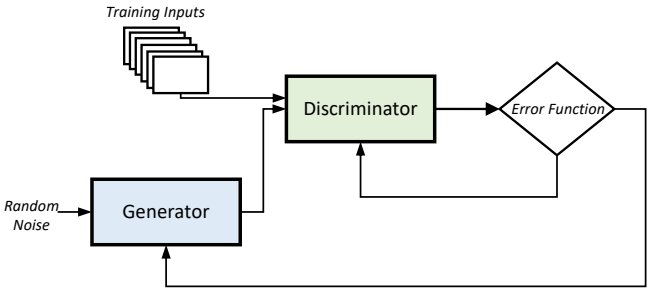
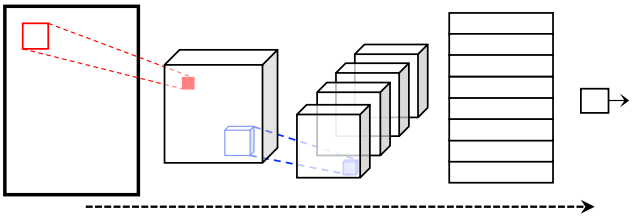
Dr. Kyrkou is a member of ACM and the Technical Chamber of Cyprus. He is a Technical Program Committee Member of the IEEE International Symposium on Nanoelectronic and Information Systems and the International Conference on Pervasive and Embedded Computing. He received an award for graduating top of his class during his B.Sc. studies at University of Cyprus and a full scholarship for his postgraduate studies from the Department of Electrical and Computer Engineering, University of Cyprus.

Prof. Onur Mutlu is a Professor of Computer Science at ETH Zürich. He is also a faculty member at Carnegie Mellon University, 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, have invented over the years have influenced industry and have been employed in commercial microprocessors and memory/storage systems. He obtained his PhD and MS in ECE from the University of Texas at Austin and BS degrees in Computer Engineering and Psychology from the University of Michigan, Ann Arbor. His industrial experience spans starting the Computer Architecture Group at Microsoft Research (2006-2009), and various product and research positions at Intel Corporation, Advanced Micro Devices, VMware, and Google. He was the recipient of the ACM SIGARCH Maurice Wilkes Award, the inaugural IEEE Computer Society Young Computer Architect Award, the inaugural Intel Early Career Faculty Award, faculty partnership awards from various companies, a healthy number of best paper or Top Pick paper recognitions at various computer systems and architecture venues. He is an ACM Fellow, an IEEE Fellow "for contributions to computer architecture research and practice", and an elected member of the Academy of Europe (Academia Europaea). His computer architecture course lectures and materials are freely available on YouTube, and his research group makes software artifacts freely available online. For more information, please see his webpage at http://people.inf.ethz.ch/omutlu/.

Lois Orosa is a PostDoc in the SAFARI research group at ETH Zürich. His current research interests are in computer architecture, hardware security, memory systems, and ML accelerators. He obtained his PhD from the University of Santiago de Compostela, and he was a PostDoc in the Institute of Computing at University of Campinas. He was a visiting scholar at University of Illinois at Urbana-Champaign and Universidade NOVA de Lisboa, and he acquired industrial experience at IBM, Recore Systems, and Xilinx.

Jungwook Choi is currently an assistant professor at Hanyang University, South Korea. He was formerly a research staff member at IBM T.J. Watson Research, Yorktown Heights, NY, where he has worked in the areas of approximate machine learning and deep learning algorithms for hardware acceleration. His research interests include high performance, energy efficient, and reliable implementation of machine learning and deep learning algorithms. He has a PhD in electrical and computer engineering from the University of Illinois at Urbana-Champaign.

APPENDIX

Neural Network (NN)	Description	Pictorial Representation of the Network
Feed-Forward Neural Network	These are the neural networks 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 *fully-connected* when every neuron in the preceding layer is connected to every neuron in the successive layer.	
Recurrent Neural Network (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.	
Convolutional Neural Network (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.	
Generative Adversarial 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.	
Capsule Network (CapsNet)	CapsNets are build 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 other hand, represents the instantiation probability. The connections between two consecutive capsule layers are learned dynamically during inference through the *routing-by-agreement* algorithm, which iteratively updates the *coupling coefficients* of the CapsNet. In this way, capsules learn to interpret high level features in a hierarchical manner.	
Spiking Neural Network	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.