DEPLOYING ADVERSARIALLY ROBUST LEARNING ENABLED COMPONENTS FOR CYBER PHYSICAL SYSTEMS

By

Xingyu Zhou

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

Aug 12th, 2022

Nashville, Tennessee

Approved:

Aniruddha Gokhale, Ph.D.

Xenofon Koutsoukos, Ph.D.

Gabor Karsai, Ph.D.

Abhishek Dubey, Ph.D.

Alan Peters, Ph.D.

To my parents, Rongguan Zhou and Yan wu, who provide me with everlasting support during my study.

ACKNOWLEDGMENTS

My deep gratitude goes first to my PhD advisor, Dr. Aniruddha Gokhale, who supported my study and research during the past years. The contents of this thesis would not be possible without his motivation, expertise, and guidance. I am extremely inspired by his passion for new ideas and open mind for all thoughts. Also, I would like to thank my co-advisor Dr. Xenofon Koutsoukos for guiding me to construct a formal way of research thinking, especially how to formulate and identify research problems throughout careers. Besides my advisor and co-advisor, I would like to thank Dr. Gabor Karsai, Dr. Abhishek Dubey, Dr. Alan Peters, for being members of my thesis committee, Dr Himanshu Neema, Peter Volgyesi and Dr. Daniel Balasubramanian for detailed research instruction and funding support.

I would like to give my special thanks to my friend Yi Li, who is not only the instructor in my research area but also my table tennis coach. During my research work, I got many reliable collaborators. In particular, I would like to thank Shunxing Bao for his guidance in academic thinking and writing. In addition, I would like to thank my frequent coauthors Robert Canady and Yogesh Barve for much help and brainstorming during our research. Also, Yuankai Huo, Zhengkai Zhang and Hongyang Sun for valuable experience sharing. My sincere thank also goes to my other colleagues at Vanderbilt University and Institute for Software Integrated Systems for their insightful comments and feedbacks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Cyber-physical systems (CPS) lie at the intersection of computing and the physical world [1]. They provide the foundation in realizing a range of applications in many domains like power grids, transportation, agriculture, supply chains and many others. Traditional system challenges of reliability and security have always been a concern for these systems [2]. However, since the goals of different CPS can vary, design approaches for these systems have utilized model-based design that employ domain-specific solutions or even case-specific solutions to handle a variety of system challenges to reason about various properties and ensuring systems that are correct-by-construction.

However, recent advances in machine learning, especially deep learning and their increasing use in the form of learning-enabled components (LECs) in CPS, bring both new opportunities and challenges. For instance, many questions such as which are the right LECs to use, how to design these LECs, what kind of composition of these LECs can form effective CPS applications, and how and where to deploy them become some of the critical questions that researchers and engineers must answer for the next generation of CPS. Since LECs are constructed of data-driven approaches, traditional model-based design of CPS alone is not sufficient for the next generation of CPS but rather a blending of model- and data-driven techniques is becoming the norm.

CPS usually sets up a design goal to be robust, which refers to the ability of the system to maintain operational performance under complex environmental disturbances or even malicious threats. The robustness issue in CPS could potentially affect the realms of both data (from sensors) and computers (to process sensor data). Sensor data bridges the gap between the physical world and computers. Different representations of data lead to different data types and volumes, which needs to be transferred using reliable communication paths. From the computation point of view, resilient CPS take data input from various sources to make informed decisions. The state-of-the-art in decision making involves learning-enabled components and this raises a greater demand on powerful computing capabilities. Meanwhile, wider applications of computation resources also pose more flexible requirements on physical availability aspects on computation devices.

On one hand, the resilience of a CPS could be motivated by the deployment of learning-enabled components (LECs). On the other hand, given higher complexity involved in current CPS designs, data-driven LECs are vulnerable to many threats. Among all kinds of potential threats, the adversarial attack against ML

models have become an inevitable critical aspect of the system resilience.

We emphasize this relationship in Figure 1.1, which formulates the key research ideology beneath this dissertation. The problem of adversarial robustness has become a significant aspect of LEC design towards more reliable CPS. A very typical example of traffic sign recognition is used here. In order to make an autonomous driving/transportation system, we use cameras to capture road visions. We process input data with LEC models and capture street signs inside the scene for more resilient and smarter decision making. However, this kind of LEC predictions could be extremely vulnerable under adversarial attacks and easily misclassified as a wrong sign [3; 4].

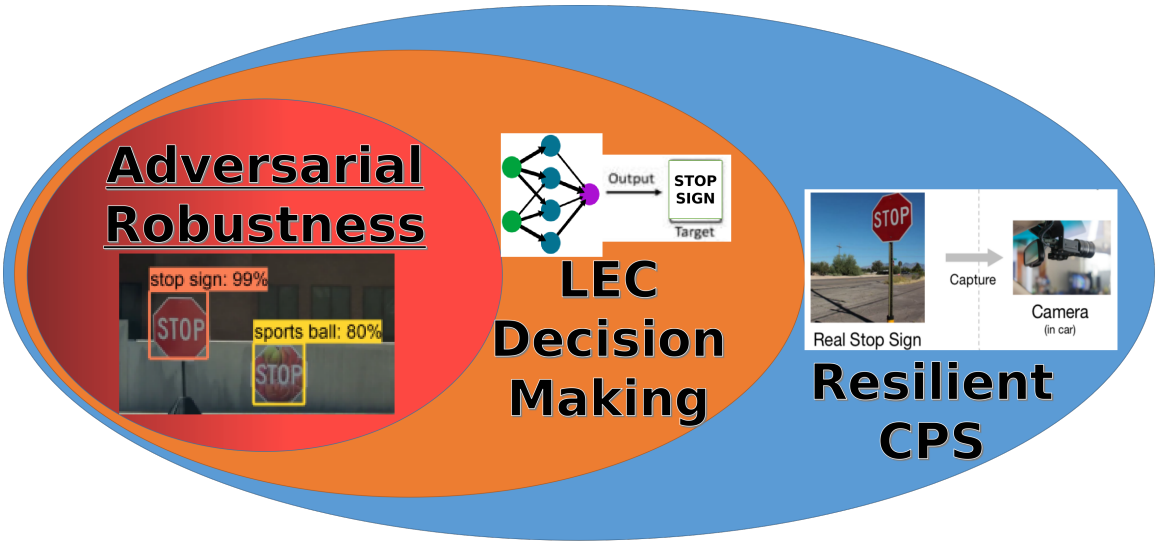

Figure 1.1: Regarding the relationship between CPS, resilience, LEC and adversarial robustness.

To make matters worse, when we compose LECs to build large CPS, the adversarial robustness of the overall system can be further degraded. The blended model- and data-driven design approach, however, incurs multiple challenges and can lead to designs that lack sufficient robustness and run-time resilience. For example, due to a lack of perfect sensors and simulators, system designers need to consider relatively generic settings for their domains. Accordingly, reusable and scalable hardware and software architectures become a significant demand for many applications. Although, this leads to simplicity, it can also be a cause of vulnerability. Furthermore, the critical nature of cyber-physical systems make these reusable parts even more exposed, which adds to the urgent demand for safety insurance during system design.

This work investigates several problems based on realistic scenarios in which application system designers aim to incorporate LECs under uncertain and adversarial environments. In a nutshell, the following cumulative principles are proposed to summarize the development of this work regarding the application of learning-based components in CPS:

- Machine learning is powerful.
- Machine learning is powerful but has its limitations.
- Machine learning is powerful and has its limitations but we can mitigate potential negative impacts through adaptive evaluations and resilient application deployments.

1.2 Key Research Challenges

In these above-described state of challenges in data-driven design of CPS and the need for resilient operations, we encounter a series of fundamental challenges and research questions that form the foundations of this dissertation.

1.2.1 Challenge 1: Determining threat on robustness of machine learning predictors

Recent advances in machine learning, especially deep neural networks, enable lower knowledge and hardware thresholds for building well-performed prediction models. For well-studied decision systems as in smart grid power systems, these techniques have shown to be successful in typical tasks like power quality classification [5] and load forecasting regression. However, current black-box statistical inference techniques used in the LECs make them vulnerable to adversarial attacks [6]. In particular, deep neural networks are susceptible to adversarial examples, which are synthetic perturbations added to original samples in a way that can misguide the original neural network prediction systems. Much past research has shown how these adversarial examples can pose threats to current machine learning systems in various ways [7].

Since we put most attention on the adversarial threats in CPS, we outline three cumulative spatial-temporal aspects that together form the potential robustness issue.

- Accurate predictions in cyber-physical systems are critical for infrastructure management, targeted pricing or other active response tasks. Therefore, we urgently need to use robust machine learning models especially deep neural network models for these tasks;
- Prediction models need data inputs from sensors. However, the hierarchical nature of sensor network topology can lead to compromise of the partial network and further the possibility of adversarial data injection;
- To make matters worse, state-of-art machine learning, especially deep learning, suffers from inherent vulnerabilities. So that they can be easily exploited by adversarial attacks, leading to undetected perturbations in inputs but obvious performance deviations in outputs.

In summary, *we need robust machine learning-based prediction in real-world cyber-physical systems.*

1.2.2 Challenge 2: Evaluating the resilience of machine learning predictors in CPS

The use of learning-enabled components that must execute on the computers for cyber-physical systems impose demands for efficient and reliable evaluation under realistic settings. This essentially becomes the problem of how to bridge the gap between domain-specific application data and machine learning techniques. To provide high-quality and user-friendly evaluation frameworks, model-driven design ideology with domain-specific abstractions can be induced. However, this leads to the challenge of designing suitable evaluation workflows that can balance the general requirement for domain-specific modeling and flexible resilience testing settings.

From the point of view of evaluating the adversarial robustness, we face the following challenges:

- First, there is no standardized approach dictating how much information to use in evaluation. There are different attack settings [8] like white-box, where the attacker has full knowledge of the predictor model including its architecture and weights and can directly use this information to maximize a target loss function using gradient-based optimization methods, and black-box, where the attacker can only obtain prediction results without accessing detailed architectures or the weights of the predictors. To make matters worse, there is no universal limitation on the amount of data access. Exposing too much training data would let the attacker extract the predictor [9].

- Secondly, not much attention has been paid to problems other than classification type and that too image classification type only. However, past research highlights the presence of adversarial examples in regression type problems also [10], which represents a major fraction of DL-based applications.

- Thirdly, although many different attack methods are known, there are no known standardized configurations to codify and study adversarial attacks. To conduct such a study, a user would need to manually configure the settings such as the allowed attack strength or the allowed computation steps.

- Finally, although some efforts to investigate adversarial attacks exist in the form of an integrated toolbox that allows a user to conduct attacks and defenses [11], many existing defense methods are domain- or case-specific and hence no end-to-end, general and automated exploration procedure for evaluating the robustness of deep learning models exists.

In a nutshell, although many attack and defense methods have been proposed to date, the diversity of current research in this field makes it challenging for users in practice to rapidly test the robustness of a trained model under adversarial settings. Thus, we surmise that designing user-friendly resilience evaluation workflows for deep learning models is needed for practical applications.

1.2.3 Challenge 3: Mitigating the impacts of adversarial attacks on LECs

Although machine learning methods, such as deep learning, have been widely used in many scenarios, correspondingly, their vulnerability to adversarial attacks has also grown. Unfortunately, there have not been any systematic solutions to adversarial examples thus far. Consider an application scenario where a trained prediction model can be deployed and made to output prediction results, then it should be most convenient to deploy it in an out-of-box manner. In this way, the system can hold all data confidential and only expose the model but in doing so, the deployed model cannot be updated easily. Adversarial machine learning shows that the exposure of models would be vital enough for an attacker to design adversarial examples. In this way, resilient design and evaluation of a robust prediction system becomes a challenging task.

Based on typical implementations and settings of prediction problems and prior research in cyber-physical systems, we believe we should construct our defense framework and try to obtain the following practical but challenging goals:

- First of all, this should be a framework that enables domain specific abstractions for the specific problem setting like load prediction. Therefore, it can formalize the security and resilience testing under adversarial settings in a relative user-friendly way;
- Secondly, this framework should seek a balanced performance between adversarial robustness and natural robustness. This means, it should greatly reduce adversarial impacts and meanwhile when it works on normal clean data, its performance loss should be as small as possible;
- Finally, as there has been so much past research works on this topic, these efforts should not be thrown away. So there is a need to make this framework compatible with existing techniques and frameworks so that the practical deployment cost could be minimized.

1.2.4 Challenge 4: Determining suitability of hardware resources for system resilience

Due to high computation burden of current deep learning models, the deployment of these models in cyber-physical applications demands more optimal computation allocations among hardware devices. In the past ten years, the development of Cloud Computing enables remote access to strong computation power. However, it is often the case that a cyber-physical application only allows limited remote network access and even runs in a partially or totally offline manner. This has give rise to Edge Computing, which shifts computations that are typically carried out in the Cloud to the network edge to address the real-time latency needs of applications and to overcome the inordinate costs in transferring massive amounts of data being generated at the edge to the cloud.

It is not hard to imagine edge devices are often constrained in their compute capacity and lower energy consumption is critical. Hardware accelerator devices, such as field programmable gate arrays (FPGAs), graphical processing units (GPUs) and application specific integrated circuits (ASICs) among others, have emerged as an alternative to traditional CPUs since they not only help perform computations faster but also consume much less energy than a traditional CPU thereby helping to lower both capex (i.e., lower procurement costs) and opex (i.e., lesser energy usage). Since different accelerator technologies can illustrate different traits for different application types that run at the edge, there is a critical need for effective mechanisms that help developers select the right technology (or a mix of) to use in their context. However, due to the difficulty in programming and running applications on these devices, there is a general lack of effective benchmarking capabilities for accelerator technologies at the edge and how their performances could impact the system resilience.

Another realistic challenge regarding the resilience aspect of the models deployed on hardware devices is whether the machine learning predictor deployed on the edge device would be even more vulnerable to attacks than the cloud-based. This challenge arises from the higher likelihood of hardware device exposure to the outside world. When a prediction model has already been deployed on a physical hardware device, it can become difficult to update it in many cases. This actually provides potential attackers more time and flexibility to compromise models and data on devices, which can pose serious threats to the system resilience.

1.3 Contributions of the Thesis

This section summaries our research contributions, including theoretical model construction and solution framework design. Our research provides novel scopes from two main aspects. Chapter 3-5 investigates adversarial threats under generalized sensor network setting. Later on, Chapter 6-9 extends our view to potential hardware platforms for model deployment and further investigates resilience and robustness issues on them. The detailed contributions of this thesis are listed below.

1.3.1 Chapter 3: Evaluating Resilience of Grid Load Predictions

Chapter 3 resolves Challenges 1 and 2, where we propose and evaluate the resilience of grid load predictions under stealthy adversarial attacks. The main contributions of this work are:

- Develop a model-based and cloud-supported platform for rapidly designing and evaluating resilient learning/testing settings for sensor network architectures.
- Present a general step-by-step framework that can formalize the security and resilience testing in distributed sensor networks under adversarial settings.

- Design a generic procedure that can be utilized to implement stealthy adversarial attacks on machine learning predictors with the presence of self-checking detectors in sensor networks.
- Conduct a case study for distributed power network load forecasting and demonstrate potential risks even with prediction procedures that incorporate anomaly detection algorithms.

This work has resulted in the following publication. Zhou, X., Li, Y., Barreto, C. A., Li, J., Volgyesi, P., Neema, H., & Koutsoukos, X. (2019, November). Evaluating Resilience of Grid Load Predictions under Stealthy Adversarial Attacks. In 2019 Resilience Week (RWS) (Vol. 1, pp. 206-212). IEEE.

1.3.2 Chapter 4: Adversary Mitigation Using DDDAS

This Chapter 4 resolves Challenges 3. We construct a defensive data repair framework based on the dynamic data-driven system paradigm to mitigate adversarial impacts. We make the following contributions in this work:

- We extend the framework built in Chapter 3 that can formalize the security and resilience testing in distributed sensor networks under adversarial settings;
- We design an iterative dynamic data repair scheme of Dropout-Detect-Reconstruct-Tradeoff to boost the robustness of data using the DDDAS paradigm for ongoing predictions; and
- We conduct a case study for distributed power network load forecasting to demonstrate potential risks for machine learning predictors and the efficiency of our defensive data repair framework.

This work has resulted in the following publication. Zhou, X., Canady, R., Li, Y., & Gokhale, A. (2020). "Overcoming Stealthy Adversarial Attacks on Power Grid Load Predictions Through Dynamic Data Repair." In 3rd International Conference on Dynamic Data Driven Application Systems (InfoSymbiotics/DDDAS2020).

1.3.3 Chapter 5: Adversarial Data-driven Prognostics

This Chapter 5 resolves Challenges 1,2 and 3. We study the impact of adversarial examples in data-driven prognostics and potential ways to mitigate these impacts. We make the following contributions in this work:

- We present a workflow that can formalize the security and resilience testing in data-driven prognostics settings.
- We show the vulnerability of deep learning prognostics models under various settings.

- We investigate the possibility of inducing randomization elements and domain knowledge of semantic structural context to mitigate adversarial impacts.
- We conduct a robustness case study using the engine degradation dataset on typical applications of health status classification and remaining useful life prediction.

This work has resulted in the following publication: Zhou, X., Canady, R., Li, Y., & Gokhale, A. (2020). "Overcoming Adversarial Perturbations in Data-driven Prognostics Through Semantic Structural Context-driven Deep Learning." In Annual Conference of the Prognostics and Health Management Society 2020 (PHM20).

1.3.4 Chapter 6: Edge Hardware Accelerator Recommendation

This Chapter 6 resolves Challenges 2 and 4. We present *HARE (Hardware Accelerator Recommender for the Edge)*, which is a framework to help users rapidly and cost-effectively select the right hardware accelerator for a given compute-intensive task. Our work focuses on a general framework to model compute latency and energy usage across accelerator devices, and makes the following contributions:

- We present a novel hardware-software co-evaluation framework that formalizes the performance comparison involving training samples obtained on real hardware;
- We develop a simple approximation method that can be utilized to implement long-term cost analysis for hardware-based machine learning inference behavior at the edge;
- We show case studies involving two realistic machine learning application settings and demonstrate HARE in these contexts.

This work has resulted in the following publication: Zhou, X., Canady, R., Bao, S., & Gokhale, A. (2020). "Cost-effective Hardware Accelerator Recommendation for Edge Computing." In 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20).

1.3.5 Chapter 7: Cloud Assisted TinyML for PHM

This Chapter 7 further investigates Challenges 2 and 4. We extend our view from Chapter 6 to the novel edge LEC deployment hardware option of TinyML on microcontrollers(MCUs). We study the potential of TinyML in data-driven prognostics and emphasize the necessity of compensating its limited hardware resources with more mature cloud computing techniques. Our work proposes the following application patterns of TinyML-based CPS applications especially PHM applications where incorporating cloud computing could make a huge difference:

- Cloud-assisted model training and code generation for heterogeneous edge hardware devices including MCUs.
- Data storage and integration for heterogeneous sensor data sources.
- Model inference computation offloading for latency, power and communication optimization.

This work has resulted in the following publication: Zhou, X., Kang, Z., Canady, R., Bao, S., Balasubramanian, D. A., & Gokhale, A. (2021). "Exploring Cloud Assisted Tiny Machine Learning Application Patterns for PHM Scenarios." In Annual Conference of the Prognostics and Health Management Society 2021 (PHM21).

1.3.6 Chapter 8: Fast Black-box Adversarial Attack on/with BNN

Based on practical edge hardware deep model deployment procedures discussed in Chapter 6, we propose to combine edge computing and adversarial machine learning together targeting Challenge 1 and 4. A simple but efficient black-box adversarial attack test has been conducted on low precision neural networks (especailly binary neural networks) running on FPGAs. This shows the potential risk for edge machine learning models and serves as the motivation for further research (a poster accepted at IBM IEEE CAS/EDS20 3rd AI Compute Symposium).

The main contributions of this paper are the following:

- We show how it is possible to realize high-speed adversarial attacks using programming logic. We further show the potential risks of machine learning systems on the edge/IoT, especially for the popular quantized neural networks.
- We present a general procedure that can be utilized to implement black-box adversarial attacks using FPGA devices. This procedure is general enough to be easily adapted to other existing learning systems and can be regarded as a generalized safety testing procedure.
- We show a novel application setting of utlizing an existing hardware IP from an absolute high-level without going through the whole hardware design procedure. This can be regarded as a hardware-oriented algorithm design process that can be useful for designers that are more focused on the algorithm application side.

1.3.7 Chapter 9: Universal Adversarial Perturbations on Quantized Models

Motivated by the edge model deployment procedure shown in Chapter 6 and potential potential vulnerabilities of edge LECs in Chapter 8, this chapter further tends to solve challenge 1, 3 and 4 from a joint view.

We study the potential threat of using a same universal adversarial perturbation to compromise all prediction models targeting the same task but with different quantization compressions across the cloud/edge pattern. To solve this issue, we also propose a defensive model deployment workflow. This chapter makes the following contributions (full paper has been accepted at IC2E 2022 10th IEEE International Conference on Cloud Engineering conference):

- We present a reusable and reproducible evaluation workflow to formalize the security and resilience testing of quantized neural network models under universal adversarial perturbation (UAP) attacks.
- We show the consistency of robustness of adversarially trained models and propose a defensive quantization method towards more robust quantized models that can be codified into a configurable and reusable middleware solution that edge services can utilize.
- We evaluate the UAP attack and defense on the most widely used CIFAR10/CIFAR100/SVHN datasets. We show the vulnerability of normal quantized models and highlight the compactness and the efficiency of our proposed defense strategies.

1.4 Organization

The detailed technical contributions of this work are in Chapters 3-9. This thesis is organized as follows.

- Chapter 2 reviews the related work in machine learning for smart grids, health management and prognostics, model-driven design, adversarial machine learning and prediction model deployments on edge;
- Chapter 3 discusses the problem of stealthy adversarial attack evaluation in power load predictions;
- Chapter 4 discusses the problem of adversarial impact mitigation using the DDDAS paradigm;
- Chapter 5 discusses the problem of adversarial data-driven prognostics;
- Chapter 6 discusses the problem of hardware accelerator selection for edge computing;
- Chapter 7 discusses the vision of machine learning model deployments on MCUs for prognostics and health management;
- Chapter 8 shows the vulnerability of low-precision neural network model on edge FPGA accelerators under simple black-box adversarial attacks;
- Chapter 9 discusses the problem of universal adversarial attack impact evaluation for edge based machine learning prediction systems and proposes a defensive model deployment workflow to solve this issue;

- Chapter 10 concludes this thesis and discusses future directions in this research area.

CHAPTER 2

Related Works

2.1 Deep Learning for Smart Grids

For modern large-scale electricity markets, smart grids can make use of machine learning techniques for resilient decision making to increase income meanwhile to decrease cost. Many papers/dissertations have been published since 2016, covering deep learning applications in smart grids. These application fields cover various scenarios include load forecasting, defect/fault detection, cyber security, stability analysis and almost all views of smart grids [12]. In the interest of research intersection, this section puts attention on two sub-fields of load forecasting and power disturbance classification.

Active predictions on system data can help system holders seek a dynamic balance between supply and demand to maximize income.In particular, the accuracy of the forecasts improves aggregating the demand of large groups of customers [13]. Research shown in paper [14] is a relative early work published in 2017 that propose a long short-term memory (LSTM) deep recurrent neural network-based framework to tackle this prediction issue. They tested their prediction framework on a publicly available set of real residential smart meter data. And at that time their work shows better performance compared to various benchmarks. Research work [15] also makes trials on prediction models with more flexible network architectures. Hybrid prediction models involving a mixture type of features are also explored in work [16; 17], in which they also embed additional information of weather and time markings of holidays as well as seasons.

On the other hand, the power quality(PQ) issue has a direct impact on the overall continuation of electric transmission and distribution network, the real-time monitoring of electric power signals has become highly necessary for modern smart grids [18]. Therefore, it is essential for an electricity service provider to own the capability to recognize and classify power disturbances accurately. Even before deep learning era, neural networks have shown to be successful in this typical intelligent task [5]. Recent development of deep learning greatly reduce the threshold of building a well-performed predictor. Researchers have made trials on different architectures including convolution neural network (CNN) [19; 20], recurrent neural network(RNN) [21], gated recurrent units(GRU) [22] and other combinations.

2.2 Health Management and Prognostics

Nowadays, as we face more and more large-scale information systems, it is more and more important for us to have a good understanding of what condition this system is currently in of even more specifically how long the remaining useful life the system has [23]. The former belongs to the classification setting where

the model predicts the remaining useful life of the system in a certain range and outputs a categorical result to indicate to which health status category the system belongs to. In the more general regression setting, the model outputs a numerical prediction value for the remaining useful life. The main difficulty for the system-level prognostics problem is that the overall performance(such as remaining useful life) is not the simple addition of each component's performance. There are two main types of methods involved for these predictive prognostics tasks: model-based and data-driven.

Classical model-based methods assume that the model must be accurate enough to depict the system behaviors. One example of such kind of method is Bond Graph[24] which is used for graphical description of dynamic behaviour of physical systems. It needs to know formally how different components of a system interact and the system output can be computed analytically. It is not difficult to imagine that for most real-world cases, we only have a partial knowledge of the system and this makes it impossible to describe directly.

In contrast, data-driven prognostics methods are suitable for scenarios where complete analytical models of the physics are difficult or impossible to formulate. In particular, recent development of deep learning techniques have made deep strides in health management and prognostics [25; 26]. For a relatively complex system with a number of sensors, the service provider can utilize data flows from multiple smart data sources to perform data-driven prognostics using a deep learning model with low barrier [27].

One common issue for current data-driven methods is that learning frameworks need to learn completely from scratch. In this way, there is no space for external expert knowledge to be further induced. All expert knowledge must be represented and embedded in the training set. In this context, recent research has shown the potential of the hybrid approach of combining physics-based semantic information [28] with pure data-driven prognostics.

2.3 Model-driven CPS Evaluation

To bridge the gap between domain specific application data and machine learning, researcher put efforts into integrated testbed platforms for cyber-physical scenarios [29]. One of the system-level pioneering works in this area is SURE [30; 31], which is the abbreviation of SecUre and REsilient Cyber–Physical Systems. SURE platform incorporates realistic models of cyber and physical components and their interactions. These can be used to compose cyber attacks on CPS behavior and operation and security risks of different attack settings can be evaluated.

Our smart grid evaluation framework in Chapter 3 and 6 builds on DeepForge [32]. DeepForge is a model-based and cloud-based collaborative development environment with deeply integrated domain specific modeling features created using WebGME [33]. It combines model integrated computing with rapid prototypical development of machine learning models. DeepForge presents machine learning models with

four hierarchical concepts: Pipelines, Operations, Executions and Jobs. *Operations* are atomic functions which accept inputs and return outputs. A *Pipeline* refers to a stack of tasks, such as data pre-processing, training or testing. Executing pipelines results in the creation of *Executions*. A *Job* corresponds to the selected operation along with its run status and metadata associated with its execution. DeepForge provides built-in extensions to the state-of-art Keras/TensorFlow machine learning framework. It enables real-time collaboration between modelers and analysts and uses strict version control for reproducible experiments.

2.4 Adversarial Machine Learning

Recent advances of machine learning especially deep neural networks enable lower thresholds for building prediction models. However, current black-box nature of statistical inference systems make them vulnerable to adversarial attacks [7] . In particular, deep neural networks are susceptible to adversarial examples [34], which are synthetic perturbations added to original samples in a way that can misguide the original neural network prediction systems. The field of adversarial machine learning has been active long before current deep learning era for more than a decade [35], most work has put attention on classification problems. Moreover, security and robustness in state-of-art deep machine learning models has aroused significant concern [36] since the discovery of adversarial examples in current deep neural networks. However, adversarial regression, which is widely seen in cyber-physical system (CPS) settings, is still a relatively new problem [6; 37]. Consequently, for cyber-physical system applications even though prediction models can be built without barrier using state-of-art deep neural networks, a more cautious view still needs to be taken due to potential risks with this kind of learning-based components [10].

Complementary to the work showing adversarial threats, much efforts have been put on robustness improvement for current deep learning models. One popular defense technical path is improving model robustness during the training phase [38; 39; 40]. Research has shown that adversarial training with data augmentation [41] helps in improving robustness of neural networks to adversarial attacks. Adversarial training improve model robustness on a single model and does not induce additional computation costs for model deployment. For the inference phase, inducing randomization elements [42] has shown much potential. There has been research [43] showing that random cropping, padding with resizing of the adversarial images reduces their effectiveness. A recent work [44] also suggests adding small random noises to one input for several predictions and ensemble prediction results with the highest confidence class. However, these methods have only been specially designed for computer vision and classification tasks and cannot be applied to other scenarios easily. For CPS settings like smart grids, we need a more flexible end-to-end robustness improvement solution with assured quality of service.

2.5 Machine Learning on Edge

Edge Computing shifts computations that are typically carried out in the Cloud to the network edge to address the real-time latency needs of applications and to overcome the inordinate costs in transferring massive amounts of data being generated at the edge to the cloud. This question is even more pronounced [45] for resource-constrained edge computing/IoT, where traditional cloud computing is not always efficient for data processing due both to higher costs involved in transferring large volumes of data produced at the edge of the network to a distant cloud and also to the adverse impact on real-time response times expected by users of services deployed at the edge [46]. This problem is particularly acute in scenarios where cameras with video streams are involved [47] because compute intensive tasks, especially deep learning based machine learning applications on these data, often require millions of operations for each inference [48].

Thus, to maintain low latencies and compliance with the power sensitivity of edge deployment, smaller models [49] operating on more efficient hardware are preferred [50]. To that end, hardware acceleration technologies, such as field programmable gate arrays (FPGAs), graphical processing units (GPUs) and application-specific integrated circuits (ASICs) among others, have shown significant promise for edge computing [51]. In a general sense, selecting a suitable hardware accelerator technology for a given computational task can be regarded as finding out a hardware device placement and optimization strategy for a given topology. Prior work has explored this general problem for service placement for edge computing [52; 53]. From a hardware perspective, workload-specific accelerator placement frameworks for automobiles and unmanned aircraft have been proposed [54; 55; 56]. In these workloads, energy usage is dominated by vehicles rather than computational overhead. From a higher cloud-level point of view, research on device and service placement also exists [57; 58].

Even with the proliferation of edge computing, it is worth pointing out the higher likelihood of hardware access in realistic cyber-physical systems also brings new challenges [59]. There has been research work [60; 61; 62] showing the possibility of using hardware-specific side-channel information to steal model or data on potential edge applications.

CHAPTER 3

Grid Load Predictions under Stealthy Adversarial Attacks

Recent advances in machine learning enable wider applications of prediction models in cyber-physical systems. Smart grids are increasingly using distributed sensor settings for distributed sensor fusion and information processing. Load forecasting systems use these sensors to predict future loads to incorporate into dynamic pricing of power and grid maintenance. However, these inference predictors are highly complex and thus vulnerable to adversarial attacks. Moreover, the adversarial attacks are synthetic norm-bounded modifications to a limited number of sensors that can greatly affect the accuracy of the overall predictor. It can be much cheaper and effective to incorporate elements of security and resilience at the earliest stages of design. In this paper, we demonstrate how to analyze the security and resilience of learning-based prediction models in power distribution networks by utilizing a domain-specific deep-learning and testing framework. This framework is developed using DeepForge and enables rapid design and analysis of attack scenarios against distributed smart meters in a power distribution network. It runs the attack simulations in the cloud backend. In addition to the predictor model, we have integrated an anomaly detector to detect adversarial attacks targeting the predictor. We formulate the stealthy adversarial attacks as an optimization problem to maximize prediction loss while minimizing the required perturbations. Under the worst-case setting, where the attacker has full knowledge of both the predictor and the detector, an iterative attack method has been developed to solve for the adversarial perturbation. We demonstrate the framework capabilities using a GridLAB-D based power distribution network model and show how stealthy adversarial attacks can affect smart grid prediction systems even with a partial control of network.

3.1 Problem Overview

For electricity markets, the profit and safety is based on a dynamic balance between supply and demand. As a result, an accurate demand prediction system is essential for the pricing strategy to maintain infrastructures as well as to maximize the profit. For smart grids, service providers may try to reduce the uncertainties by forecasting the demand of their customers. In particular, the accuracy of the forecasts improves aggregating the demand of large groups of customers [13].

With recent developments of machine learning techniques especially deep learning, neural network prediction models can be constructed to gain good results. Time-series network models from distributed meter readings of different areas in the smart grid is useful in predicting the future load demands.

However, the complexity of current inference systems leads to vulnerabilities that can be exploited by

adversarial attacks. In particular, deep neural networks are susceptible to adversarial examples, which poses a great risk in current machine learning systems. Adversarial examples are synthetic modifications added to original samples in a way that can misguide the neural network prediction systems. These are inputs to machine learning models that an attacker has intentionally intercepted and disguised to cause the prediction model to make mistakes [7].

Even though the field of adversarial machine learning has been active for more than a decade [35], most work has been conducted on classification problems. Adversarial regression, which is widely seen in cyber-physical system (CPS) settings, is still a relatively new topic. Moreover, since the discovery of adversarial examples in current deep neural networks in 2013 [34], security and robustness in state-of-art deep machine learning models has become a hot topic [36] and aroused significant concern. Consequently, even though a power load prediction model can be built easily using state-of-art deep learning techniques, a more cautious view still needs to be taken due to security issues that affect cyber-physical systems with integrated learning-based components [10].

In a nutshell, accurate load forecasting in smart grids is critical for managing infrastructure through targeted pricing and for maximizing profit. However, the hierarchical topology of power networks could lead to partial compromise of the network. Testing and realizing these kinds of security risks becomes difficult due to reliance on domain-specific knowledge for customizing the state-of-art machine learning techniques. To address these issues in the power system CPS domain we followed a model-based design approach [63] [64] [31] and developed our forecasting method and security testing framework using DeepForge [32].

In the following sections we attempt to put attention on the following objectives:

- Develop a model-based and cloud supported platform for rapidly designing and evaluating resilient learning/testing settings for sensor network architectures.
- Present a general step-by-step framework that can formalize the security and resilience testing in distributed sensor networks under adversarial settings.
- Design a generic procedure that can be utilized to implement stealthy adversarial attacks on machine learning predictors with the presence of self-checking detectors in sensor networks.
- Conduct a case study for distributed power network load forecasting and demonstrate potential risks even with prediction procedures that incorporate anomaly detection algorithms.

The rest of the paper is organized as follows. Section II provides the motivation for evaluating security risks in the power system. Section III illustrates the theoretical background of our adversarial attack evaluation platform. Section IV presents a case study to demonstrate the capabilities of our platform by utilizing

a power distribution network and simulating it with GridLAB-D. Finally, Section V concludes the paper and draws remarks for future directions.

3.2 Motivation

A cyber-physical system (CPS) is an intersection of computers and the physical world [1]. The two domains are connected with a multitude of sensors and/or actuators with dynamic system characteristics. In general, sensor networks are tightly integrated and utilized in a CPS for dynamic control and decision making. On the other hand, potential security risks in sensor networks can often be easily generalized and transferred to specific application scenarios. In particular, for critical large-scale infrastructures like smart grids, a large network of sensors are required to work together to support high-level decisions.

Machine learning techniques are required for smart CPSs to make decisions automatically and more adaptive to the environment. For a smart grid power network, load forecasting models are learned and applied to make predictions on future loads for efficient and profitable power system operations [65]. Research on load forecasting have been conducted over thirty years [66] and the critical role of load forecasting in smart grids have been demonstrated from various aspects [67].

In this paper, we attempt to explore the vulnerability of machine learning models in distributed sensor network settings. We consider how smart meter readings from different sources in a power distribution system are used for forecasting load using a pre-trained machine learning model. For a more realistic setting, we also use an anomaly detector (trained from the same dataset as the prediction model) to see whether input data deviates significantly from the nominal values.

In order to negatively impact the power system, the attacker can manipulate input data [68] to mislead the predictor. We set *reasonable* flexible constraint settings on the attacker due to the physical extent of the network and the cost of compromising individual sensors. In particular, we allow the attacker to modify a limited number of meter readings with an upper bound on the modification ratio of each meter value. These kinds of constraints enable the attack against the predictor to be stealthy enough to remain hidden from the anomaly detector that is used along with the predictor. In the following sections, we present an architecture for inference model learning and testing for sensor network settings, and demonstrate sensor network level adversarial attacks using a power distribution network load forecasting case.

3.3 Methodology

In this section we provide more details on the underlying methodologies of our framework. We use state-of-art tools of TensorFlow/Keras to build generic executable pipelines to show the prediction and attack tests under flexible settings. This approach enables easier collaboration and potentially more effective dissemina-

tion of testing results.

3.3.1 Model-Based Framework

To bridge the gap between domain specific application data and machine learning, our framework builds on DeepForge [32]. DeepForge is a model-based and cloud-based collaborative development environment with deeply integrated domain specific modeling features created using WebGME [33]. It combines model integrated computing with rapid prototypical development of machine learning models.

DeepForge presents machine learning models with four hierarchical concepts: Pipelines, Operations, Executions and Jobs. *Operations* are atomic functions which accept inputs and return outputs. A *Pipeline* refers to a stack of tasks, such as data pre-processing, training or testing. Executing pipelines results in the creation of *Executions*. A *Job* corresponds to the selected operation along with its run status and metadata associated with its execution. DeepForge provides built-in extensions to the state-of-art Keras/TensorFlow machine learning framework. It enables real-time collaboration between modelers and analysts and uses strict version control for reproducible experiments.

To unify the data representation,we use a generalized input and processing data format for networked sensor data. Each data input is a two-dimensional matrix consisting of data from n sensors involving T time steps. As this is a simple notation, there may be optional pre-processing steps required to transform raw input data into this format. For the grid load forecast case, the data of power meter sensor readings are numerical values and our testbed allows normalization on input data to ease sequential processing steps.

To support the power domain we developed several reusable *atomic* operations such as data gathering using GridLAB-D simulator, pre-processing data, model training, and testing. We developed hybrid adversarial attack methods ranging from single-step FGSM [36] to more complex adversarial attack settings. Figure 3.1 shows the key atomic operations (operation screenshot truncated) provided by our framework and a sample adversarial attack pipeline constructed using these components. For testing purposes, we also embed a pre-trained prediction model as well as a detection model for the load forecasting case study.

Ten step-by-step pipelines ranging from simple train/test workflows to more complex attack/detection evaluations are incorporated. All of these pipelines can be utilized and modified easily. This effectively provides a highly user-friendly abstraction for domain-specific deep-learning applications.

We formalize the application and security testing procedure pipelines for sensor networks commonly used in CPS and divide them into the following five major parts and provide targeted executable pipelines using Tensorflow and Keras to generalize them for different scenarios:

- **Basic predictor training/testing**: This is the overall goal and a crucial step in the machine learning model for the sensor network data implementation. A predictor training and evaluation pipeline is

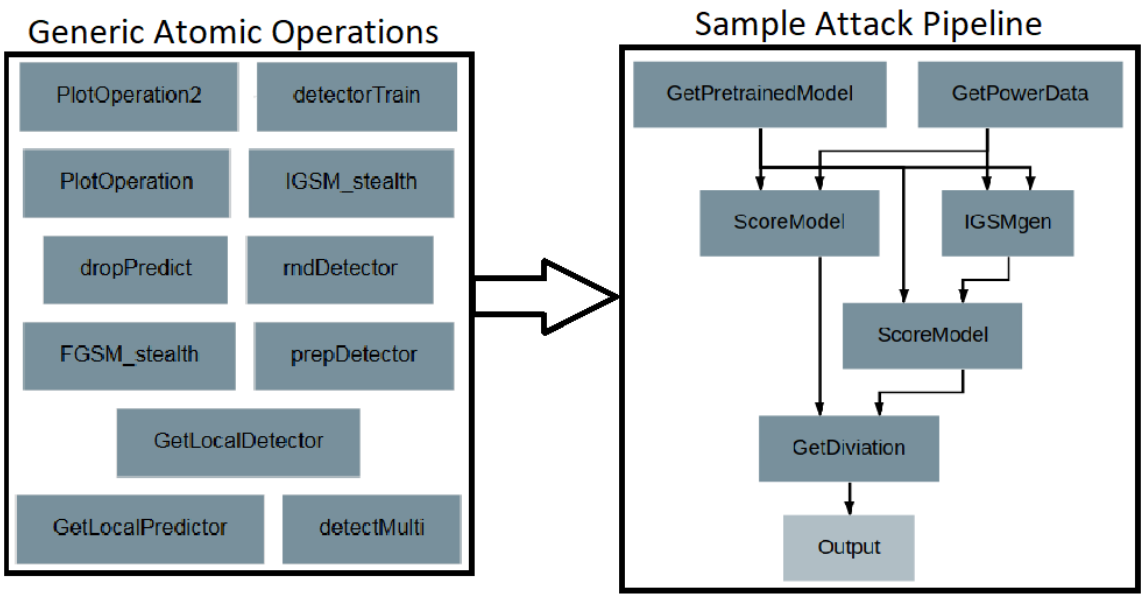

Figure 3.1: Creating Attack Pipelines using Generic Atomic Operations

provided for generic use.

- **Adversarial attack on the prediction model**: This involves a worst white-box attack setting that allows users to test the robustness of the predictor. A few different adversarial settings are pre-built as basic pipelines targeting maximizing prediction deviations from original predictions or simply maximizing or minimizing prediction results.

- **Anomaly detector for the system**: In practical uses, the user also holds an auto-encoder model to detect anomalous sensors or to denoise the noisy sensor data. For the sensor network, we propose an LSTM auto-encoder concerning the general requirement for multiple sensors involving multiple time steps.

- **Stealthy adversarial attacks**: Considering the existence of an anomaly detector, the attacker can currently conduct stealthy adversarial attacks to deviate the predictor as well as evading the anomaly detector. Stealthy attacks need to consider both maximizing prediction loss as well as evading the detector.

- **Evaluation of attacks and defenses**: Allow the user to test different system settings to explore the robustness of the detection and prediction system. A general randomization-based exploration tool is provided to experiment on potential strategies.

3.3.2 Predictor Model

Predicting the total demand of the distribution system is essentially a multi-variate time-series regression task. We solve the prediction problem using state-of-art methods based on *Recurrent Neural Networks (RNNs)*. In particular, we use a Long-Short-Term Memory (LSTM) network [69], which accepts data sequences as inputs and returns a sequence of predictions. There is no universal standard for a good prediction model in realistic tasks. However, for regression problems, typically the statistical metric of mean squared error (MSE) is used. We use MSE for evaluating our models.

3.3.3 Anomaly Detector

In our literature survey, we did not find a detailed explanation for the origin of adversarial examples. However, we could gather approximate reasoning of adversarial examples from 'unexplored spaces' or 'over-explored spaces' from higher dimensions [70]. This enabled us to determine key characteristics of adversarial examples from a statistical point of view.

In this way, we can view adversarial examples as some kind of anomaly that needs to be detected. Previous research conducted in detecting adversarial examples [71] [72] mostly focused on image classification tasks, such as *MNIST* digit recognition or *Cifar-10* object classification. A recent trend of detection seeks to make this detector construction process more automatic via generative models [73] [74]. The basic intuition behind these detection techniques is to judge whether the input sample is likely to be from the normal distribution of the sample data [75].

Using machine learning models for anomaly detection in CPS has become more popular recently due to improvements in both deep-learning techniques and computation power. Researchers have experimented with auto-encoders [76] [77] [78] and generative adversarial networks [79]. It is worth pointing out that in CPS, the input data formats vary significantly, which limits creation of a universal detection framework. Therefore, we can only seek domain-specific or even case-specific solutions [78]. Here, for numerical data coming from distributed sensors, we use an auto-encoder to build an anomaly detector.

Auto-encoder models learn internal representations with the objective $f(x) = x$ mapping to the input itself. In order to detect anomaly using an auto-encoder, a common procedure is to set statistical thresholds for the residual between the original input and the reconstructed input. Most detection cases only need to give binary outputs for the whole input sample. For the sensor network in our case study, we expect the detector to find whether specific sensors in the network are likely to be compromised [80].

We set individual detection thresholds for each sensor meter in the network using a simple procedure. After training the auto-encoder using the training data, we use the training data to compute the fitting error (MSE) for all sensors and using maximum MSE of each sensor as the error threshold for anomaly detec-

tion. During the prediction phase, the auto-encoder takes inputs and compares output residuals with the pre-computed thresholds and generates a list of sensors with potential for adversarial attacks.

3.3.4 Stealthy Adversarial Attack

There are two important premises for a successful adversarial attack. First, the attack should not be detected by the machine learning system it is attacking. Secondly, the attack should cause worse performance of the machine learning prediction system. Based on these two general requirements, an adversarial attack can be formulated as an optimization problem which attempts to find the best synthetic perturbations that maximize the prediction loss while keeping the modification magnitude at a small enough level so as to go undetected.

3.3.4.1 L0-FGSM Attack

Among the various attack methods developed so far, one of the most well-known and popular is the FGSM (Fast Gradient Sign Method) [36] which formulates the optimization using only a single equation:

$$\eta = \varepsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{3.1}$$

Here θ represents the parameters of the model, x represents inputs to the model, y refers to the targets associated with x (for tasks with targets) and $J(\theta, x, y)$ is the cost function used to train the neural network. The magnitude constraint added to the original sample is represented by ε. This method is quite simple and intuitive. The attacker makes modifications to maximize the loss function, meanwhile as the modification is small enough it actually preserves the original information structure. It is worth pointing out this attack method regard the requirement of being stealthy as self-evident under the magnitude constraint of ε.

We adapt our algorithm to CPS, which are highly complex with a data input space that is potentially bigger than a fixed range. *Algorithm 1* shows a single attack step to deviate this predictor. Note that each meter (value in our input vector) has its unique regression case because the input range may not be in a fixed area. Therefore, the deviation is denoted as a ratio rather than a fixed value. And the number of meters allowed to be modified is also limited.

Algorithm 1 l_0-FGSM Attack

Require: $\boldsymbol{x}_0$: original observation; f: predictor; $J(f,x)$: cost function of predictor f according to input data x; D: l_∞ max deviation ratio; N: set of meters can be modified; $allowModN$: data selector to get values from a set of meters.

1: $\Delta\boldsymbol{x} \leftarrow \mathbf{0}$, $i \leftarrow 0$
2: $grad \leftarrow \nabla J(f, \boldsymbol{x}_0)$
3: $grad \leftarrow allowModN(grad, N)$
4: $\Delta\boldsymbol{x} \leftarrow D * \boldsymbol{x} * sign(grad)$
5: **return** $\boldsymbol{x} + \Delta\boldsymbol{x}$

3.3.4.2 Stealthy L0-IGSM Attack

Algorithm 1 generates simple single-step attacks, which suffer from two limitations. First, the non-linearity of the predictor itself makes it almost impossible for a single-step to reach the optimal loss increase. On the other hand, an out-of-distribution detector can detect those sensors which are modified large enough and thus nullify the adversarial effects. For certain cases with detectors, the attacker can reformulate the optimization problem into a joint form [81] to solve for the joint adversarial perturbation. However, the detector we are using provides thresholds for individual sensors. which makes it difficult for such piecewise functions to be formed together.

To address these two limitations, we apply an iterative approach to reach for a more optimal adversarial perturbation. In practice, for each $\boldsymbol{x}_0$, our approach performs $NumIter$ number of iterations with maximum deviation ratio D, takes steps of step length ratio $(1+D)^{(1/NumIter)}$, and computes $\Delta\boldsymbol{x}$ using the gradient sign method starting from the result of previous iteration. With the existence of a detector, intermediate results are first checked with the detector to remove exposed parts and then sent into the next iteration for further exploration. *Algorithm 2* shows how we generate the stealthy attack.

Algorithm 2 Iterative Stealthy Attack

Require: $\boldsymbol{x}_0$: original observation; f: predictor; d: detector; $J(f,x)$: cost function of predictor f according to input data x; D: max. deviation ratio; N: number of sensors allowed modifying; $allowModN$: data selector to get values from a set of meters; $NumIter$: number of iterations.

$\Delta\boldsymbol{x} \leftarrow \mathbf{0}$, $i \leftarrow 0$
$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
$\alpha \leftarrow (1+D)^{(1/NumIter)}$
while $i < NumIter$ **do**
  $grad0 \leftarrow \nabla J(f,\boldsymbol{x})$
  $grad \leftarrow allowModN(grad0, N)$
  $\Delta\boldsymbol{x} \leftarrow \alpha * \boldsymbol{x} * sign(grad)$
  $\Delta N \leftarrow d(\boldsymbol{x}+\Delta\boldsymbol{x})$
  $grad \leftarrow allowModN(grad0, N-\Delta N)$
  $\Delta\boldsymbol{x} \leftarrow \alpha * \boldsymbol{x} * sign(grad)$
  $\boldsymbol{x} \leftarrow \boldsymbol{x}+\Delta\boldsymbol{x}$
end while
return $\boldsymbol{x}$

Essentially, Algorithm 2 is a hybrid attack combining L_0 and L_∞ constraints and the detection threshold constraints. In practice, an attacker is allowed to modify Num meter readings with a maximum deviation ratio D. For each input $\boldsymbol{x}_0$, we choose the most sensitive, but undetected, meters using their gradient values for each iteration. Figure 3.2 shows a high-level execution pipeline as an example for a stealthy attack we have implemented. In Figure 3.2, input artifacts like data and pre-trained models are shown in light blue and operations in dark blue. An attacker fetches the pre-trained prediction models and data to attack and uses the 'IGSM_stealth' operation to generate adversarial perturbations. A generic prediction evaluation operation

of 'ScoreModel' is used on both original and adversarial samples. The adversarial impact can be computed later based on the deviation from these two metric results shown as the overall output. The integrity of model-based tools provides a simple framework combining different types of practical constraints and a low computation complexity for fast prototypical tests.

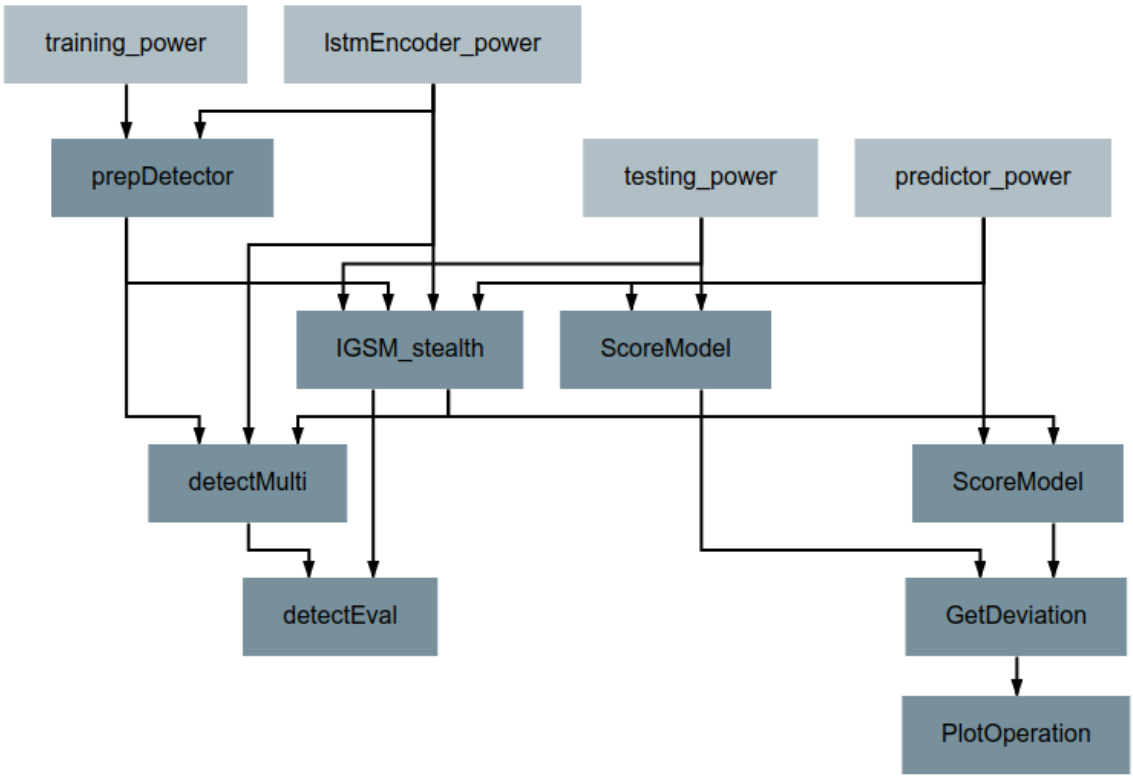

Figure 3.2: Stealthy Adversarial Attack Pipeline

3.4 Experiment Results

For demonstration purposes, we utilize a case study based on a medium-scale power distribution network over time. It is worth noting that the prediction and attack evaluation strategies can also be generalized to other distributed sensor network settings with no barrier.

3.4.1 Power System Setting

For this case study, we make a detailed simulation of an electric distribution system using GridLAB-D and the prototypical distribution feeder model provided by the Pacific Northwest National Laboratory (PNNL) [82]. The distribution model captures the fundamental characteristics of distribution utilities in the US. Figure 3.3 shows the topological structure of the power distribution network. The figure shows the load data collection mechanism for the distributed smart meter setting. Meters are connected to the user households and their

usage data reports are transmitted to the control center in a hierarchical manner. In this case, we use the prototypical feeder *R1-12.47-3*, which represents a moderately populated area. Furthermore, we added representative residential loads to the distribution model using the script in [83]. In summary, our distribution model has 109 commercial and residential loads, which in-turn include appliances such as heating, ventilation, and air conditioning (HVAC) systems, water heaters, and pool pumps. GridLAB-D allows us to model the response of the loads to weather and market's prices, giving realism to the simulations.

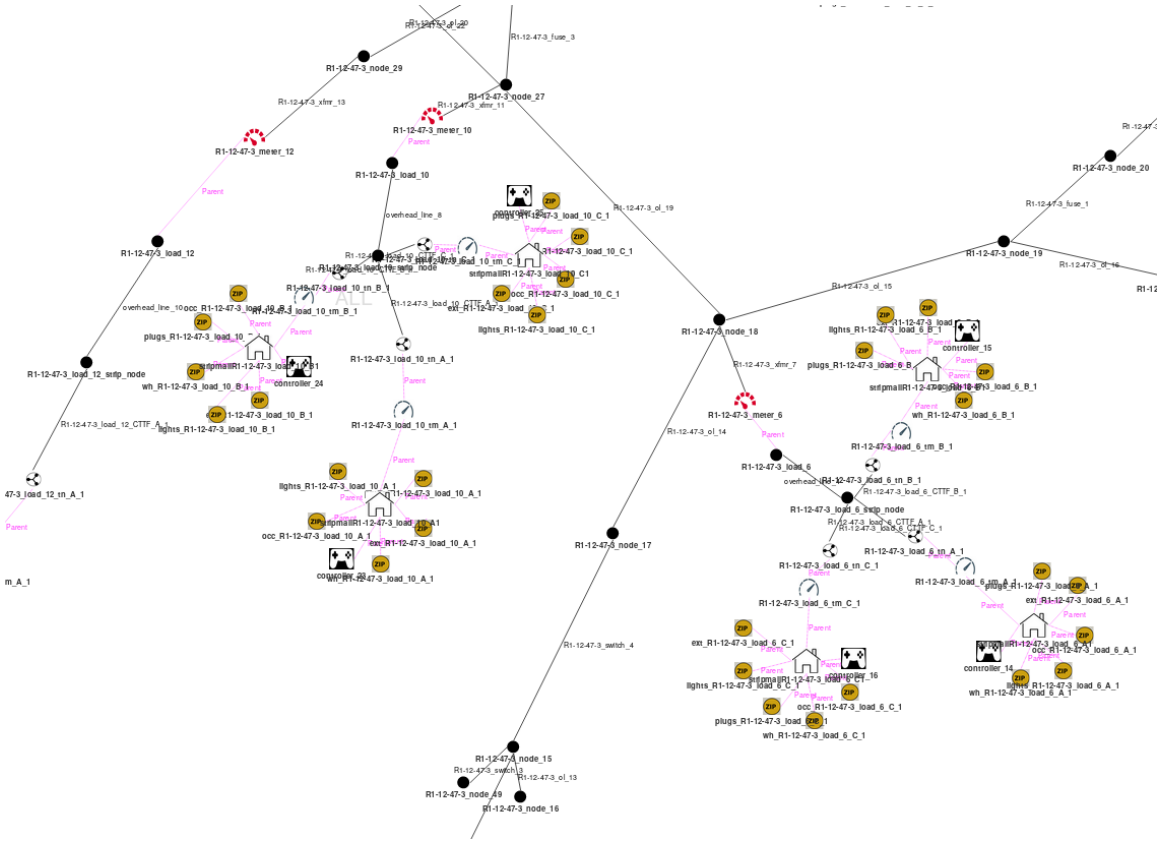

Figure 3.3: High-level Organization of the Power Distribution Network

For each hourly time step, the predictor takes loads from meter readings in the past 24 hours and also takes into account the temperature data during the same period of time. After absorbing and formatting these inputs, the predictor gives out the future total network load for the next hour. This prediction is conducted iteratively to provide references for dynamic pricing or facility maintenance.

3.4.2 Results

We use DeepForge to build a load forecasting model for this power distribution network using techniques of LSTM and deep neural networks. We use the historical data from 109 loads (both commercial and residential) and the temperature during summer time (June to August) as inputs. The simulations also incorporate weather

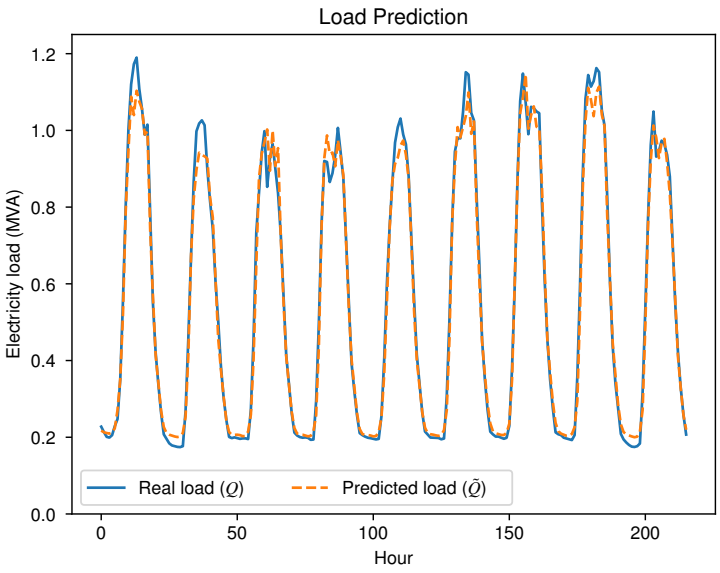

Figure 3.4: Load Prediction Illustration

data from Nashville, TN, to capture the impact of temperature in the electricity consumption. In this case, we use the measurements of the past 24 hours to predict the total load during the next hour. We make predictions every hour. Owing to the large-scale dataset generated for our problem, we chose a relatively large-scale neural network that includes three LSTM layers (with 150 units) and two fully connected layers (with 500 units). Figure 3.4 shows an example of the load prediction. The predictor is trained from 90% of the total data and leads to a mean squared error (MSE) of 0.1255 (*units:* Mega Volt-Amp) on the test dataset.

Along with the predictor, the power management system also holds an anomaly detector to detect whether sensors are working normally. We use a hard-decision detector which only outputs the binary classification results for sensors. The output for this detector is a vector with binary values indicating whether each meter reading in the network is likely to be an anomaly. Figure 3.6 shows the confusion matrix for the anomaly detector when 30% of meters in this network are added with a 20% level of Gaussian noise. The detector in our experiments shows an overall detection accuracy of 98.1 under random Gaussian noise. However, as we show later, this detector is still vulnerable to carefully designed adversarial attacks.

In our attack scenario, the sensor data is manipulated under constraints. At every time step, the attacker can manipulate a fixed number of meters in the network (30% in our experiments). Moreover, for each meter, the attacker is allowed to deviate the meter reading by a constraint level of 20%. This is a modification setting that is equivalent to the noise level in the original detection test.

Figure 3.5 shows the modification ratio of one sample for the IterativeGSM attack. The horizontal axis denotes the meter index and vertical index indicates multiple data for one meter sample. We can see that in a

total of 109 meters, 32 ($rate = 30\%$) meters can be modified by a ratio no more than 20%.

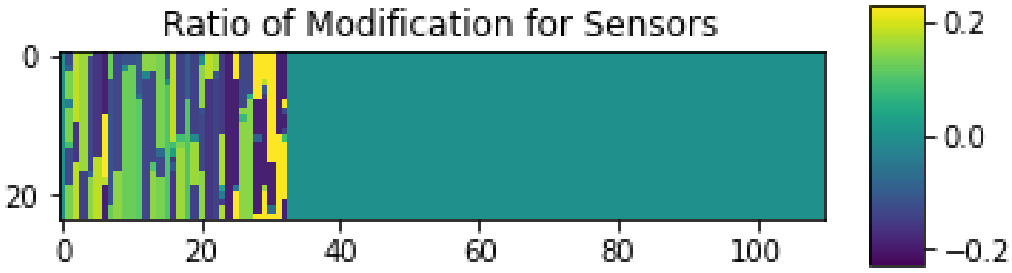

Figure 3.5: Modification Ratio for IGSM

Under same level of constraints, we explore four different adversarial attack settings (with and without detector):

- FGSM. Single step attack to maximize the deviation from the prediction
- IGSM. Iterative attack to maximize the deviation from the prediction
- DirectedGSM ($reverse = 1$). Iterative attack to minimize the predicted values
- DirectedGSM ($reverse = -1$). Iterative attack to maximize the predicted values

Figure 3.7 shows the confusion matrix for the anomaly detector when 30% of meters in this network are added with a level of 20% adversarial perturbation. The most critical metric here is the false positive (FP) part

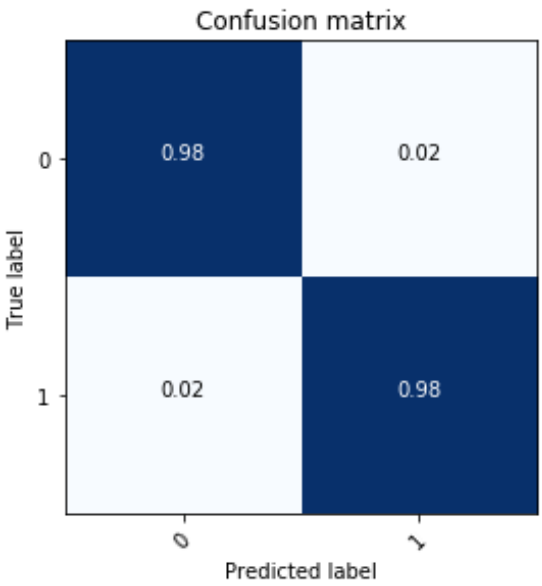

Figure 3.6: Detector Results using 30% Meters with 20% Level Gaussian Noise

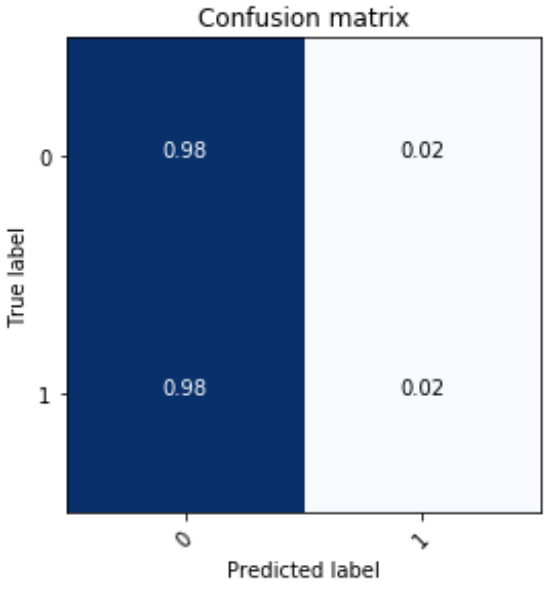

Figure 3.7: Detector Results using 30% Meters with 20% Level adversarial perturbation

of the detection under adversarial perturbation. In contrast to the original detection results (see Figure 3.6), the ratio of undetected adversarial meters shows an increase from 2% to 98%. This clearly demonstrates that the generic stealthy attack procedure described above can evade static anomaly detectors with a very high success rate.

To better evaluate the impact of adversarial attacks under detection settings, we updated the prediction step using a straightforward anomaly removal strategy. After the anomaly detection step, the sensor detected as abnormal from the input data space would be set as zero values. Thus, for the prediction, the abnormal input data is deactivated and the resulting input data is only relied on the remaining (most likely normal) sensors. In this way, the attacker only puts attention on the undetected part and tries to maximize the adversarial impact while keep the perturbation stealthy.

Table 3.1 shows experiment results under different attack and defense settings. The detection and anomaly removal strategy itself only brings in negligible deviations from the original predictions on normal data. Unfortunately, adversarial attacks with full knowledge of the detectors can find out the worst case for the predictor while still evading the detector. Our experiments show that the static detection and prediction mechanism remains vulnerable under adversarial settings even when only a small portion of sensors in the network are intentionally modified. A previous research [10] also explores the vulnerability of load forecast system under adversarial attack on weather data, where the significance of the single variable for weather is shown for the forecast design.

Table 3.1: Prediction Results (MSE) with Different Prediction Deployment Settings

Attack/ Detection Settings	Original/ NoAttack	Adversarial/ NoDetect	Original/ StaticDetect	Adversarial/ StaticDetect
Fast-GSM (rate=0.3,step_len=0.2)	0.1255	0.5375	0.1287	0.5322
Iterative-GSM (rate=0.3, step_ len=0.01,step_num=20)	0.1255	0.7801	0.1287	0.7606
DirectedGSM (rate=0.3, step_len=0.01 ,step_num=20, reverse=1)	0.1255	0.4785	0.1287	0.4913
DirectedGSM (rate=0.3, step_len=0.01 ,step_num=20, reverse=-1)	0.1255	1.025	0.1287	0.9899

3.5 Conclusion

In this paper, we demonstrated how to evaluate security and resilience of load forecasting predictors for a power distribution network. To enable rapid prototyping and evaluation, we introduced domain-specific abstractions in the model-based platform of DeepForge to construct a testing framework. To illustrate the capabilities of this framework, we used GridLAB-D for power network simulation, and provided various configurable and flexible security test settings in different forms. Our experiments showed that CPS that use

machine learning techniques for load predictions can suffer from worst-case stealthy bound-limited adversarial attacks even under a partial network compromise.

Limitations of this work mainly lie in the following aspect. Our current investigation of the resilience of the power distribution network uses a static dataset generated from GridLAB-D. Incorporating GridLAB-D as a DeepForge extension would make it more flexible. This will enable GridLAB-D to work in a simulation-in-the-loop manner with simultaneous learning. This means users will be able to define simulations in a more flexible way that incorporates more kinds of user-defined uncertainties such as weather anomalies and dynamic network changes.

CHAPTER 4

Overcoming Stealthy Adversarial Attacks with DDDAS

For power distribution networks with connected smart meters, current advances in machine learning enable the service provider to utilize data flows from smart meters for load forecasting using deep neural networks. However, recent research shows that current machine learning algorithms for power systems can be vulnerable to adversarial attacks, which are small designed perturbations crafted on normal inputs that can greatly affect the overall performance of the predictor. Even with only a partial compromise of the network, an attacker could intercept and adversarially modify data from some smart meters in a limited range to make the load predictor deviate from normal prediction results. In this part, we leverage the dynamic data-driven applications systems (DDDAS) paradigm and propose a novel data repair framework to defend against these kinds of adversarial attacks. This framework complements the predictor with a self-representative auto-encoder and works in an iterative manner. The auto-encoder is used to detect and reconstruct the likely adversarial part in the input data. Different reconstruction results come up given different sensitivity levels in detection. As new data flows in each iterative time step, the service provider continuously checks the error of the previous prediction step and dynamically trades off between different detection sensitivity levels to seek an overall stable data reconstruction. Case studies on power network load forecast regression demonstrate the vulnerability of current machine learning algorithms and correspondingly the effectiveness of our defense framework.

4.1 Problem Overview

Advances in machine learning have enabled prediction models to be applied to a range of cyber-physical systems yielding state-of-art performances, particularly smart power systems in need of tasks like power quality classification [5] and load forecast regression. In modern smart grids, accurate load forecasting is critical for managing the infrastructure through targeted pricing and predictive maintenance. Advances in machine learning enable the service provider to utilize data flows from smart meters to perform load forecasting [13] using a deep learning model. However, recent research [10] reveals that current machine learning algorithms proposed for power system application scenarios can be vulnerable to adversarial attacks [7], which are inputs with small designed perturbations added to normal ones that can adversely affect the overall performance of the predictor [34; 36]. In partially compromised hierarchical power networks, an attacker could intercept and maliciously modify data from some smart meters with small perturbations that can still make the load predictor deviate from normal prediction results.

To address these issues, we adopt the dynamic data-driven applications systems (DDDAS) paradigm [84]

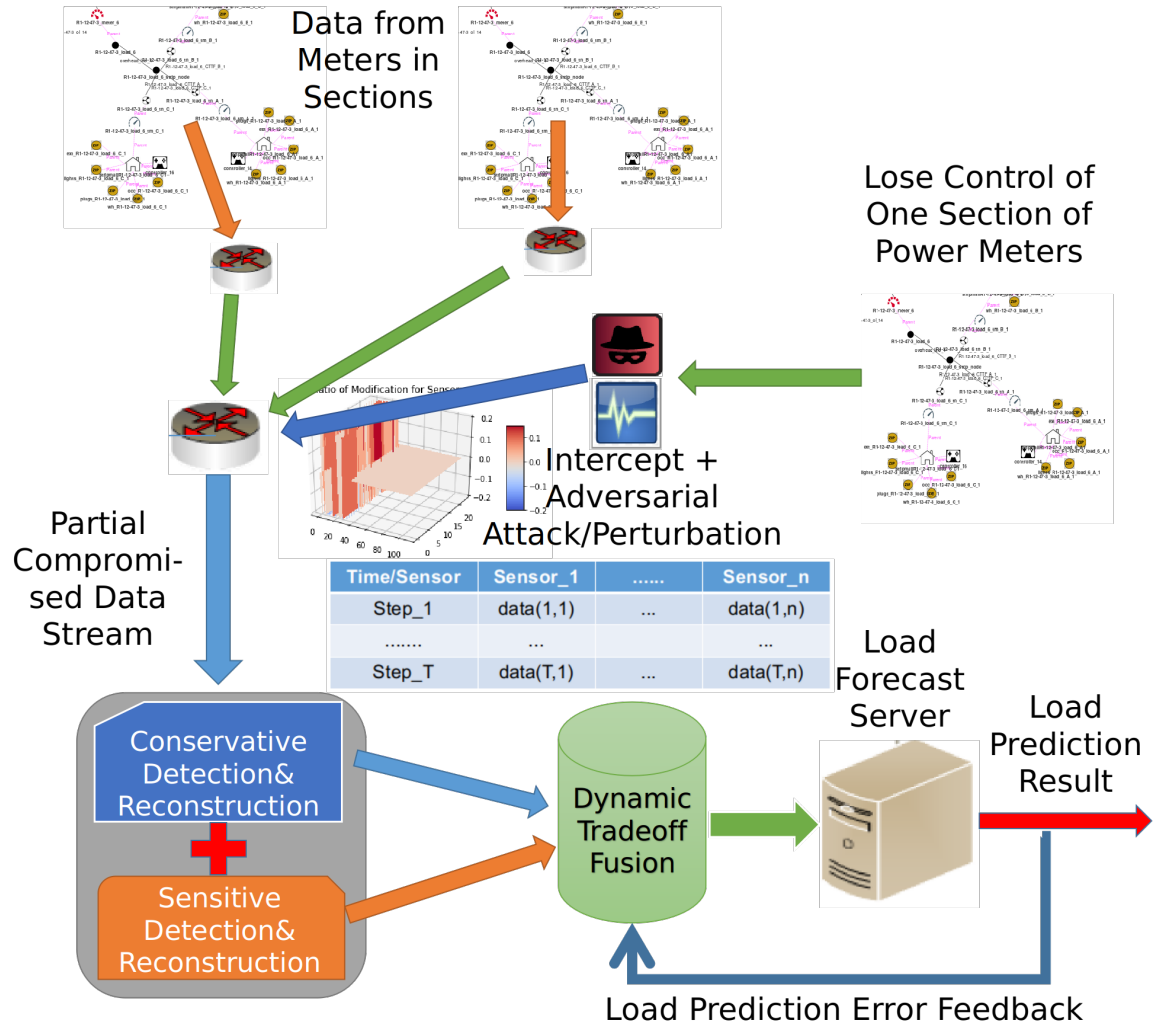

Figure 4.1: Overall Workflow for Dynamic Data Repair under Adversarial Attack

in providing a novel data repair framework to defend against such kind of adversarial attacks as shown in Figure 4.1. This framework extends our prior work [85] of a cloud-supported platform for sensor networks (e.g, smart grid networks) to formalize general resilience testing procedures under adversarial settings using the model-driven approach [32]. To the best of our knowledge, this work is the first to introduce such a kind of dynamic data repair against adversarial attacks [72], and make the following contributions in this paper.

- We present a framework that can formalize the security and resilience testing in distributed sensor networks under adversarial settings;
- We design an iterative dynamic data repair scheme of Dropout-Detect-Reconstruct-Tradeoff to boost the robustness of data using the DDDAS paradigm for ongoing predictions; and
- We conduct a case study for distributed power network load forecasting to demonstrate potential risks

for machine learning predictors and the efficiency of our defensive data repair framework.

This chapter is organized as follows. Section 4.2 illustrates the theoretical background of our adversarial attack setting and dynamic data repair framework in a step-by-step manner. Section 4.3 presents a case study to demonstrate the capabilities of our framework on a power distribution network. Finally, Section 4.4 concludes the paper and presents opportunities for future research.

4.2 Methodology

In this section we provide details of our approach. The techniques will be introduced following the execution order of attack and defense. Our predictor is based on deep learning. Specifically, the model absorbs data from distributed sensors and fetches their values from current and some time steps back to predict the total system load for the next time step.

4.2.1 Model of Stealthy Adversarial Attacks

There are two important feature for a successful adversarial attack. First, the attack should not be 'obvious' enough to be detected by the system it is attacking. Secondly, the attack should lead to 'obvious' performance deviation in the compromised prediction system.

One of the most well-known and popular adversarial attack is the FGSM (Fast Gradient Sign Method) [36] which formulates the optimization incorporating these two constraints using only one single equation:

$$\eta = \varepsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{4.1}$$

Here θ represents the parameters of the model, x represents inputs to the model, y refers to the targets associated with x (for tasks with targets) and $J(\theta, x, y)$ is the goal loss function for deviating the neural network. The magnitude constraint added to the original sample is represented by ε. This method is quite simple and intuitive. The attacker add fixed magnitude perturbations to maximize the loss function.It is worth pointing out this attack method regards the requirement of being stealthy as self-evident under the fixed magnitude constraint of ε. This makes sense for computer vision cases where colorful images usually have high pixel values(usually 255 max) and the degree change under this constraint is not very different from the the range times a constraint ratio. For a realistic CPS scenario, settings become more complex with a data input space that is potentially bigger than a fixed range as well as the adoption of anomaly detectors. As a result, we reformulate an adversarial attack as an optimization problem which attempts to find the best synthetic perturbations that maximize the prediction loss while keeping the modification magnitude at a small enough level so as to go undetected.

We adapt the general adversarial attack to CPS taking flexible settings into consideration with *Algorithm 2* from our prior work[85]. Compared to the FGSM attack discussed above, this algorithm shows an iterative attack that allows each meter (value in input data array) to have its unique modification value because the input range may not be fixed. Therefore, the deviation is denoted as a ratio rather than a fixed value. And the number of meters allowed to be modified is also limited by l_0 Norm. We choose iterative attacks rather than single-step attacks because the latter suffer from two limitations. First, the non-linearity of the neural network itself makes it impossible for a single-step to reach the optimal loss increase. Secondly, chances of being detected for these modified sensor would be increased with large modifications. Catering to these limitations, we design this iterative attack for a more optimal perturbation.

In *Algorithm 2*, for each $\boldsymbol{x}_0$, our approach performs $NumIter$ number of iterations with maximum deviation ratio D, takes steps of step length ratio $(1+D)^{(1/NumIter)}$, and computes $\Delta\boldsymbol{x}$ using the gradient sign method from the output of previous iteration. With the existence of a detector, intermediate results are first checked with the detector to remove exposed parts and then sent into the next iteration for further exploration.

Algorithm 3 Iterative Stealthy Attack

Require: $\boldsymbol{x}$: original observation; f: predictor; d: detector; $J(f,x)$: cost function of predictor f according to input data x; D: max. deviation ratio; N: number of sensors allowed modifying; $allowModN$: data selector to get values from a set of meters; $NumIter$: number of iterations.

$\Delta\boldsymbol{x} \leftarrow \mathbf{0}$
$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
$\alpha \leftarrow (1+D)^{(1/NumIter)}$
$i \leftarrow 0$
while $i < NumIter$ **do**
  $grad0 \leftarrow \nabla J(f,\boldsymbol{x})$
  $grad \leftarrow allowModN(grad0, N)$
  $\Delta\boldsymbol{x} \leftarrow \alpha * \boldsymbol{x} * sign(grad)$
  $\Delta N \leftarrow d(\boldsymbol{x} + \Delta\boldsymbol{x})$
  $grad \leftarrow allowModN(grad0, N - \Delta N)$
  $\Delta\boldsymbol{x} \leftarrow \alpha * \boldsymbol{x} * sign(grad)$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \Delta\boldsymbol{x}$
  $i \leftarrow i + 1$
end while
return $\boldsymbol{x}$

In summary, Algorithm 2 is a hybrid attack combining L_0, L_∞ and the detection threshold constraints. The attacker is allowed to modify a limited number of meter readings with a maximum deviation ratio D. For each input $\boldsymbol{x}_0$, we modify the most sensitive but undetected meters and use their gradient values to maximize the loss.

4.2.2 Resilient Detection and Reconstruction

To mitigate adversarial impacts, first we need to be able to detect the adversarial data. There are two levels of detection for adversarial attack on systems. The first one is a binary classifier to detect whether the input data is under adversarial attack. The much deeper one is to detect what exact features in the input data are adversarially modified. Previous work utilizing uncertainty in prediction results have shown high accuracy in the first type of detection [81]. Our problem belongs to the more difficult second type and aims to detect whether an exact sensor(with subset data) in the network is under attack.

One thing worth noticing is that applying dropout to adversarial example detection essentially utilizes incomplete information of the system and therefore brings in the potential risk of false alarms. A widely adopted strategy is to set thresholds for the detection results. Only when the sensor has been detected as anomaly for more than a certain number of times will it be regarded as anomaly in this detector.

Based on all these discussions above, we design a general resilient detection strategy against adversarial examples as shown in Algorithm 4. The detector takes input data and conducts a resilient detection for $dctIter$ iterations. After iterations of detections results will be gathered together and sensors detected as anomaly for more than $dctThres$ times will be returned as anomaly.

Algorithm 4 Resilient Detection (*resDetect*)

Require: $\boldsymbol{x}$: original observation; f: predictor; d: detector; AE: auto-encoder self-representation model; $dctIter$: number of detection iteration; $dctThres$: detection iteration threshold; $FrequencyCount$: counter function to return index with a frequency higher than a threshold; P_R: detection dropout rate.

$i \leftarrow 0$
$idxAdv \leftarrow EmptyList$
while $i < dctIter$ **do**
  $\boldsymbol{x}' \leftarrow DropOut(\boldsymbol{x}, P_R)$
  $idxAdv \leftarrow idxAdv.append(d(AE(\boldsymbol{x}') - \boldsymbol{x}))$
  $i \leftarrow i + 1$
end while
$idxAdv \leftarrow FrequencyCount(idxAdv, dctThres)$
return $idxAdv$

Above we propose a resilient detection method against adversarial examples in a regression problem on a sensor network. However, the overall goal is to minimize the predictor loss and make the prediction results robust against adversarial attacks. From a coding point of view, resilient detection with randomization elements can also be regarded as using a part of data to check the remaining part. The biggest difference between resilient detection here and the error-correction code is the complexity of the data involved. In communication systems usually binary data is considered while in the general learning and inference case here, data can be real values and thus more difficult to analyze and modify.

We propose an approximate method here to make modified data closer to the original normal one. As the detector itself is an auto-encoder learning to mapping to the data distribution itself, we can use this mapping to

reconstruct the likely adversarial parts. We can then resend this reconstructed data to go through the resilient detection and reconstruction again. We can either set a fixed number of reconstruction iterations or continue this process until the reconstructed data converges. This resilient correction is inspired by low-density parity checking(LDPC) from coding in communication theory[86]. This whole pipeline is shown in Algorithm 5.

Algorithm 5 Resilient Correction (*resCor*)

Require: $\boldsymbol{x}$: original observation; f: predictor; d: detector; AE: auto-encoder self-representation model; $NumCor$: number of correction iteration; $resDetect$: resilient detection function from Algorithm 4.

$ret \leftarrow \boldsymbol{x}$
$idxAdv \leftarrow EmptyList$
$i \leftarrow 0$
while $i < NumCor$ **do**
  $idxAdv \leftarrow resDetect(\boldsymbol{ret})$
  $tmp \leftarrow ret$
  $ret[idxAdv] \leftarrow 0$
  $ret[idxAdv] \leftarrow \frac{1}{2}(AE(ret)[idxAdv] + tmp[idxAdv])$
  $i \leftarrow i+1$
end while
return *ret*

4.2.3 Iterative Dynamic Repair

The resilient detection and reconstruction procedure is configurable and sensitive to measurements. One key property for prediction tasks like load forecasting is that as new data flows in continuously, the system can utilize new data to validate the quality of previous predictions for which the DDDAS paradigm [84] is best suited to provide adaptive data repair against adversarial attacks as shown in Algorithm 6.

For the resilient detection and reconstruction, given a fixed dropout rate, the sensitivity can be adjusted with the number of detection iteration ($dctIter$) and the detection iteration threshold ($dctThres$). Given the infinite number of combination settings for the resilient detection, we consider three settings with the least computation burden (sensitivity from high to low):

- $x1in2t \leftarrow resCor(x, dctIter = 2, dctThres = 1)$
- $x1in1t \leftarrow resCor(x, dctIter = 1, dctThres = 1)$
- $x2in2t \leftarrow resCor(x, dctIter = 2, dctThres = 2)$

We implement adjustments in iterative time steps to seek a balanced trade-off between sensitivity levels. These three settings lead to three detection and reconstruction sensitivity levels from high to low. And we implement adjustments in iterative time steps to seek a balanced trade-off between sensitivity levels. As a result, the overall prediction result with dynamic repair is computed as a weighted sum of these three resilient reconstructions[38]. For each time step, the system checks the previous prediction deviations from these three

Algorithm 6 Dynamic Repair (*dynRepair*)

Require: $\boldsymbol{x}$: original observation data flow; f: predictor; $NumTime$: number of execution time steps; $resCor$: resilient correction function; $ErrThres$: ideal prediction error threshold; $return$: return function for each time step; y: ground truth value.

$\alpha = [1.0, 0.0, 0.0]$, $\alpha_{bias} = 0.05$, $x \leftarrow \boldsymbol{x}[0]$,$t \leftarrow 1$
$pred, pred1in1, pred1in2, pred2in2 \leftarrow EmptyList$
$x1in1t \leftarrow resCor(x, dctIter = 1, dctThres = 1)$, $pred1in1.append(f(x1in1t))$
$x1in2t \leftarrow resCor(x, dctIter = 2, dctThres = 1)$, $pred1in2.append(f(x1in2t))$
$x2in2t \leftarrow resCor(x, dctIter = 2, dctThres = 2)$, $pred2in2.append(f(x2in2t))$
$pred[0] \leftarrow pred1in1t * \alpha[0] + pred1in2t * \alpha[1] + pred2in2t * \alpha[2]$
while $t < NumTime$ **do**
  $resPre1 \leftarrow abs(pred[t-1] - y[t-1]$, $resPre2 \leftarrow abs(pred[t-2] - y[t-2]$
  if $t > 1$ **and** $resPre1 > ErrThres$ **and** $resPre1 > resPre2$ **then**
    $res1in1 \leftarrow abs(pred1in1[t-1] - y[t-1])$
    $res1in2 \leftarrow abs(pred1in2[t-1] - y[t-1])$
    $res2in2 \leftarrow abs(pred2in2[t-1] - y[t-1])$
    $idx = argmin([res1in1, res1in2, res2in2])$
    $\alpha \leftarrow \alpha - \alpha_{bias}$, $\alpha[idx] \leftarrow \alpha[idx] + 3 * \alpha_{bias}$
  end if
  $x \leftarrow \boldsymbol{x}[t]$
  $x1in1t \leftarrow resCor(x, dctIter = 1, dctThres = 1)$, $pred1in1.append(f(x1in1t))$
  $x1in2t \leftarrow resCor(x, dctIter = 2, dctThres = 1)$, $pred1in2.append(f(x1in2t))$
  $x2in2t \leftarrow resCor(x, dctIter = 2, dctThres = 2)$, $pred2in2.append(f(x2in2t))$
  $pred[t] \leftarrow f(x1in1t) * \alpha[0] + f(x1in2t) * \alpha[1] + f(x2in2t) * \alpha[2]$,
  $return(pred[t])$,$t \leftarrow t + 1$
end while

levels and allocate higher weights on the least deviated reconstruction level. This weight update occurs when the deviation of the last step is larger than an expected threshold and the overall prediction deviations for the last two past time steps get an increase. In this way, the system dynamically adjusts the weights for these three sensitivity levels in an iterative manner and seeks a stable data repair based on reconstructions over the time steps. The full implementation is shown in Algorithm 6.

4.3 Evaluation

4.3.1 Power System Setting

For data collection, we conduct a detailed simulation of an electric distribution system using GridLAB-D provided by the Pacific Northwest National Laboratory (PNNL) [82]. We selected the prototypical feeder of a moderately populated area *R1-12.47-3*, and included representative residential loads like heating, ventilation and air conditioning (HVAC) systems to the distribution network model [83]. In summary, our distribution model has a total of 109 commercial and residential user loads. Smart meters are connected to end users and their usage data reports are transmitted to the upper-level control center in a hierarchical manner. For each hourly time step, the prediction model takes load data from distributed meter readings in the past 24 hours and also takes into account the temperature data for the same period of time. We build a load forecasting

model for this power distribution network using a relatively large LSTM deep neural network (with 3 LSTM layers of 150 units and 2 fully-connected layers of 200 units). The predictor on the clean data generates a mean squared error (MSE) of 0.1255 (Mega Volt Amp) on the test data set for a total of 216 time steps.

The attack scenario is a manipulation of sensor data under reasonable constraints with full knowledge of the prediction and detection model. In each time step, the attacker can manipulate a fixed number of meters in the network ($10\% - 50\%$ in our experiments). Moreover, for each meter, the attacker is allowed to deviate the meter reading by a limited level of 20%. Under these constraints, we generate stealthy adversarial examples using the iterative attack method.

4.3.2 Evaluating Reconstruction and Repair

We evaluate our dynamic data repair framework on various settings. We propose results based on strong attacks with a maximum modification ratio of 20% for compromised sensors.

Figure 4.2-4.6 show the experiment results under different attack rates with 80 detection iterations and 10% detection dropout rate through 216 test time steps. To clearly show the impact of adversarial attack and on the other hand the efficiency of our defense framework, we show results for different settings in two ways. The left Figure (a) shows absolute prediction deviations from normal prediction results. And the right Figure (b) shows mean absolute prediction deviations of current prediction and all ones prior to current time step.

Figure 4.2 shows the prediction results when 10% of sensors in the network are compromised. Figure 4.3 shows the prediction results when 20% of sensors in the network are compromised. We can see that for the pure adversarial data, the mean absolute deviation would be able to reach the level of 0.3(unit Mega Volt Amp) even though only 20% of meter readings are modified. This further proves the significance of the adversarial vulnerability problem we are trying to investigate in this paper. For both these 10% and 20% cases, pure resilient reconstruction gains the best performance. The combination of dynamic data repair still keeps a good enough performance(reducing error to less than $1/2$ of pure adversarial) but leads to a little higher overall deviation than the pure resilient reconstruction.

Figure 4.4 shows the prediction results when 30% of sensors in the network are compromised. This is the same modification setting as the previous discussion shown in Figure 3.5 and Figure 3.7. The combination of dynamic data repair starts to outperform the pure resilient reconstruction defense setting after the very first load peak where the adversarial prediction deviates the most.

Figure 4.5 shows the prediction results when 40% of sensors in the network are compromised. The combination of dynamic data repair has shown an obvious better performance than the pure resilient reconstruction defense setting throughout the working process.

Figure 4.6 shows the prediction results when 50% of sensors in the network are compromised. From

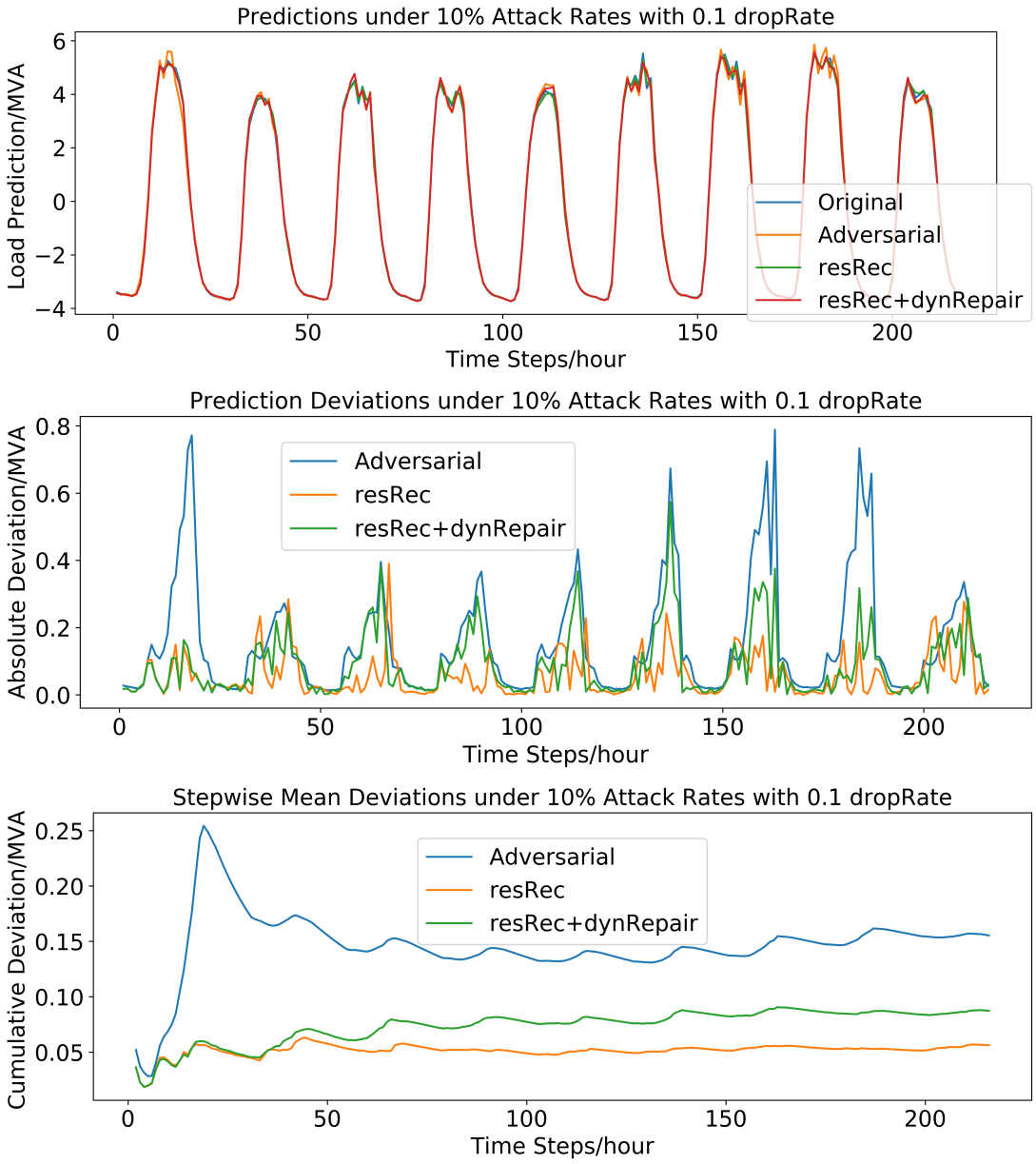

Figure 4.2: Predictions under 10% Compromise and 10% Detection Dropout Rate. From above to bottom, three figures show: (1) Adversarial Regression Results; (2) Absolute Prediction Deviation; (3) Cumulative Mean Absolute Deviation.

our empirical analysis, with the ratio of compromised sensors reaching 50%, the adversarial impact gets an obvious increase and even though the correction procedure mitigates the adversarial impacts it cannot further reduce the mean absolute deviation to the level of 0.3(unit Mega Volt Amp) and from a deployment point of view this might not come up with an overall practical performance. This may also indicate the premise of the resilient detection is the completeness of the data, which will certainly be violated if the ratio of sensors under attack reaches 50%.

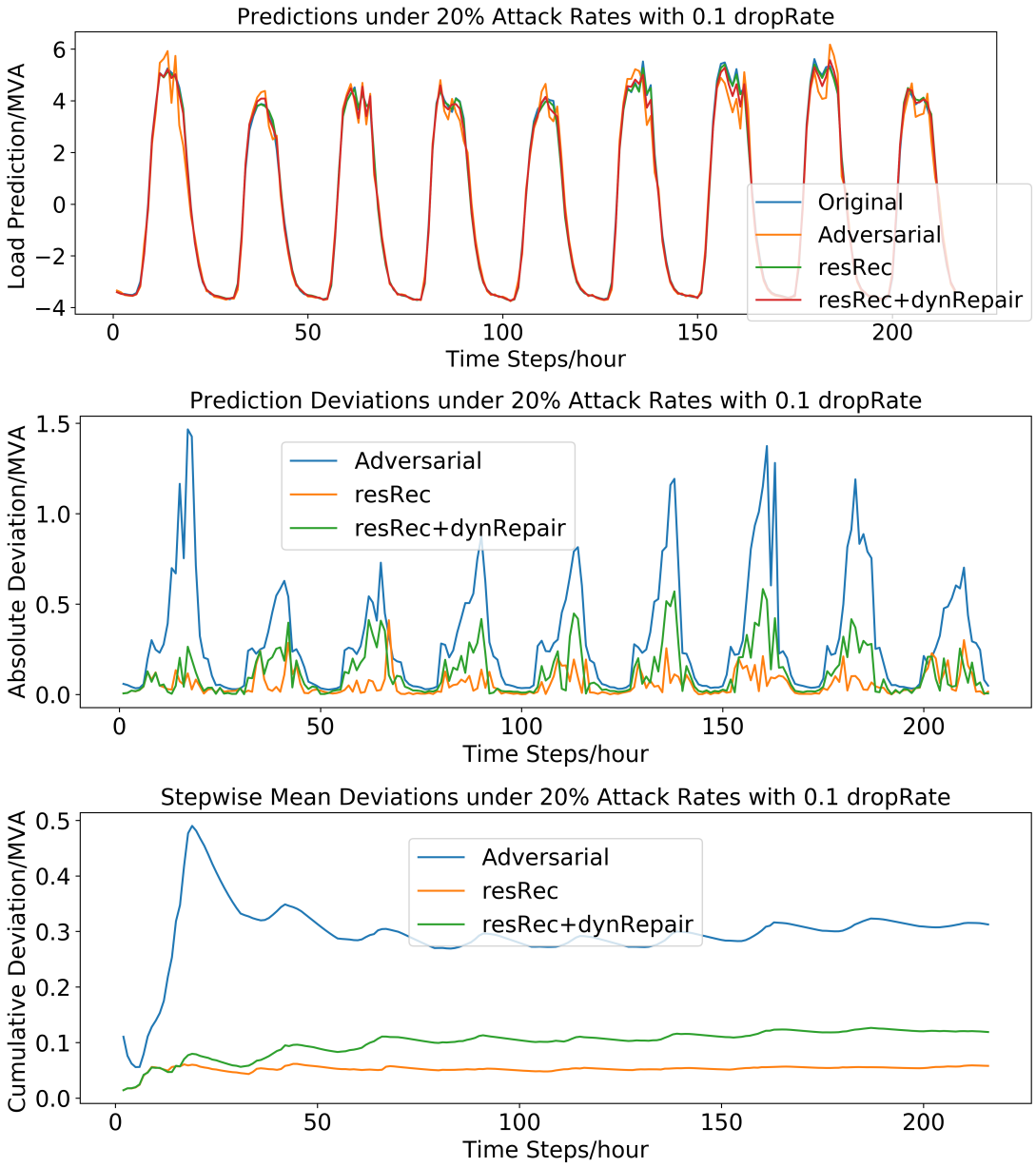

Figure 4.3: Predictions under 20% Compromise and 10% Detection Dropout Rate. From above to bottom, three figures show: (1) Adversarial Regression Results; (2) Absolute Prediction Deviation; (3) Cumulative Mean Absolute Deviation.

We present experiment results under more flexible settings in Table 4.1. The table includes experiment results under four levels of detection dropout rate: $5\%, 10\%, 20\%, 30\%$ with $20, 40, 60, 80$ reconstruction cycles. The error metric we choose to show here is the most commonly used mean squared error(MSE) over the test dataset. For different attack rates, best defense settings are marked in dark black. We can see that low detection dropout rates with more detection cycles usually show more stable prediction performances.

From figures above we can also see that adversarial impacts in this load forecast case usually occurs at

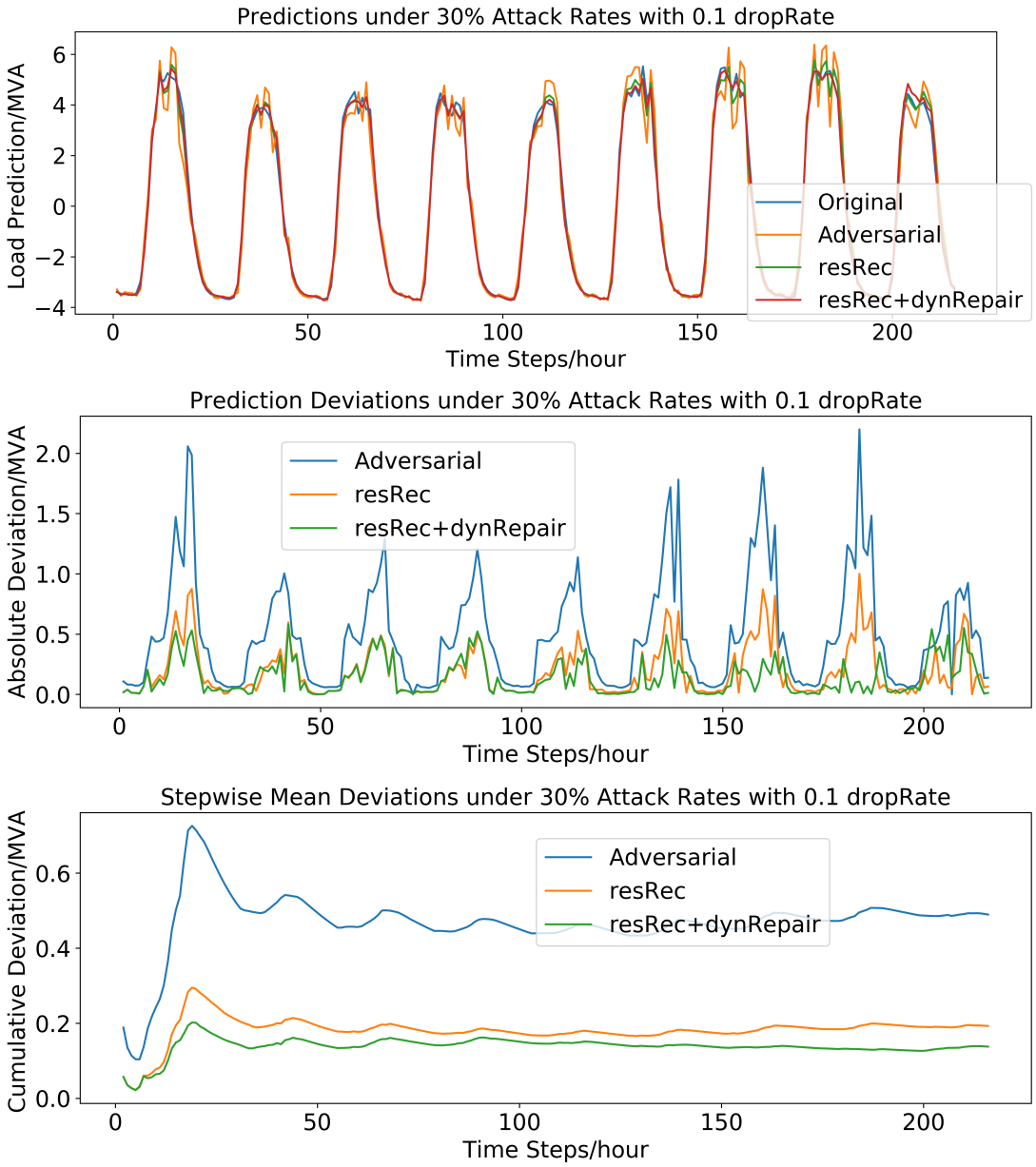

Figure 4.4: Predictions under 30% Compromise and 10% Detection Dropout Rate. From above to bottom, three figures show: (1) Adversarial Regression Results; (2) Absolute Prediction Deviation; (3) Cumulative Mean Absolute Deviation.

peak points. And the data repair framework successfully decreases prediction deviations at these vulnerable points without much impact on other locations. On the other hand, these experiment results also clearly show the trade-off caused by the iterative data repair. With a large number of detection iterations, the chance of being totally stealthy for an adversarial sensor is reduced to a neglectable level. Meanwhile, low threshold settings lead to obvious negative impacts caused by false alarms. And this sensitive repair might lead to an unstable prediction performance over time steps. As shown in our experiments, this potential risk is most

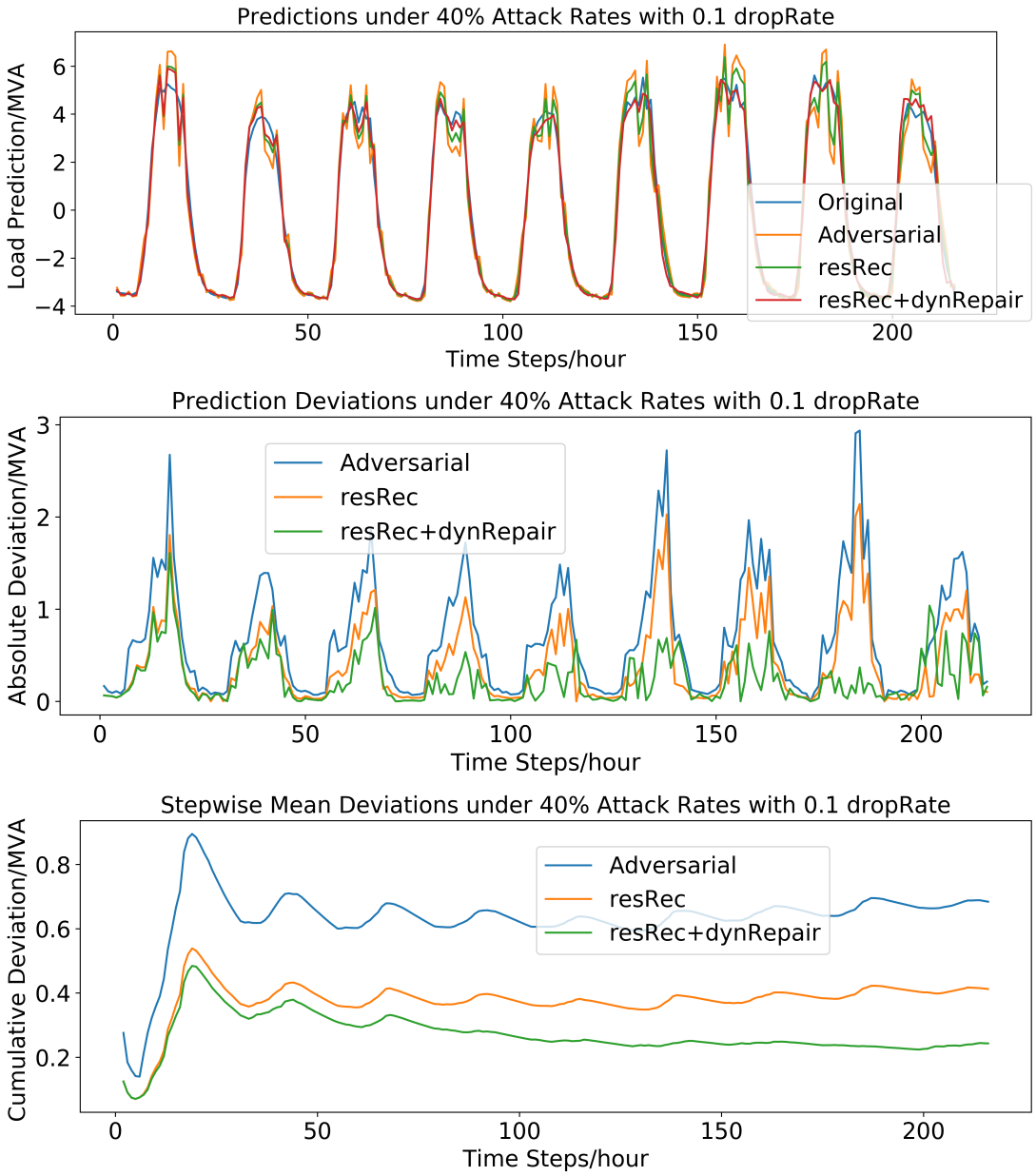

Figure 4.5: Predictions under 40% Compromise and 10% Detection Dropout Rate. From above to bottom, three figures show: (1) Adversarial Regression Results; (2) Absolute Prediction Deviation; (3) Cumulative Mean Absolute Deviation.

obvious when the detection dropout rate is high. As a result, the combination of a relative low detection dropout rate along with more iterations would usually lead to smoother and more stable performances.

In summary, we show the efficiency of our dynamic data repair framework. One important feature of our framework is that it takes advantage of existing pre-trained models in a resilient way. This means it can be combined with other defense techniques with no barrier. Furthermore, it is a generalized prediction model

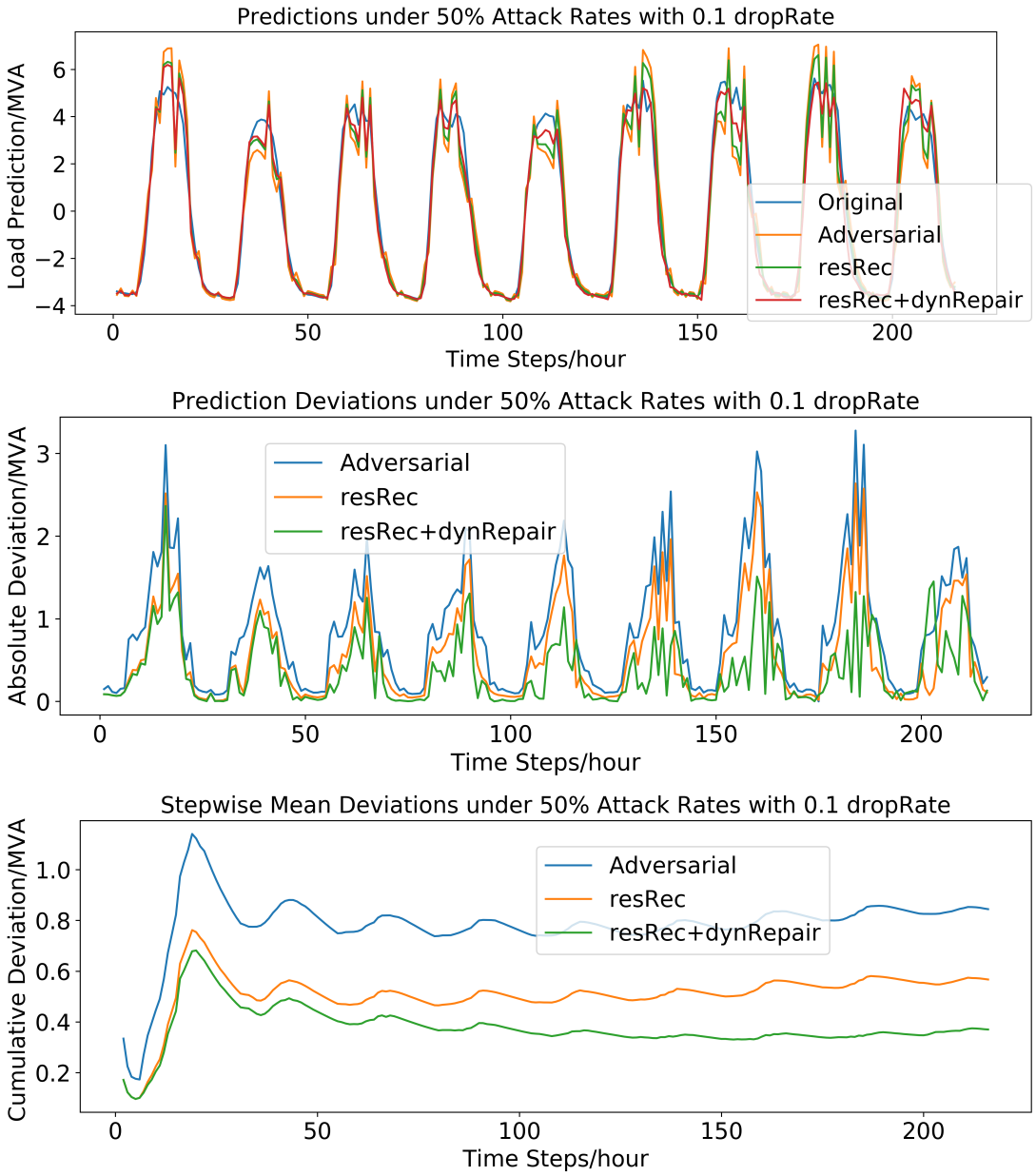

Figure 4.6: Predictions under 50% Compromise and 10% Detection Dropout Rate. From above to bottom, three figures show: (1) Adversarial Regression Results; (2) Absolute Prediction Deviation; (3) Cumulative Mean Absolute Deviation.

deployment strategy to improve robustness that could be easily applied to various learning settings.

4.4 Conclusion

This paper demonstrated how to analyze and improve the robustness of learning-based prediction models in power distribution networks using the DDDAS paradigm. Given the existence of threats from stealthy adversarial attacks, we first designed a resilient detection and reconstruction strategy using randomization

Table 4.1: Prediction Mean Squared Error(MSE) Under Different Attack and Defense Settings

drop	adv /%	natErr	advErr	resRec/numCycle				resRec+dynRepair/numCycle			
				20	40	60	80	20	40	60	80
/%	10	0.126	0.173	0.152	0.144	0.142	**0.141**	**0.146**	0.147	0.150	0.149
	20	0.126	0.311	0.211	0.163	0.148	**0.143**	0.170	0.159	0.155	**0.149**
5	30	0.126	0.538	0.380	0.300	0.257	**0.232**	0.281	0.216	0.197	**0.188**
	40	0.126	0.921	0.729	0.626	0.559	**0.523**	0.566	0.442	0.345	**0.301**
	50	0.126	1.329	1.090	0.979	0.909	**0.862**	0.876	0.719	0.581	**0.500**
	10	0.126	0.171	0.139	**0.137**	0.138	0.139	0.152	0.148	**0.146**	0.147
	20	0.126	0.311	0.174	0.144	**0.139**	0.139	0.158	**0.150**	0.152	0.168
10	30	0.126	0.538	0.310	0.236	0.211	**0.200**	0.224	0.183	**0.179**	0.179
	40	0.126	0.921	0.632	0.524	0.481	**0.464**	0.406	0.293	0.270	**0.267**
	50	0.126	1.329	0.984	0.859	0.808	**0.784**	0.688	0.526	0.477	**0.455**
	10	0.126	0.173	**0.139**	0.146	0.145	0.145	**0.140**	0.153	0.150	0.151
	20	0.126	0.311	0.142	0.139	**0.138**	0.171	**0.160**	0.300	0.308	0.286
20	30	0.126	0.538	0.229	**0.201**	0.216	0.218	**0.182**	0.192	0.229	0.273
	40	0.126	0.921	0.541	0.473	0.459	**0.457**	0.912	0.994	0.803	**0.760**
	50	0.126	1.329	0.850	0.777	0.767	**0.759**	0.567	**0.527**	0.559	0.579
	10	0.126	0.173	**0.139**	0.140	0.139	0.139	0.147	0.140	**0.138**	0.142
	20	0.126	0.311	**0.138**	0.138	0.139	0.139	0.150	0.148	0.139	**0.139**
30	30	0.126	0.538	0.219	0.201	0.202	**0.201**	0.197	**0.184**	0.197	0.213
	40	0.126	0.921	0.521	0.491	0.488	**0.485**	**0.378**	0.388	0.429	0.462
	50	0.126	1.329	0.876	0.842	**0.840**	0.838	**0.598**	0.644	0.699	0.786

elements. We then proposed a practical, iterative dynamic data repair strategy to seek an optimal trade-off between reconstruction results from different sensitivity levels. Our work not only shows the importance of introducing randomization elements to increase robustness in learning-based systems but also the effectiveness of deviation feedback for predictions on-the-fly. The limitation of this defense framework is also clear. Even though our defense framework has shown promising results, the computation cost for an optimal defense efficiency can be very high thereby requiring new approaches to simplify and accelerate computations for real time applications.

CHAPTER 5

Adversarial Data-Driven Prognostics

Deep learning has shown impressive performance across a variety of domains, including data-driven prognostics. However, research has shown that deep neural networks are susceptible to adversarial perturbations, which are small but specially designed modifications to normal data inputs that can adversely affect the quality of the machine learning predictor. We study the impact of such adversarial perturbations in data-driven prognostics where sensor readings are utilized for system health status prediction including status classification and remaining useful life regression. We find that we can introduce obvious errors in prognostics by adding imperceptible noise to a normal input and that the hybrid model with randomization and structural contexts is more robust to adversarial perturbations than the conventional deep neural network. Our work shows limitations of current deep learning techniques in pure data-driven prognostics, and indicates a potential technical path forward. To the best of our knowledge, the work in this chapter is the first to investigate the implications of using randomization and semantic structural contexts against current adversarial attacks for deep learning-based prognostics.

5.1 Problem Overview

Machine learning has gained much attention to solve a variety of challenges in cyber-physical systems (CPS). Data-driven methods, particularly deep learning, have made deep strides in health management and prognostics, such as anomaly detection and remaining useful life estimation. For a relatively complex system with a number of sensors, the service provider can utilize data flows from multiple smart data sources to perform data-driven prognostics using a deep learning model. However, recent research in prognostics has shown that statistical learning methods, e.g., deep neural networks, are susceptible to small adversarial perturbations, which are small but specially designed modifications to normal data inputs that can adversely affect the quality of the machine-learned predictor [87]. For instance, owing to the deployment of external sensors in many application scenarios, an attacker could intercept and maliciously modify sensor readings to conduct these kinds of adversarial attacks, which could subsequently lead to critical damage caused due to inaccurate system health status evaluation.

We study the impact of such adversarial perturbations in data-driven prognostics as shown in Figure 5.1. Data-driven prognostics methods are suitable for scenarios where complete analytical models of the physics are difficult or impossible to formulate. In this context, recent research has shown the success of combining physics-based semantic information [28] with pure data-driven prognostics. For this paper, two typical prog-

nostics problem settings comprising classification and regression are considered [23]. In the classification setting, the model predicts the remaining useful life of the system in a certain range and outputs a categorical result to indicate to which health status category the system belongs to. In the more general regression setting, the model outputs a numerical prediction value for the remaining useful life.

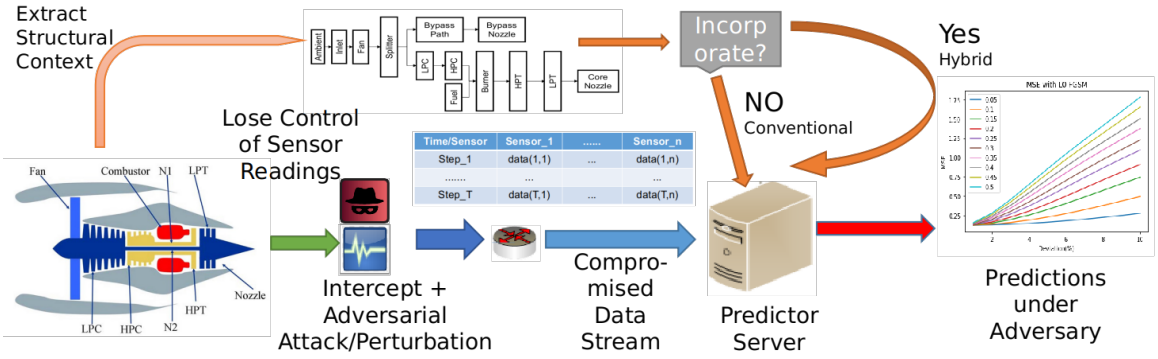

Figure 5.1: Overall Workflow for Robustness Evaluation of Deep Learning-based Prognostics on Conventional & Hybrid Models (with Randomization and Structural Contexts)

We demonstrate our ideas using the widely-used C-MAPSS turbine engine dataset [88].We explore the vulnerability of prediction models and potential ways to defend against adversarial attacks.To the best of our knowledge, this work is the first to investigate the implications of using semantic structural contexts against current adversarial attacks for deep learning-based prognostics [89]. We find that we can introduce obvious errors in prognostics by adding imperceptible noise to a normal input and that the model involving randomization and structural contexts is more robust to adversarial perturbations. Our work highlights the limitations of current deep learning techniques in pure data-driven prognostics, and presents a potential technical path forward.

We make the following contributions in this paper:

- We present a framework that can formalize the security and resilience testing in data-driven prognostics settings.
- We show the vulnerability of deep learning prognostics models under various settings.
- We investigate the possibility of inducing randomization elements and semantic structural context to mitigate adversarial impacts.
- We conduct a robustness case study using the engine degradation dataset on typical applications of health status classification and remaining useful life prediction.

The rest of the chapter is organized as follows. Section 5.2 provides the motivation for evaluating and mitigating security risks in data-driven prognostics. Section 5.3 illustrates the theoretical background for our

adversarial attack and defense settings. Section 5.4 presents a case study to demonstrate the capabilities of our framework on the C-MAPSS turbine engine dataset [90]. Finally, Section 5.5 concludes the paper and alludes to future research directions.

5.2 Motivation

It is essential for service providers to maintain a capability to monitor and predict the health status of their system. Two primary system modeling technical paths have been widely used including model-based and data-driven. Classical model-based methods assume that the model must be accurate enough to depict system behaviors, e.g., Bond Graphs [24] or analytical battery physical systems [91]. These methods need to know formally how different components of a system interact and how the system output can be computed analytically. For many real-world cases, however, due either to partial knowledge or the system's large-scale, it becomes impossible to reveal the pattern in a detailed analytical way. Consequently, as the complexity of the system increases and the amount of generated data volumes increases, data-driven health management and prognostics are preferred [92].

Recent advances in machine learning, particularly deep neural networks, enable a lower threshold for building a prediction model for health management and prognostics. However, deep neural networks are susceptible to adversarial attacks, which are a small amount of additional data designed as perturbations to misguide the original neural network prediction systems. Recently, the impact of adversarial examples in deep learning [34] has given rise to many concerns [36].Prior research [7] has shown how these adversarial examples can pose threats to current machine learning systems. Most prior work in the field of adversarial machine learning during the past decade has paid attention to classification tasks [35]. As regression tasks start playing an increasingly important role in CPS scenarios, the topic of adversarial regression is attracting more research attention.

For the health management and prognostics field, deep learning techniques have shown to be successful in a number of tasks like power disturbance classification [5] and remaining useful life prediction [93]. Consequently, even though a health prediction model could be built easily using state-of-the-art deep neural networks and tool-flows, a more cautious view still needs to be taken due to potential risks of adversarial attacks with these kind of learning-based components [87].

In a nutshell, although accurate status classification or prognostics is critical for efficient data-driven health management, the vulnerability in this broad practical scenario has not been carefully investigated to date. To address these issues in the prognostics and health management domain, we propose an approach to show model vulnerabilities from domain-specific settings and explore some potential ways to mitigate underlying adversarial impacts.

5.3 Methodology

In this section we delve into the details of the underlying methods used in our framework. These methods will be introduced in a step-by-step manner following the evaluation execution path of attack and defense on neural network predictors.

5.3.1 Adversarial Attack

As discussed above, attacks on neural network-based prognostics are essentially small adversarial data modifications. There are two important features for a successful adversarial attack. First, the attack perturbation should not be 'obvious' enough to be detected by the system that it is attacking. Secondly, the attack should lead to 'obvious' performance deviation in the compromised prediction system.

Overall, given a model $f(\cdot)$, one well-recognized way to define an adversarial attack [94] is the worst-case target loss L for a given perturbation budget defined by an ε bounded distance magnitude $D(x,x')$ between the original data point x and the perturbated data point x'.

$$\max_{x':\mathcal{D}(x,x')<\varepsilon} L\big(f(x'),y\big) \tag{5.1}$$

It is worth pointing out that the attack methods often regard the requirement of being stealthy as self-evident under the fixed maximum magnitude constraint of ε. This can be generalized to data with or can be pre-processed into a fixed range using techniques like MinMax Normalization [95].

5.3.1.1 Gradient-based Attack

Based on the two features above, we can regard an adversarial attack against a prediction model as an optimization problem. The goal of this optimization problem is to solve for an adversarial perturbation on the original data input that on one hand maximizes the target loss function and on the other hand be under some realistic constraints. Here we consider a white-box setting where an attacker can obtain full knowledge of the prediction model. In this way, the attack procedure would be a gradient-based optimization computation using the loss function. We select two typical adversarial attack methods of Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD).

FGSM (Fast Gradient Sign Method) [36] is one of the most well-known and popular adversarial attack methods. It formulates the optimization problem incorporating these two constraints using only one single equation:

$$x' = clip_{[0,1]}\{x+\varepsilon\cdot sign(\nabla_x L\big(f(x'),y\big)\} \tag{5.2}$$

Here, x represents inputs to the model and is assumed to be in the range of $[0,1]$, y refers to the targets associated with x (for tasks with targets) and $L\big(f(x'),y\big)$ is the goal loss function for deviating the neural network predictor $f(\cdot)$. The magnitude constraint added to the original sample is represented by ε. This method is quite simple and intuitive. The attacker adds fixed magnitude perturbations to maximize the loss function.

PGD (Projected Gradient Descent Method) [96] is recognized as a universal attack procedure and by far one of the strongest attack methods. Instead of starting from the exact data point, it starts from a random perturbation in the $\|l\|_p$ norm ball around the input sample. It then utilizes the FGSM approach but with a much smaller gradient step towards the greatest loss and projects the perturbation back into the allowed $\|l\|_p$ ball range. In contrast to one-step FGSM, PGD iteratively adds small perturbations using updated gradients. With the same maximum level magnitude of the perturbation, PGD proves to be much stronger than FGSM.

5.3.1.2 Adversarial Goal Setting

In a real-world CPS scenario, the settings become more complex with a data input space that is potentially larger than a fixed range. As a result, we reformulate an adversarial attack as an optimization problem which attempts to find the best synthetic perturbations that maximize the prediction loss while keeping the modification magnitude at a small enough level so as to go undetected.

In CPS settings not only do input data formats show more complex patterns, the adversarial attack goals can also be more flexible than, say, computer vision classification tasks. For an adversarial regression scenario like remaining useful life prediction, we propose three adversarial attack settings using corresponding loss functions: **Mean Squared Error** (to maximize absolute prediction deviation), **Maximization** (to maximize prediction value) and **Minimization** (to maximize prediction value). The latter two are not commonly discussed but would be extremely meaningful for health management and prognostics application scenarios where the cost of underestimating or overestimating could be high and unbalanced [90].

5.3.2 Randomization-based Defense

Although there has not been an overall defense technique for existing adversarial attacks [97], previous robustness improvement works that induced randomization elements have shown a high success rate in terms of detecting adversarial examples [81]. As redundancy design is a significant ideology toward higher system reliability, it is intuitive to combine this internal ideology with a probabilistic implementation. Therefore, we choose two typical randomization-based methods and further propose a potential way utilizing semantic structural contexts to help mitigate adversarial impacts in data-driven prognostics.

5.3.2.1 Gaussian Augmented Adversarial Training

One mainstream defense technique that has attracted much attention is improving model robustness during the training phase. Research has shown that data augmentation during training using Gaussian data augmentation [41] helps in improving robustness of neural networks to adversarial attacks. The intuition behind this is straightforward: in the high dimensional space, data points within a $\|l\|_p$ ball range should own the same label. Based on this assumption, the original training dataset can be extended by adding a norm-bounded Gaussian noise but with the same label. The training based on this extended dataset generates more resilient models against small perturbations.

5.3.2.2 Input Dropout Inference

In the inference phase, there has been research [43] showing that random cropping or random padding with resizing of the adversarial examples reduces their effectiveness. Furthermore, randomly deactivating input neurons in the inference phase [42] has also been discussed. For the model deployment strategy, a recent work [44] adds small random noises to one input for several predictions and suggests the ensemble of prediction results with the highest probable class. However, these methods have only been under trial on computer vision and classification tasks and cannot be applied to other scenarios directly.

5.3.2.3 Generalized Random Ensemble

Based on the above discussions, we design a generalized model application strategy against adversarial examples using randomization elements applicable for both classification and regression problems.

The system holds a predictor and a self-representative auto-encoder detector for reconstruction. Given a data input, the system randomly sets input feature values to zero at a dropout rate level P_R. We discuss two kinds of dropout here: (1) Normal Dropout with a constant rate on all features and (2) *SemanticDropout* using system structural contexts to assign features with different dropout rates as shown in Algorithm 7. The intuition behind this *SemanticDropout* is to decouple most correlated features from the same sub-component that are most likely to be adversarially modified in the same direction. Then the data with deactivated values is sent to a self-representative auto-encoder to recover the matrix. This recovered data is then used for prediction. This $\{Drop => Reconstruct => Predict\}$ procedure is conducted repeatedly for $nIter$ iterations and generates $nIter$ sets of predictions.

To deal with the classification problem, the output matrix contains likelihood values for classes. We sort these likelihood values for each class. The likelihood for one class is computed as the sum of the results of the highest $nThres$ rounds. The overall classification result refers to the class with the highest sum value.

To deal with regression problem, the system first uses some training data with the trained model and

generate some adversarial examples using deviation maximization loss function (like mean squared error). These adversarial data from training data would be used to decide whether the model is more likely to be adversarially-maximized or adversarially-minimized or equal likely. Given that the output matrix contains numerical prediction values from the step above, we sort these prediction values for *nIter* rounds. If the model is adversarially-maximized, the final prediction result would be the mean value of the smallest $nThres$ prediction results. In contrast, if the model is adversarially-minimized, the final prediction result would be the mean value of the largest $nThres$ prediction results. Otherwise, the final prediction result would be the mean value of the median $nThres$ prediction results.

Algorithm 7 Semantic Dropout

Require: $\boldsymbol{x}$: original observation with $nFeature$ feature rows; $adjMat$: adjacency matrix of features (same sub-component features regarded as adjacent); $CorrCoef$: correlation matrix computation function; Dot: dot product; $Dropout$: normal dropout function; $MinMaxScale$: function to scale inputs into a range between a min and max value; P_R: base dropout rate.

$\boldsymbol{xcorr} \leftarrow Abs(CorrCoef(\boldsymbol{x}))$
$Diag(\boldsymbol{xcorr}) \leftarrow 0$
$Diag(\boldsymbol{xcorr}) \leftarrow MaxByRow(\boldsymbol{xcorr})$
$\boldsymbol{adjCorr} \leftarrow Dot(adjMat, \boldsymbol{xcorr}) + Dot(\boldsymbol{xcorr}, adjMat)$
$\boldsymbol{adjCorrMean} \leftarrow MeanByRow(\boldsymbol{adjCorr})$
$\boldsymbol{adjCorPr} \leftarrow MinMaxScale(\boldsymbol{adjCorrMean}, min = 0.5P_R, max = 1.5P_R)$
$\boldsymbol{xSemDrop} \leftarrow \boldsymbol{x}.copy()$
while $i < nFeature$ **do**
  $\boldsymbol{xSemDrop}[i] \leftarrow DropOut(\boldsymbol{x}[i], \boldsymbol{adjCorPr}[i])$
  $i \leftarrow i + 1$
end while
return $\boldsymbol{xSemDrop}$

5.4 Evaluation

For demonstration purposes, we conduct a case study using the widely-used C-MAPSS jet engine degradation dataset [88]. It is worth pointing out that the proposed attack and defense settings as well as implementation methods can also be generalized to other data-driven health management and prognostics settings without any constraints.

5.4.1 Dataset Description

The C-MAPSS dataset [88] was initially generated for the PHM08 Data Challenge. It is a dataset for data-driven remaining useful life (RUL) prediction for jet turbofan engines. The standard version has four sub-datasets consisting of running data of the engine under a certain mode. The engine starts degrading from a time point and breaks down (RUL=0) at the end of the running cycle. Apart from time label, there are 24

features for each data point as shown in Table 5.2. The first three are the three operational settings (does not state exactly what they represent) that have a substantial effect on engine performance.The remaining ones represent the 21 sensor values.Some researchers also use trends to make a initial selection on features [98] to guarantee that the selected features have relative clear trends throughout the degradation process [99]. As we are trying to demonstrate the potential risk of data-driven predictors, we only choose the first set $FD001$ for our experiments here. There are some data preparation steps that could be applied on this dataset including duplicate removal and normalization like MinMax [95] or Batch normalization [100]. We implement MinMax normalization to transform feature values into a fixed range of 0 and 1.

5.4.2 System Models

Since we focus on sequence analysis, we choose the method of long short-term memory (LSTM) network [69]. A recurrent LSTM network enables us to input sequence data into a network, and make predictions per individual time steps of the sequence data. Given its good support for time series, LSTMs have been widely used for data-driven prognostics [101] for more than a decade [102]. Our work makes use of recurrent neural networks in two ways using sensor readings of different components. One is to to predict the health status or remaining useful life. The other is to use an LSTM auto-encoder to help reconstruct data matrices with deactivated zero values after randomization executions. The input dimension is $50 * 24$ indicating 24 features across 50 time steps. For the health predictor, we include two LSTM layers with 100 units. The status classifier has a Sigmoid (binary classification) or Softmax (multi-class classification) activation layer to generate likelihood for classification outputs. The regression predictor is added with a single neuron to compute the numerical output after the LSTM layers. For the auto-encoder, we include two LSTM layers with 50 units followed by a fully-connected layer with 24 units to generate an output with the same shape with the input.

5.4.3 Prediction and Attack Setting

For health management and prognostics applications, the input and output formats vary for different scenarios. For this engine dataset, research can be conducted on both remaining useful life value regression or health status classification [103]. To show the generalized existence of adversarial impacts across different data-driven settings, we consider three tasks with five adversarial settings as shown in Table 5.1. Here, the labels for classification tasks are transformed from the original numerical remaining useful life values.

For the binary classification task, a predictor judges whether the engine is already in the failure state. We consider engines with no more than 15 cycles of life lengths as being in failure state. This predictor outputs a binary result of being true or being false. And as a result, the adversarial attacks make use of the known

Adv Loss	Attack Goal	Outcome
None	$f(x) = y$	Normal Prediction
Binary Crossentropy	$f(x') \neq y$	Misclassify
Categorical Crossentropy		
Minimization	$\min f(x')$	Decreased Prediction
Maximization	$\max f(x')$	Increased Prediction
Mean Squared Error	$\max(f(x') - y)^2$	Increased Absolute Prediction Deviation

Table 5.1: Attack Goal for Perturbated Sample $x' : \mathcal{D}(x, x') < \varepsilon$

model knowledge and use the loss function of **Binary Crossentropy** to lead the predictor to misclassify the modified sample into another class.

For the multiclass classification task, a predictor judges whether the engine is in the steady phase, the degrading phase or the critical phase. We consider engines with no more than 50 cycles of life lengths as being critical, with 50 to 105 cycles as being degrading and with more than 105 cycles as being steady. This predictor outputs a class indicating the phase. As a result, the adversarial attacks make use of the known model knowledge and use the loss function of **Categorical Crossentropy** to lead the predictor to misclassify the modified sample into wrong classes.

For the regression task, a predictor outputs a positive numerical value of the estimated remaining useful life for the given time step of data input. This is the most fundamental task of this dataset. Here the attacker has three options for the attack goal setting using different loss functions:**Mean Squared Error** to maximize absolute prediction deviation, **Maximization** to maximize prediction value and **Minimization** to minimize prediction value.

The attack is a manipulation of sensor data under reasonable constraints with full knowledge of the prediction and detection model. Among the 24 features provided for data points, the first three are control settings and the remaining 21 are sensor readings. We assume the attacker can modify values of sensor readings but the control settings are kept untouched. For the attack strength, we conduct experiments on clean data along with different attack perturbation levels of $advEps = 0.01 - 0.10$. That is equivalent to $1 - 10\%$ of data value range based on the MinMax Normalization. Under these constraints, we generate adversarial examples using the strongly iterative PGD attack(50 step) to maximize the prediction deviation.

5.4.4 Evaluating Attack and Defense

For comparison purposes, we conduct experiments on models trained from only clean natural data and also from Gaussian Augmented data (with a norm bound of 0.10). Moreover, for each model, we include three

No.	Parameter	Detail	Group
1	C1	Control input 1	0
2	C2	Control input 2	0
3	C3	Control input 3	0
4	T2	Total temperature at fan inlet	1
5	T24	Total temperature at LPC outlet	2
6	T30	Total temperature at HPC outlet	3
7	T50	Total temperature at LPT outlet	5
8	P2	Pressure at fan inlet	1
9	P15	Total pressure in bypass-duct	2
10	P30	Total pressure at HPC outlet	3
11	Nf	Physical fan speed	4
12	Nc	Physical core speed	6
13	epr	Engine pressure ratio (P50/P2)	0
14	Ps30	Static pressure at HPC outlet	3
15	phi	Ratio of fuel flow to Ps30	3
16	NRf	Corrected fan speed	4
17	NRc	Corrected core speed	6
18	BPR	Bypass Ratio	2
19	farB	Burner fuel-air ratio	5
20	htBleed	Bleed Enthalpy	0
21	Nf_dmd	Demanded fan speed	4
22	PCNfR_dmd	Demanded corrected fan speed	4
23	W31	HPT coolant bleed	5
24	W32	LPT coolant bleed	5

Table 5.2: Sensor Feature Grouping according to Their Semantic Structural Contexts

settings of no defense, normal dropout and *SemanticDropout*. In this section we first introduce how we incorporate semantic structural contexts and then we show more detailed experimental results.

5.4.4.1 Structural Information Embedding

Without any background knowledge of aircraft engine dynamics, we can get this high-level information from the layout description given by the original CMAPSS dataset [104]. Among 24 column variables, the first three are control inputs and the rest are sensor values. Five features including three control inputs are regarded as conducting impact globally. Further, we use this mapping to build the feature adjacency matrix for the proposed *SemanticDropout* method discussed. Using the semantic context knowledge of the high-level component structure, we can briefly relate features to where they work in the engine and how different features are connected as shown in Figure 5.2. The definition of 24 column features as well as the proposed semantic grouping are also shown in Table 5.2. Features in the same group would be marked as being adjacent to each other. The global impact features would be assumed to be adjacent to all other features.

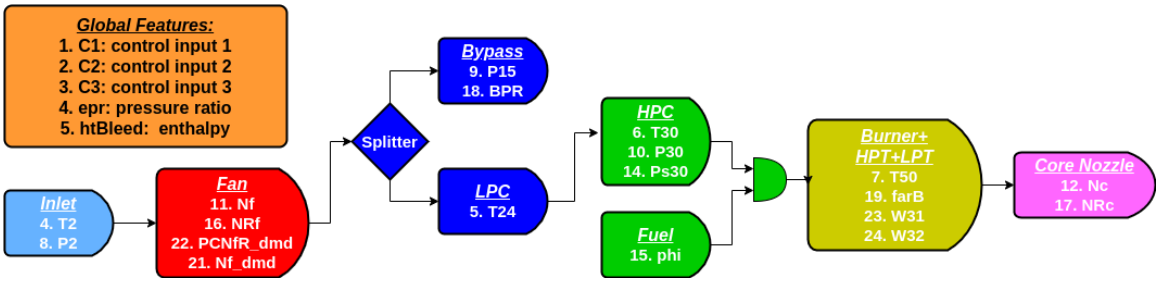

Figure 5.2: Groups of Features Along Layout Components (one color per group)

5.4.4.2 Experimental Results

We conduct experiments on various settings. To evaluate the worst case, Without loss of generality, the number of detection iterations is set to 10 and the detection threshold is set as half of that. For all prediction result figures, the detection dropout rate is set at 40%. And we show results under increasing adversarial attack strengths($advEps$ ranging from 0.01(1%) to 0.10(10%)).

Figure 5.3 shows binary classification accuracy under adversarial attack. Figure 5.4 shows multiclass classification accuracy under adversarial attack. From both these classification cases, the *SemanticDropout* method shows highest adversarial robustness. But there are some obvious differences shown from our experiments. In the multiclass case, the Gaussian Augmented model always shows better robustness whereas in the binary case the Gaussian Augmented model does not hold a stable performance. This shows the potential impact of noise strength for the adversarial training phase. The other point is that the robustness of the multiclass model is higher and in other words the binary classification model itself is more vulnerable. From our empirical experiences this should have something to do with the balance of dataset splitting and output encoding [105].

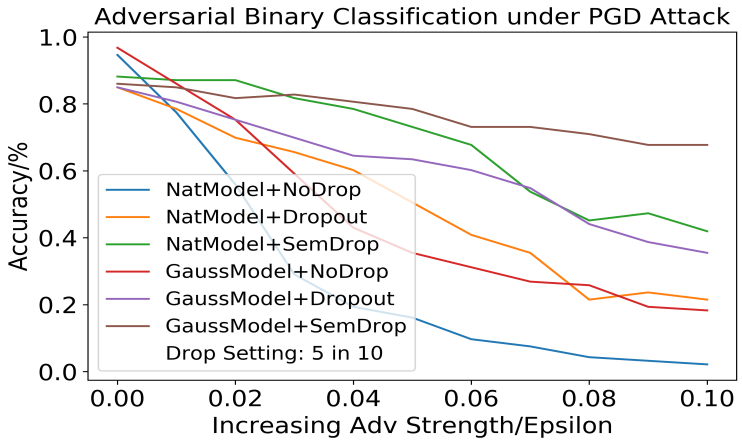

Figure 5.3: Failure Stutus Binary Accuracy under Increasing Adversarial Attack Strength

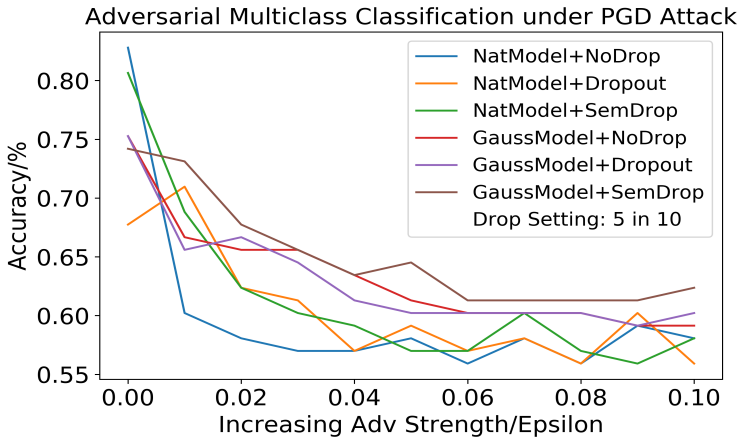

Figure 5.4: Health Status Multiclass Accuracy under Increasing Adversarial Attack Strength

Figure 5.5 shows adversarial regression errors under prediction deviation maximization attack setting. Figure 5.6 shows adversarial regression errors under prediction minimization attack setting. Figure 5.7 shows adversarial regression errors under prediction maximization attack setting. For most of these model and attack settings, the *SemanticDropout* method shows highest adversarial robustness.

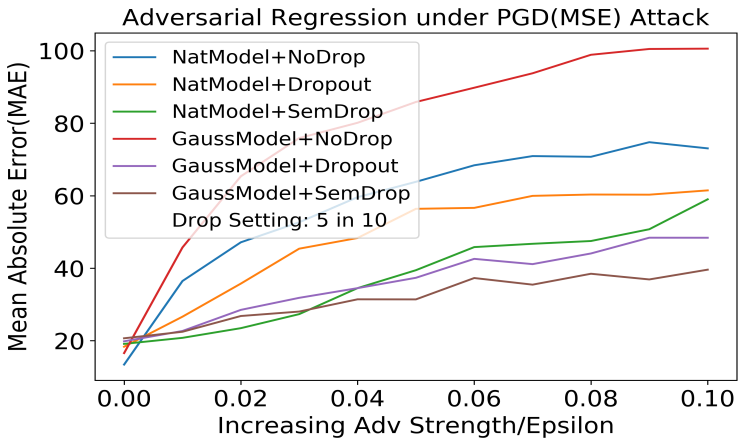

Figure 5.5: Remaining Useful Life Prediction Mean Absolute Error under Deviation Maximization Attack

Figure 5.9-5.11 show prediction results under a medium level($advEps = 0.05$) adversarial PGD attack on the Gaussian Augmented model. Figure 5.9 shows prediction results when the attacker uses the mean square error between clean and adversarial prediction as the optimization loss function and aims to make prediction results on adversarial data that deviates from clean data as much as possible. Figure 5.10 shows prediction results when the attacker uses the negative derivative of the predictor as the optimization loss function and aims to make prediction results on adversarial data as small as possible. Figure 5.11 shows prediction results when the attacker uses the derivative of the predictor as the optimization loss function and aims to make

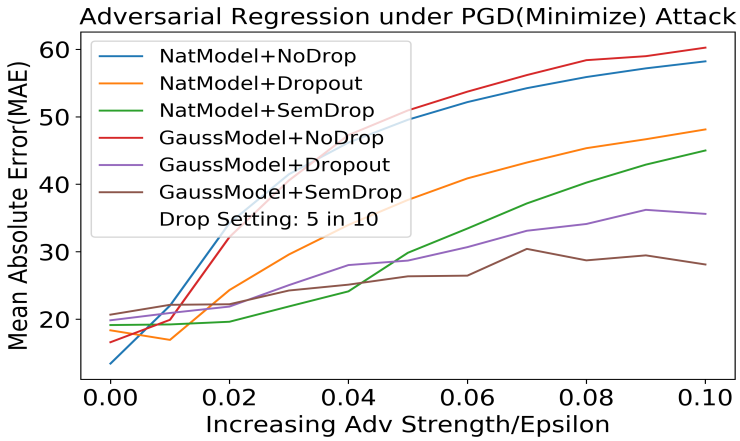

Figure 5.6: Remaining Useful Life Prediction Mean Absolute Error under Prediction Minimization Attack

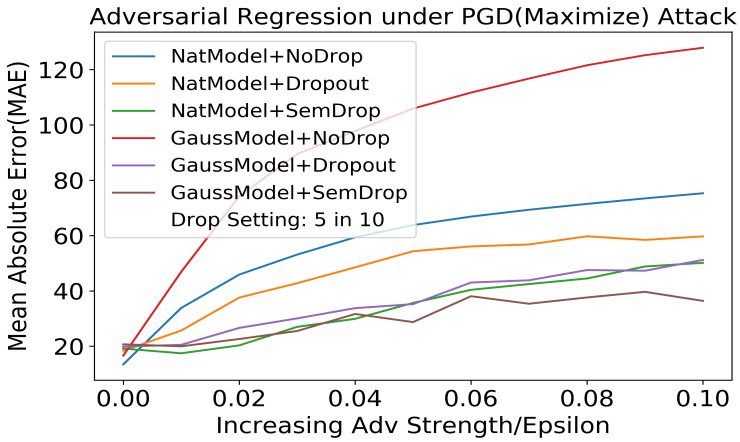

Figure 5.7: Remaining Useful Life Prediction Mean Absolute Error under Prediction Maximization Attack

prediction results on adversarial data as large as possible.

Figure 5.8 We provide an example adversarial perturbation as shown in with a ground truth RUL value of around 60. We can see features get different magnitudes of modification along time steps. And from this example here, we can also see why it is difficult to isolate the origin of vulnerability. Even for a certain feature, it might show different sensitivity levels along time steps. By far, we cannot provide a general description of how a certain sensor reading would be sensitive to adversarial attacks.

To make visualization easier, we sort test data according to their ground truth remaining useful life values. From figures above we can also see that adversarial impacts in this data-driven prognostics case are more prone to making prediction results larger. This is most obvious in Figure 5.9 where adversarial data misguides the predictor to in both directions in a greedy way but we can see the maximization shows up much than minimization. In addition, the proposed *SemanticDropout* method with autoencoder-based re-

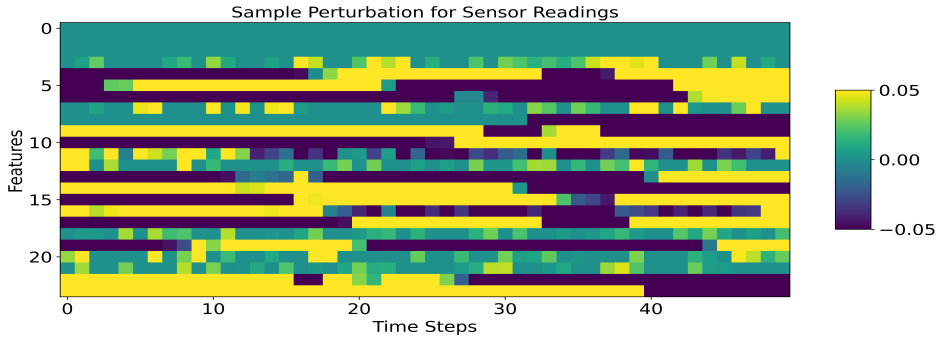

Figure 5.8: Example Adversarial Perturbation under Maximum Magnitude Constraint $advEps = 0.05(5\%)$

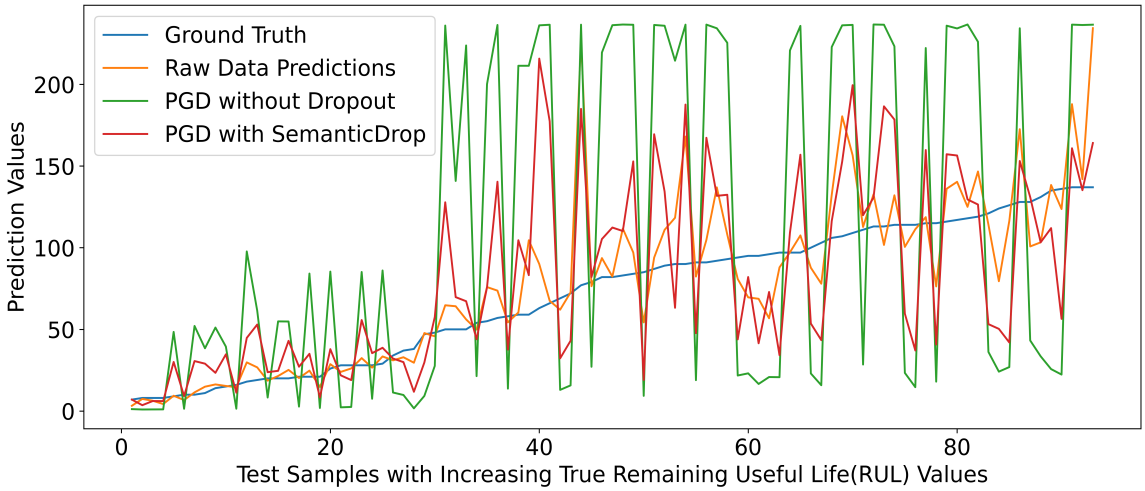

Figure 5.9: Regression Results (with GN) under Deviation Maximization PGD Adversarial Attack with $advEps = 0.05(5\%)$

construction successfully decreases prediction deviations at these vulnerable points without much impact on other locations. On the other hand, these experiments also show the trade-off from potential defense mechanisms. With a large value of detection dropout, the chance of being totally stealthy for an adversarial input data is low. Meanwhile, erasing and reconstructing more elements of data obviously leads to higher reconstruction error and thus prediction error. This becomes an inevitable trade-off by inducing defensive steps since these extra processing would not generate extra information. As shown in our experiments, this potential risk is most obvious when the detection dropout rate is high. As a result, the combination of a medium level detection dropout rate along with reconstruction would render a better trade-off between natural error by the defense mechanism and robustness error by the adversarial attack.

We present experiment results under more flexible settings in Table 5.3. The table shows experimental results under three levels of detection dropout rate: $30\%, 40\%, 50\%$ (three columns from left to right in each setting) with two cycles of reconstruction using dropout and autoencoder. The error metric we choose to show

Settings	AdvLoss	AdvEps	Nat+NoDrp	Nat+SemDrp	GN+NoDrp	GN+SemDrp
Binary Classification Accuracy /%	Binary Cross-Entropy	0	94.6	90.3 / 88.2 / 82.8	**96.8**	88.2 / 86 / 88.2
		0.02	55.9	**86 / 87.1 / 83.9**	75.3	86 / 81.7 / 82.8
		0.04	19.4	80.6 / 78.5 / 74.2	43	**82.8 / 80.6 / 78.5**
		0.06	9.7	63.4 / 67.7 / 60.2	31.2	**74.2 / 73.1 / 73.1**
		0.08	4.3	46.2 / 45.2 / 41.9	25.8	**71 / 71 / 69.9**
		0.1	2.2	38.7 / 41.9 / 37.6	18.3	**66.7 / 67.7 / 65.6**
Multiclass Classification Accuracy /%	Categorical Cross-Entropy	0	**82.8**	75.3 / 80.6 / 77.4	75.3	76.3 / 74.2 / 72
		0.02	58.1	61.3 / 62.4 / 64.5	65.6	**66.7 / 67.7 / 71**
		0.04	57	58.1 / 59.1 / 58.1	63.4	**62.4 / 63.4 / 61.3**
		0.06	55.9	57 / 57 / 57	60.2	**61.3 / 61.3 / 61.3**
		0.08	55.9	57 / 57 / 57	60.2	**60.2 / 61.3 / 60.2**
		0.1	58.1	58.1 / 58.1 / 58.1	59.1	**60.2 / 62.4 / 61.3**
Regression Mean Absolute Error /cycle	Mean Squared Error	0	**13.4**	17.4 / 19.1 / 16.9	16.6	21.1 / 20.7 / 19.7
		0.02	47.1	**25.6 / 23.5 / 24.8**	65.4	27.9 / 26.8 / 26
		0.04	59.6	37.5 / 34.5 / 35.4	80.1	**34.9 / 31.4 / 32.6**
		0.06	68.4	50.6 / 45.8 / 47.5	89.8	**38.1 / 37.3 / 38**
		0.08	70.7	54.3 / 47.5 / 49.3	98.9	**40.9 / 38.5 / 36.6**
		0.1	73.1	66 / 59 / 59.5	100.6	**43.6 / 39.6 / 37.9**
	Minimize	0	**13.4**	17.4 / 19.1 / 16.9	16.6	21.1 / 20.7 / 19.7
		0.02	34.2	**19.6 / 19.6 / 18.8**	32.2	22.8 / 22.2 / 23.4
		0.04	46.2	**26.2 / 24.1 / 25.6**	47.3	25.5 / 25.1 / 26.2
		0.06	52.2	34.7 / 33.4 / 34.3	53.7	**26.4 / 26.5 / 28.5**
		0.08	55.9	41 / 40.2 / 41.2	58.4	**28.4 / 28.7 / 31.1**
		0.1	58.2	45.7 / 45 / 46.1	60.2	**30.2 / 28.1 / 31.6**
	Maximize	0	**13.4**	17.4 / 19.1 / 16.9	16.6	21.1 / 20.7 / 19.7
		0.02	45.9	24.4 / 20.3 / 21.7	74.1	**21.7 / 22.6 / 19.5**
		0.04	59.4	**37.7 / 29.9 / 32.5**	97.8	34.1 / 31.6 / 30.3
		0.06	66.9	48.7 / 40.4 / 42.4	111.7	**38.3 / 38.1 / 33**
		0.08	71.4	55 / 44.5 / 47.3	121.6	**46.5 / 37.6 / 38.5**
		0.1	75.3	61.9 / 50.1 / 51.6	127.9	**45 / 36.4 / 37.7**

Table 5.3: Adversarial Regression Error Rates under Projected Gradient Descent(PGD) Attacks

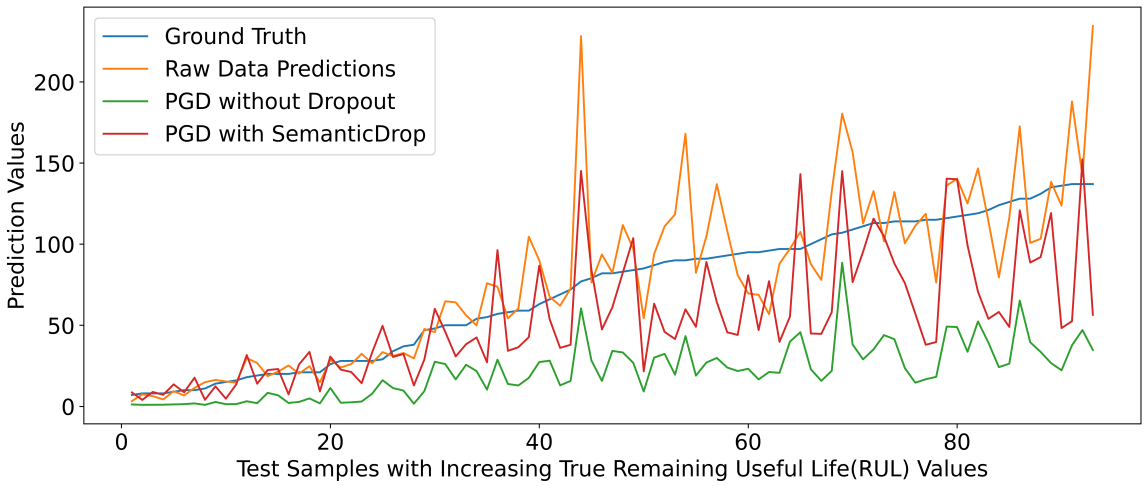

Figure 5.10: Regression Results (with GN) under Prediction Minimization PGD Adversarial Attack with $advEps = 0.05(5\%)$

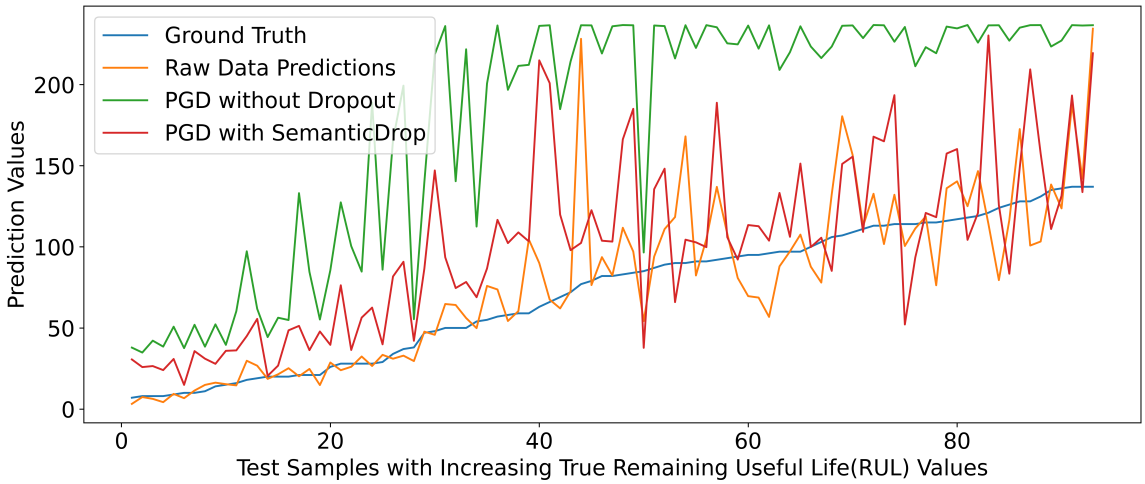

Figure 5.11: Regression Results (with GN) under Prediction Maximization PGD Adversarial Attack with $advEps = 0.05(5\%)$

here is the most commonly used mean absolute error (MAE) for regression and accuracy for classification. For different adversarial attack settings, the settings with best output results are marked in dark black. We can see that low detection dropout rates usually perform better on clean or less perturbed data but on the other hand are less robust against strong adversarial attacks.

In summary, we show the vulnerability and potential risks of deep learning based predictors in data-driven prognostics applications. We also show the efficiency as well as the potential of the proposed randomization-based defense technique involving semantic structural contexts. One significant advantage of our randomization-based framework is that it makes use of existing pre-trained models in a resilient way, which means it can

work together with other defense techniques seamlessly.

5.5 Conclusion

Data-driven models in health management and prognostics scenarios. We explore the vulnerabilities of the state-of-the-art deep neural network data-driven health management and prognostics caused by the technique called adversarial attack. We show the significance of inducing randomization elements to improve model robustness. Furthermore, we investigate the possibility of inducing randomization elements from semantic structural contexts to mitigate adversarial impacts. Our discussions and experimental implementations on the engine degradation dataset cover the most typical settings in PHM applications. These general settings would help formalize the security testing in more scenarios.

CHAPTER 6

Edge Hardware Accelerator Recommendation

6.1 Problem Overview

Hardware accelerator devices have emerged as an alternative to traditional CPUs since they not only help perform computations faster but also consume much less energy than a traditional CPU thereby helping to lower both capex (i.e., procurement costs) and opex (i.e., energy usage). However, since different accelerator technologies can illustrate different traits for different application types that run at the edge, there is a critical need for effective mechanisms that can help developers select the right technology (or a mix of) to use in their context, which is currently lacking. To address this critical need, we propose a recommender system to help users rapidly and cost-effectively select the right hardware accelerator technology for a given compute-intensive task. Our framework comprises the following workflow. First, we collect realistic execution traces of computations on real, single hardware accelerator devices. Second, we utilize these traces to deduce the achievable latencies and amortized costs for device deployments across the cloud-edge spectrum, which in turn provides guidance in selecting the right hardware.

An issue that is often encountered by developers planning to deploy their applications, particulary those involving data-driven machine learning elements, is what device hardware is best suited (performance and cost-wise) for a given task under varying data processing demands. This question is even more pronounced [45] for resource-constrained edge computing/IoT.

This problem is particularly acute in scenarios where cameras with video streams are involved [47] because compute intensive tasks, especially deep learning based machine learning applications on these data, often require millions of operations for each inference [48]. Thus, to maintain low latencies and compliance with the power sensitivity of edge deployment, smaller models [49] operating on more efficient hardware are preferred [50]. To that end, hardware acceleration technologies, such as field programmable gate arrays (FPGAs), graphical processing units (GPUs) and application-specific integrated circuits (ASICs) among others, have shown significant promise for edge computing [51].

With the proliferation of different accelerator technologies, developers are faced with a significant dilemma: which accelerator technology is best suited for their application needs such that response times are met under different workload variations, the overall cost of the deployment fits their budget and the energy consumption for these devices are below a threshold. This dilemma stems from the fact that different accelerator technologies illustrate different performance and energy consumption traits for different application scenarios. Contemporary techniques that evaluate acceleration technologies are either too specific to a device type and

incorporate only a single inference instance level [106; 107] or comprise low-level circuit design optimization analysis [108; 109]. Consequently, there remains a significant lack of understanding on the applicability of these accelerator technologies in at-scale, edge-based applications [110] due to diversity of systems settings [111].

In a general sense, selecting a suitable hardware accelerator technology for a given computational task can be regarded as finding out a hardware device placement and optimization strategy for a given topology. Prior work has explored this general problem for service placement for edge computing [52; 53]. From a hardware perspective, workload-specific accelerator placement frameworks for automobiles and unmanned aircraft have been proposed [54; 55; 56]. In these workloads, energy usage is dominated by vehicles rather than computational overhead. From a higher cloud-level point of view, research on device and service placement also exists [57; 58]. Yet, widespread adoption of these different hardware platforms requires more systematic understanding of both spatial and temporal impacts in realistic deployments in a network with compute-intensive tasks.

To address these challenges, we present *HARE (Hardware Accelerator Recommender for the Edge)*, which is a framework to help users rapidly and cost-effectively select the right hardware accelerator for a given compute-intensive task. Our work focuses on a general framework to model compute latency and energy usage across accelerator devices, and makes the following contributions:

- We present a novel hardware-software co-evaluation framework that formalizes the performance comparison involving training samples obtained on real hardware.
- We develop a simple approximation method that can be utilized to implement long-term cost analysis for hardware-based machine learning inference behavior at the edge.
- We show case studies involving two realistic machine learning application settings and demonstrate HARE in these contexts.

The remainder of the paper is organized as follows: Section 6.2 describes the design of the HARE hardware accelerator evaluation framework; Section 6.3 provides empirical experiment design and evaluation results of applications across different hardware platforms; and Section 6.4 concludes the paper.

6.2 HARE Framework Design

We now present technical details about HARE. In realizing it, we faced several challenges. The broad range of hardware options and settings makes performance quantification on the different device types very difficult. There are numerous hardware-related metrics and we had to decide which metrics were most significant and how to depict hardware acceleration patterns in a compact manner. Finally, for at-scale deployment of

devices, the overall system performance is not just a simple addition of individual performance cases.

Our approach to realizing HARE uses the following steps: (1) Conduct performance and energy benchmarking in isolation per device type, and (2) Use these insights to approximate the performance, energy and cost metrics for at-scale deployments, which is then used as the recommendation engine.

Table 6.1: Device-level Acceleration Deployment Workflows for Different Hardware Platforms

Design Flow	**Hardware**	**ResNet-50 Object Classification**	**Tiny Yolo Object Detection**
Edge CPU	Raspberry Pi 3 b+	Tensorflow/Keras	Darknet
Embedded GPU	NVIDIA Jetson Nano	TensorRT	Darknet/TensorRT
FPGA	Avnet Ultra96	DNNDK	DNNDK
ASIC	Intel NCS	OpenVINO	OpenVINO
Server GPU	NVIDIA GTX1060 6GB	Tensorflow/Keras/CUDA	Tensorflow/Keras/CUDA
Server CPU	AMD FX-6300	Tensorflow/Keras	Tensorflow/Keras

6.2.1 Benchmarking in Isolation

For our first step, we have used machine learning-based applications to drive our benchmarking efforts. This choice stems from the fact that state-of-art deep learning advances have enabled higher demands for compute intensive data stream processing tasks from edge sources, especially video and image processing using deep learning [47]. We chose application cases belonging to classification (*ResNet-50* [112]) and detection (*Tiny Yolo* [113]) as test functions to explore and compare their performance across different hardware platforms. Note that machine learning comprises many more techniques and deployment on different hardware, such as random forests [114] or decision trees [115]. However, there is a general lack of comparison standards due to heterogeneous platform workflows and limited deployment scale. Further, although it is desirable for the same model to illustrate similar accuracy across different hardware [107] as is illustrated by TensorFlow [116], the performance across these hardware may be different.

To that end we consider four types of hardware including CPU, GPU, ASIC and FPGA. CPUs are the most general hardware platform. GPUs, particulary the NVIDIA products with CUDNN [117] support both desktop and embedded training and inference. For ASICs, we used two currently available neural network acceleration chips on the market: the Intel Neural Compute Stick (NCS) [118] and Google Coral with Tensor Processing Unit (TPU) [109]. For FPGAs, High-level synthesis (HLS) is a technology that can translate behavioral-level design descriptions using C/C++ into RTL descriptions [119] and make them easier for application developers to test edge applications [120]. For deep learning tasks, Xilinx provides a toolkit called DNNDK [121] based on HLS that is aimed at providing users with an end-to-end workflow to deploy a trained network on FPGA devices. The design flows for different hardware platforms that we have used are summarized in Table 6.1.

Single device executions aim at collecting time and power consumption data for the task under consideration. Time experiment records are series of data points of execution time length for each inference. From these data records, the mean and standard deviation of response time and its inverse (inference frequency) are determined to document the capability of this device. That is, we use a normal distribution conforming to $N(muT_{\text{dev}}, stdT^2_{\text{dev}})$ to approximate time consumption per inference for a device and $N(muFreq_{\text{dev}}, stdFreq^2_{\text{dev}})$ to denote the inference frequency. We use random variables rather than static values of average or maximum to allow uncertainty quantification for overall system performance. As the computation system is assumed to be active throughout the deployment cycle, for power consumption data both static and dynamic inference consumption is recorded.

6.2.2 Inferring the Desired Metrics At-Scale

The aim of this step is to infer the performance, energy and cost metrics for large-scale deployments of ML applications that span the cloud-to-edge spectrum using insights from benchmarking of isolated use cases and solving an optimization problem.

Our method for inferring the desired metric inference can be thought of as an optimistic upper bound approximation for the hardware device selection for application deployment across the cloud and edge without considering the underlying model-related error metrics like classification accuracy. Under a linear speedup assumption with additional device parallelism, further speed up with multiple devices would increase total inference capability without increasing uncertainty in performance (variance) [122]. To ensure nHW_{dev} that a specific type of device with total inference capability $R_{\text{dev}} \sim N(muFreq_{\text{dev}} * nHW_{\text{dev}}, stdFreq^2_{\text{dev}})$ can handle input data load $L_{\text{dev}} \sim N(muFreq_{\text{in}}, stdFreq^2_{\text{in}})$, there should be enough devices deployed for the given input pressure with a design confidence level $conf$:

$$Pr(R_{\text{dev}} - L_{\text{dev}}) > conf \tag{6.1}$$

For latency constraint, we have the latency as the sum of hardware running time and communication time in the inference loop. The average latency can be compared in a straightforward way:

$$T_{\text{app}}(dev) = T_{hw} + T_{comm} = muT_{dev} + t_{e2f} + t_{f2c} \tag{6.2}$$

where $t_{e2f} = S_{e2f}/B_{e2f}$ refers to the communication time from edge to fog (which is a layer between the edge and cloud) and $t_{f2c} = S_{fec}/B_{f2c}$ refers to the communication time from edge to cloud. The communication time is computed using the transfer package size S divided by the bandwidth B, where bandwidths are

assumed to be stable. For different application scenarios, communication bandwidths vary [123].

We set the unit cost of electricity $costElec$ as a constant $\$0.1/kwh$ [124]. Moreover, the total electricity cost can be denoted as:

$$costP(dev, T_{cycle}) = P_{\text{app}}(dev, T_{cycle}) * costElec \tag{6.3}$$

Likewise, the total power consumption for a given deployment cycle length T_{cycle} can be computed by the sum of idle power and working power:

$$P_{\text{app}}(dev, T_{\text{cycle}}) = P_{\text{idle}}(dev) * T_{\text{cycle}} + P_{\text{perinf}}(dev) * muFreq_{\text{in}} * T_{\text{cycle}} \tag{6.4}$$

Based on the definitions given above, we formulate the problem of hardware accelerator selection as minimizing the total deployment cost while lowering time and power consumption to a lower target level. We consider performance requirements at the application level using hardware commercial cost, latency T_{app} for each inference loop and total working power consumption $P_{\text{app}}(dev, T_{cycle})$ over the design deployment cycle T_{cycle}.

$$\begin{aligned} &\min_{dev \in \boldsymbol{ListHW}} \sum costHW_{\text{dev}} * nHW_{\text{dev}} + costP(dev, T_{cycle}) \\ &\text{subject to:} \\ &\qquad T_{\text{app}}(dev) \leqslant t_{\text{target}} \end{aligned} \tag{6.5}$$

6.3 Evaluation

We now validate the effectiveness of the HARE framework. Based on the functional descriptions and hardware design workflows discussed above, in this part we describe experiments involving hardware deployment and other metric evaluations on test applications.

6.3.1 Testbed Configuration

The testbed and commercial price ($costHW_{\text{dev}}$) of deployment platforms at the time we conducted the experiments are presented below.

1. **CPU:** CPU is the most general option. Two options from both server and edge side are considered in our test framework.
    1. Raspberry Pi 3B ($40): Edge deployment with a Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
    2. AMD FX-6300 CPU ($70): Server deployment with 6 cores, 6 threads @ 3.5-3.8GHz
3. **GPU:** For GPU, two options from both server and edge side are considered in our test framework.

1. Jetson Nano ($99): Edge deployment with embedded GPU 128-core Maxwell and CPU Quad-core ARM A57 @ 1.43 GHz
2. GTX 1060 6Gb ($180): Server deployment with 1280-core Pascal architecture

3. **ASIC:** ASIC stands for application specific integrated circuits. We choose Neural Compute Stick(NCS) from Intel ($75). It is equipped with Movidius Myriad 2 Vision Processing Unit (VPU) with nominal 600 MHz operations at 0.9V.
4. **FPGA:** Machine learning inference tasks require both general-purpose computations and application-specific computations. We choose the relative high-end Ultra96 ($250) Zynq Ultrascale+ Development board (www.zedboard.org).

As a result, the list of potential hardware in our problem setting can be shown as:

$$\boldsymbol{ListHW} = \{RPi, FX6300, JetsonNano, GTX1060, NCS, Ultra96\} \tag{6.6}$$

In this set of devices, the Raspberry Pi (*RPi*) could be regarded as the baseline for acceleration comparisons due to its lowest performance (being a CPU). Some devices need to work with a host. NCS needs to be plugged into a USB port. And GTX1060 needs to be connected to a PCIE port with external power support.

6.3.2 Per-Device Benchmarking Results

Experiments on classification tasks are conducted on a set of 500 images with a resolution of $640 * 480$. Experiments on detection tasks are conducted on a road traffic video consisting of 874 frames with a resolution of $1280 * 720$. We primarily gather power and time consumption on these high-dimensional data compatible with ImageNet[48] to guarantee higher generalization than cifar-10[125] or other lower-dimensional datasets. The power consumption data is gathered using a specific power measurement instrument called Watts Up Pro. For time consumption, it is worth pointing out that the time metric we are using here is the total response time running on hardware for inference tasks. As a result, $T_{\text{hw}} = T_{\text{preprocess}} + T_{\text{inference}}$. That is, the total time consumption includes both the input data preprocessing time and the inference time. Tables 6.2 and 6.4 show response times for each Resnet50 and Tiny Yolo inference, respectively, while Tables 6.3 and 6.5 show both their idle and execution power consumptions.

Table 6.2: Response Time (T_{hw}) for Object classification Task using *ResNet-50* (Unit: Second)

Time	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
mean	2.089	0.133	0.029	0.218	0.039	0.268
std	0.058	0.016	0.001	0.003	0.005	0.006

Table 6.3: Power Consumption for Object classification using *ResNet-50* (Unit: Watt)

Power	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
Idle	1.8	2.2	6.2	0.4	10	72
Infer	4.8	5.6	7.6	1.9	122	145

Table 6.4: Response Time (T_{hw}) for Traffic Detection Task using *Tiny Yolo* (Unit: Second)

Time	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
mean	2.874	0.096	0.023	0.238	0.059	0.217
std	0.068	0.008	0.001	0.003	0.002	0.076

Table 6.5: Power Consumption for Traffic Detection using *Tiny Yolo* (Unit: Watt)

Power	RPi	JetsonNano	Ultra96	NCS	GTX1060	FX6300
Idle	1.8	2.3	7.4	0.4	10	72
Infer	4.8	11.7	9.2	2.1	122	150

6.3.3 At-Scale Approximation Results

We make evaluations for time, power and cost for at-scale deployments using our approximation approach.

6.3.3.1 Application Topology

We present a prototypical smart application case comprising distributed camera networks that can be generalized to scenarios like unmanned shopping using object classification and surveillance using detection as shown in Figure 6.1. Computation burdens are conducted on edge nodes. There would be different design standards for various scenarios. Without loss of generality, we set the system deployment goal as computation resources should guarantee to handle no less than half (2 of 4) of input loads from every fog group (3 groups) with an overall confidence level of 99%. We set the input with relatively high uncertainty when $stdFreq_{\text{in}} = muFreq_{\text{in}}$.

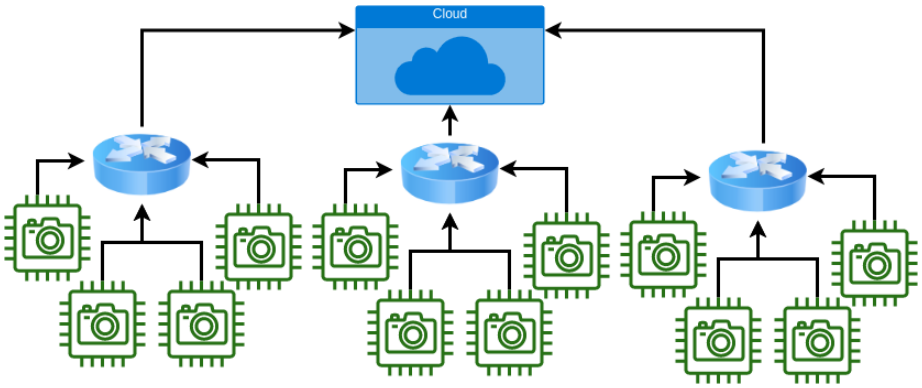

Figure 6.1: Three-level Design Topology Layout: (1) Top:Cloud servers; (2) Intermediate:3 Fog groups include communication control and some computation power; (3) Bottom:4 Edge nodes in each fog group closest to sensors and data needs to be processed.

6.3.3.2 Latency Estimation

A straightforward comparison for response times per inference can be conducted using inference time data from Tables 6.2 and 6.4. From the time point of view, Rapsberry Pi 3 b+ consumes the most for each inference task. Data for bandwidth between edge/fog and cloud is retrieved from standard IEEE802.11n Wifi setting [123] as the optimal upper bound of $135Mbps$. The size of control signal is set as $1kb$ and data signal using JPEG [126] 100% with $24bit/pixel$. Based on the definitions from Section 6.2, we show latency estimations of $T_{\text{app}}(dev)$ in Table 6.6.

Table 6.6: Inference Cycle Time $T_{\text{app}}(dev)$(Unit:Second)

Time	RPi	JetNano	Ultra96	NCS	GTX1060	FX6300
Loc	Edge	Edge	Edge	Edge	Cloud	Cloud
Res	2.088	0.131	0.029	0.218	0.046	0.275
Yolo	2.869	0.096	0.023	0.238	0.081	0.190

6.3.3.3 Power Consumption

Based on the method discussed, we can approximate the total power consumption holding application accelerations for a given cycle length. We use the idle power of one device to denote the total idle power of this type of devices in a network based on an ideal time-division setting. We set the deployment time length as 24 months for our simulated at-scale setup and the total power consumption for both classification and detection tasks is computed for this period as shown in Table 6.7.

Table 6.7: Total Working Power Consumption (Unit:kWh)

Power	RPi	JetNano	Ultra96	NCS	GTX1060	FX6300
Res	3720	372.6	87.2	144.8	2243	14236
Yolo	5040	485.7	87.1	173.9	2660	27685

6.3.3.4 Cost Evaluation

Figures 6.2 and 6.3 show results for devices with *ResNet-50* classification and *Tiny Yolo* detection applications, respectively. Under increasing input pressure, the number of devices required steps up like stairs. For a given design cycle, the total cost consisting of device and power cost would be proportional to the number of devices.

Traditional CPUs like Raspberry Pi on the edge and FX6300 on the cloud both show worst performances. This explains why hardware accelerators are necessary for such use cases involving compute-intensive tasks like ML inference. For long-term deployments, power consumption would play a significant role in the total cost. For classification and detection tasks, the Ultra96 (FPGA) and JetsonNano (embedded GPU)

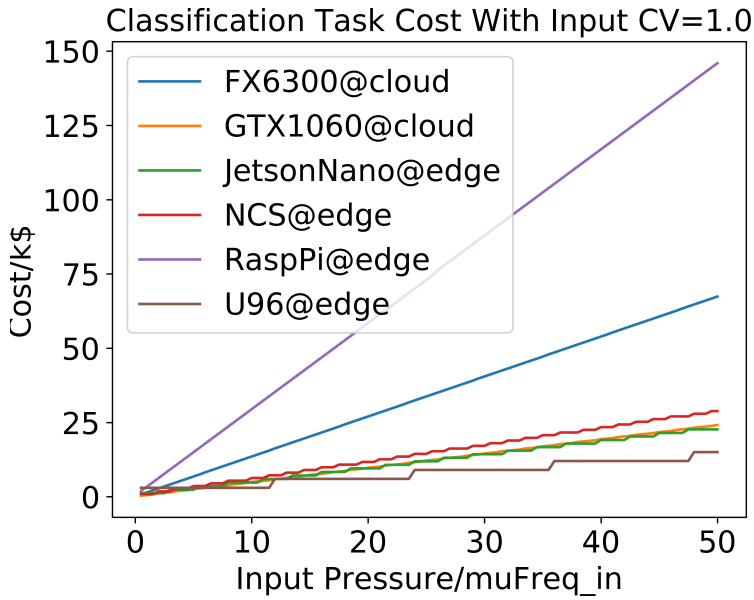

Figure 6.2: *ResNet-50* Classification Cost (Unit:*k*$) for a 24 Month Deployment Cycle Under Increasing Data Input Pressure (Frequency) and Medium Complexity($muFreq_{in} = stdFreq_{in}$)

illustrate the two most cost-efficient options for the edge side at high input pressures. NCS (ASIC) could be easily deployed to existing network topology but relatively lower individual device performance becomes its limitation. Edge accelerators show more obvious advantage for video detection case; this indicates the necessity for offloading streaming data processing in the edge computing pattern.

6.4 Conclusions

This paper presents a simple recommendation system to help users select an accelerator hardware device for their applications deployed across the cloud to edge spectrum. Our approach first measures realistic execution traces of computations on individual hardware accelerator devices. We then use these data to approximate latency, power consumption and long-term cost for at-scale deployments, which provides guidance on device selection.

Traditional CPUs like Raspberry Pi on the edge and FX6300 on the cloud both show worst performances. This explains why hardware accelerators are necessary for such use cases involving compute intensive tasks like machine learning inference. For long-term deployments, power consumption would play a significant role in the total cost. For the hardware devices used, FPGAs show both highest absolute computation power and highest energy efficiency. This makes it a good option for flexible acceleration tasks at the edge. However, for relatively low pressure scenarios, the device costs could be dominant and FPGAs are harder to program. Furthermore, the number of devices needed is based only on an ideal assumption and a large number of con-

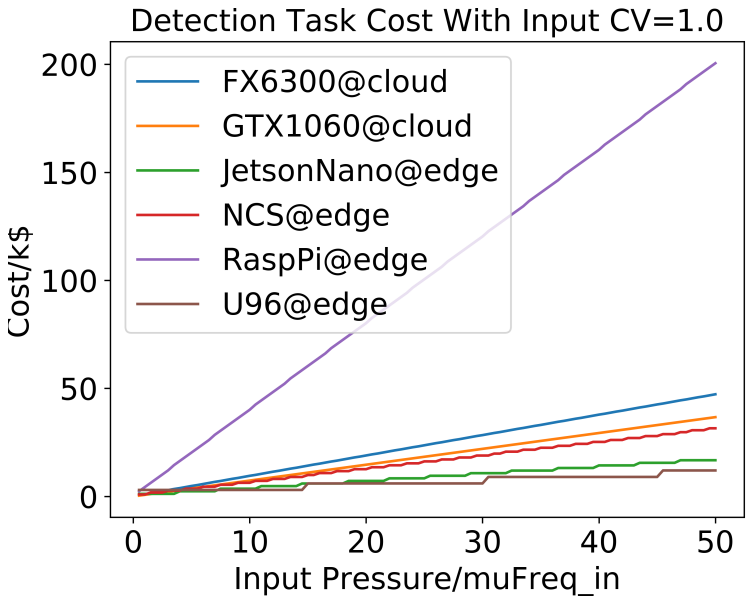

Figure 6.3: *Tiny Yolo* Detection Cost (Unit:*k*$) for a 24 Month Deployment Cycle Under Increasing Data Input Pressure (Frequency) and Medium Complexity($muFreq_{in} = stdFreq_{in}$)

nected nodes might still lead to a larger number of devices needed and higher usage threshold for seeking the desired parallelism. For machine learning inference tasks, server-side GPUs can have the most computation power and higher energy efficiency than CPUs and some lightweight edge devices. Thus, embedded GPUs like Jetson Nano could have much potential for more general computation tasks, particularly those related to graphics and vision. ASIC devices like NCS do not show the highest time efficiency or power efficiency but considering its low cost and low threshold to use, it is still competitive for neural network-specific and relatively low input frequency tasks.

CHAPTER 7

Cloud Assisted TinyML for PHM Scenarios

This chapter extends our view from Chapter 6 to the novel edge LEC deployment hardware option of TinyML on microcontrollers(MCUs). We study the potential of TinyML in data-driven prognostics as a representative of CPS. and emphasize the necessity of compensating its limited hardware resources with more mature cloud computing techniques. Our work proposes several application patterns of TinyML-based CPS applications especially PHM applications where incorporating cloud computing could make a huge difference. We show that cloud computing could help in TinyML model training&deployment, data storage&integration and model inference computation&power offloading.

7.1 Problem Overview

With the current development of machine learning, especially deep learning, statistical prediction models like neural networks have achieved state-of-the-art performance for many tasks pertaining to cyber-physical systems including prognostics and health management (PHM). Recent hardware advances have made it possible to run small but optimized machine learning models on low-power microcontrollers (MCUs), such as Arduino or STM32, at the edge. By being closer to the source of the data, PHM applications can be more efficient. This concept is called embedded machine learning, or TinyML [127]. By providing on-device sensor data monitoring and processing, TinyML provides promising opportunities for PHM applications in early warning generation and smarter decision making at low-power as well as low cost with a low-latency guarantee.

Executing edge-based machine learning models for PHM applications has been studied in the literature [128]. Past works usually use MCUs for condition monitoring and set relative simple threshold strategies for early warnings [129]. These past works show the potential of MCU-based smart decision making for PHM applications [130]. At the same time, they also show one general difficulty of using MCUs for data-driven PHM because the prediction models are usually case-specific and lack general applicability. Moreover, the limited hardware capacity of the MCUs and high data volumes of PHM applicatons makes it hard to rely on MCUs alone for complete PHM capabilities.

Furthermore, efficiently assigning prediction or inference tasks to TinyML devices for PHM applications is difficult due to the following challenges:

- PHM application data sources are often from different sensors. It is difficult to fuse sensors when we use them as data inputs for TinyML models.
- PHM applications often involve systems with manual configurations. We need to consider how these

configurations would be used as input features of TinyML models in a proper way.

- PHM applications often make use of prediction models with spatio-temporal considerations. Time synchronization and sensor data transfer would become a system-level challenge for sensor networks with limited hardware resources at the sensor end.
- Recent PHM applications involve higher data volumes at the sensor end e.g., live video or audio data. Efficient model training and optimization for potential TinyML deployments has become a challenge.

To address these challenges, we propose the use of cloud computing as a high-level support system that can work in conjunction with TinyML deployments. In this chapter, we explore potential application patterns where MCUs and cloud computing can be adapted to different kinds of machine learning models and combined in flexible ways thereby catering to diverse PHM application requirements. Overall our work emphasizes the following critical aspects of TinyML-based PHM applications where cloud computing techniques could contribute to:

- Cloud-assisted model training and code generation for heterogeneous hardware devices.
- Data storage and integration for heterogeneous sensor data sources.
- Model inference computation offloading for latency, power and communication optimization.

The remainder of the chapter is organized as follows: Section 7.2 describes the potential techniques for more efficient cooperation between cloud and TinyML models in PHM applications; Section 7.3 provides empirical experiment design and evaluation results of applications on hardware platforms; and Section 7.4 concludes the paper. Code available at: https://github.com/dustinjoe/TinyML-with-Cloud-for-PHM.

7.2 Methodology

In this section we delve into the details of the underlying methods used in our work. We first provide a general introduction to the current status of TinyML in Section 7.2.1. Then we propose three methods covering aspects of cloud computing that would help more efficient TinyML deployments in PHM applications in Section 7.2.2- 7.2.4.

7.2.1 TinyML: Machine Learning Inferencing on Tiny Microcontrollers

Microcontrollers (MCUs) are low-power computation devices with limited hardware resources. They are widely used in almost all cyber-physical systems including for PHM application. A typical microcontroller would have less than 500kb of storage and less than 100kb of RAM. In order to run a machine learning model on such a small-scale device, we need a light-weight model execution engine for the MCU as well as an optimized model representation. Even though in theory it is possible to also have model training updates on MCUs [131], the limited amount of computation power on MCUs can often make the model

learning/updating process on MCUs infeasible. As a result, from a practical point of view, in this work, we focus on conducting machine learning inference rather than learning on MCU devices.

The development of TinyML enables a systematic application of relatively complex prediction models on MCUs [132]. For example, the work in [133] describes State of Health (SoH) for battery health monitoring and presents a comparison of different ML algorithms for estimating maximum releasable capacity of Li-Ion batteries. These efforts formulate the current tiny machine learning stack as consisting of sensor fusion, data collection, model training, model optimization, hardware-targeted model conversion and realistic hardware deployment [134]. Moreover, although there exists substantial heterogeneity in both software and hardware for different levels of model training and deployment, the machine learning model training are all based on general frameworks like Tensorflow/Keras and the deployment of these models on MCUs often involves a hardware-targeted model conversion and optimization procedure [135; 136].

7.2.2 Cloud-Assisted Model Training/Deployment

Recently, there has been a growing trend towards minimizing manual code development [32] but rather have users utilize simple logic or drag-and-drop functional blocks to realize the desired features. To design a cyber-physical system with smarter decision making, a development platform purporting to minimizing coding efforts should be able to combine IoT sensing endpoints with hardware-targeted embedded machine learning model deployment. Furthermore, this platform should be able to automatically generate an optimized model that provides optimal on-device inference performance on custom hardware platforms. PHM applications are typical cyber-physical system applications that would involve a holistic workflow that requires both data collection and data processing. As a result, we can expect more end-to-end code generation and deployment workflows for PHM applications involving embedded machine learning.

With more sensors connected in PHM application, an integrated cloud platform would be critical for data integration as well as infrastructure management [137]. There have been some pioneering end-to-end solutions that have shown the prototype of these potential scenarios. Two examples are SensiML [138] and Edge Impulse [139]. They both have integrated TinyML development workflows to enhance the model training and deployment on MCU or other edge devices. This kind of public cloud service helps lower the technical barrier to realize the workflows for edge model applications. It incorporates functions of data acquisition, processing, model training&testing and actual device deployment. It also generates C code for various type of devices including MCUs.

Time/Sensor	Sensor_1	.....	Sensor_n
Step_1	data(1,1)	...	data(1,n)
.....	...		...
Step_T	data(T,1)	...	data(T,n)

Figure 7.1: Sensor Data Time Series.

7.2.3 Integration with Database Storage

For health status diagnosis and prognosis, we must often deal with continuous incoming sensor data. As a result, the model would absorb time series data from sensors and use the data format for predictions as shown in Fig 7.1. In addition, MCUs can operate as gateways to enable protocol translation and edge-to-cloud internet connection. To cope with tremendous amount of heterogeneous types of data, different cloud database systems have been exploited [140]. To seek high performance in tasks like data transformation, analytics computations and data visualization, etc., specific database systems have been actively under development [141]. With respect to the edge-to-cloud PHM continuum presented in this paper, databases that provide effective support for interaction with ML-based data mining techniques are applicable to our work. One example includes the INFN-CNAF computing center [103] used to launch large-scale data mining tasks based on InfluxDB [142] towards a global predictive maintenance solution. Apart from time series data, we also must deal with non-structural data like images, audio or even video. This adds to the necessity of using cloud as an integrated storage solution.

7.2.4 Computation Offloading

Even though TinyML development has enabled the execution of light-weight prediction models on MCUs, the limited amount of hardware resources on the MCU side cannot handle larger volumes of data from heterogeneous data sources. To solve this issue, we need to combine the cloud servers with MCU devices to formulate an edge-cloud computing paradigm for more adaptive computation burden offloading [45].

One significant concern of features from sensor data in PHM applications is that the input data dimensions have semantic structural relationships. The main motivation for data-driven prognostics is that in practice we may not be able to get accurate physics-based models to conduct reliable model-based prognostics. But how to fuse information from distributed data sources for system prognostics in a hybrid way is still a practical problem. By offering more flexible data manipulation, we argue that the combination of TinyML and cloud computing can play an important role in PHM applications.

As we have mentioned, in practice we must deal with various kinds of data for PHM applications. This leads to challenges from two inevitable aspects when we combine it with the high-level cloud computing when we are seeking a system-level performance evaluation. One is the presence of large volumes of data communication between the MCU edge nodes and the cloud server. The other is the interaction between the TinyML model and the cloud-based ML model. These two aspects may manifest in various forms in practical applications but we propose a more general framework to solve these issues.

Based on the temporal dependency of the predictor input data, we classify ML-driven PHM applications into three types:

- No Temporal: e.g, comprising discrete data input like camera images.
- Continuous: e.g., those comprising continuous streams from all sensor nodes. Only new coming data is updated for storage and prediction.
- Discontinuous: e.g., those that do not need monitoring at all times. All sensor data in the past window length of time, however, needs to be fetched.

For the first setting without temporal dependencies, the TinyML predictor serves as an early warning as well as a data selector. It uses light-weight models to judge whether it is necessary to send the current input to the cloud for further analysis. If it is not necessary then the communication bandwidth for data transmission would be saved. This working mode would be meaningful for data with relatively large granularity.

Both continuous and discontinuous modes put emphasis on time series data, but the continuous mode conducts data analysis continuously and while the latter conducts analysis periodically or based on other monitoring policies. Irrespective, the general computation offloading guideline is that TinyML can only conduct light-weight predictions using nearby sensor data, must interact with the cloud and let the cloud server conduct the more systematic analysis. The discontinuous mode, however, needs to send more data to the cloud server for one inference cycle.

We propose a resilient TinyML model application strategy to solve this issue. We dive into the multi-layer structure of neural network models. For the working mode with no temporal or continuous dependency, the data uploaded remains unchanged. But for the discontinuous mode, the data uploaded to the cloud would be the latent space representation of one intermediate layer of neural network model. With a designed network structure, the latent space representation can have much fewer dimensions than the original input. In this way, the first part of the TinyML prediction model serves as an input decoder that helps compress the transmitted data. We show an example in Fig 7.2. The input is first sent into an encoder and encoded as a latent representation. Then this latent representation is further utilized for predictions. This ideology could be utilized for both supervised and unsupervised learning models. For example, using an identity mapping auto-encoder is

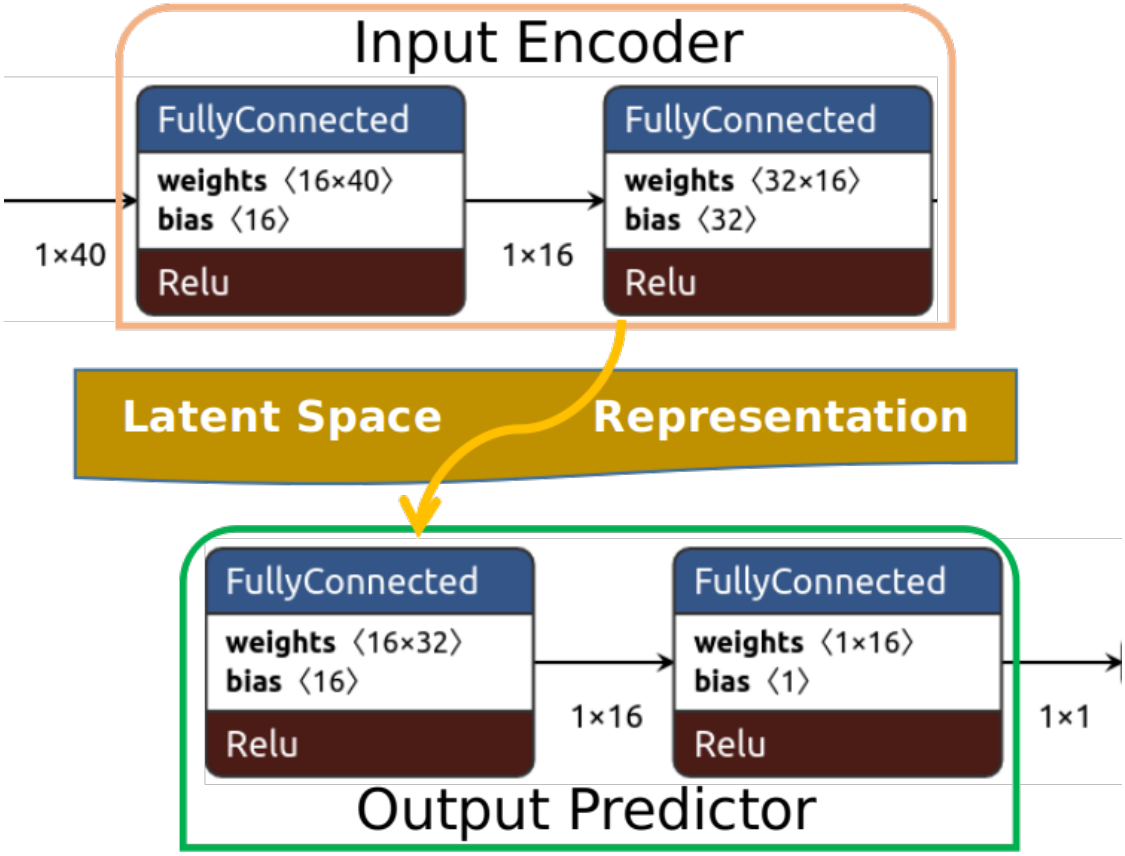

Figure 7.2: Resilient Application of TinyML Model.

a classical unsupervised method for sparser data representations and also anomaly detection [136].

From the semantic structure point of view, TinyML models are focused on data from one sensor or a small group of sensors nearby. In this way, one model on MCU copes with data from one sub-component of the system. To get the overall system status prediction, we need to find a way to integrate results from sub-components together. In machine learning this can be defined as an ensemble as shown in Fig 7.5.

The strategy for an ensemble is flexible under different application settings. For the mode without temporal dependencies or continuous mode, this can be conducted as the process of fusing results from different sensors. This may involve the fusion of data as well as prediction results. For discontinuous mode, this could be the concatenation of compact latent space representations and then utilized together for more systematic analysis. This can be realized in a hierarchical manner. Light-weight models for sub-components are trained first and frozen. Then these pre-trained models can be combined together. This means the 'Ensemble' operation shown in Fig 7.5 can either be a relative simple operation like concatenation or some extra neural network layers going forward towards a final systematic result.

In summary, we discuss potential critical roles that TinyML can play together with cloud computing in

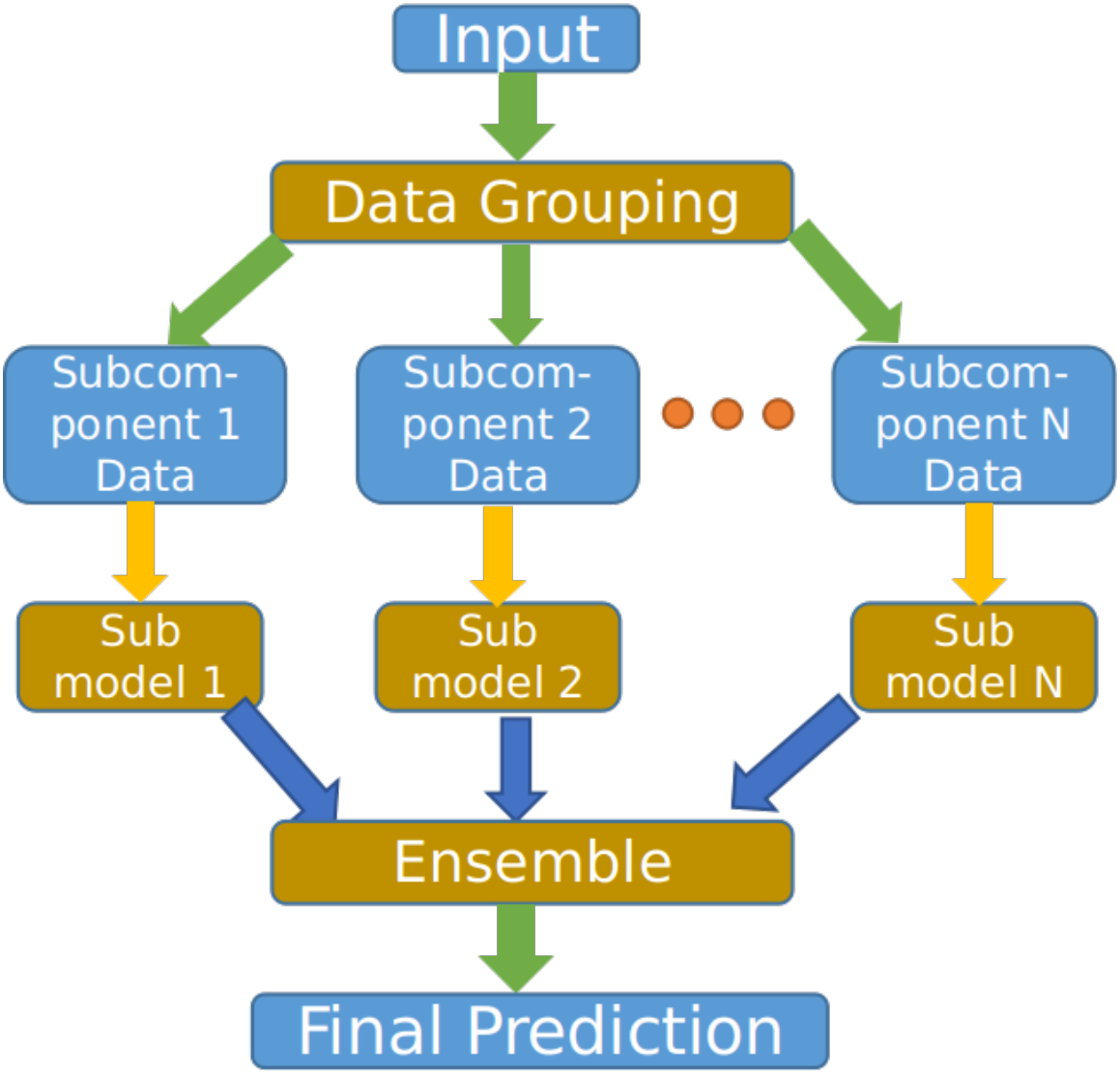

Figure 7.3: Ensemble System-level Modeling of TinyML Sub-models.

data-driven PHM applications. We emphasize three critical functions of early predictive warning, dynamic offloading and data compression that TinyML can realize inc conjunction with a cloud setting.

7.3 Evaluation

For demonstration purposes, we conduct two case studies to showcase the role of TinyML under different PHM working modes. The first study is about surface crack detection using camera images. The MCU takes pictures using on-board cameras and use a TinyML model to detect whether there is crack in the image. If one is detected, then it sends the image to cloud for further analysis. The second case study uses the widely-used C-MAPSS jet engine degradation dataset [88]. We build light-weight models for sub-components of the

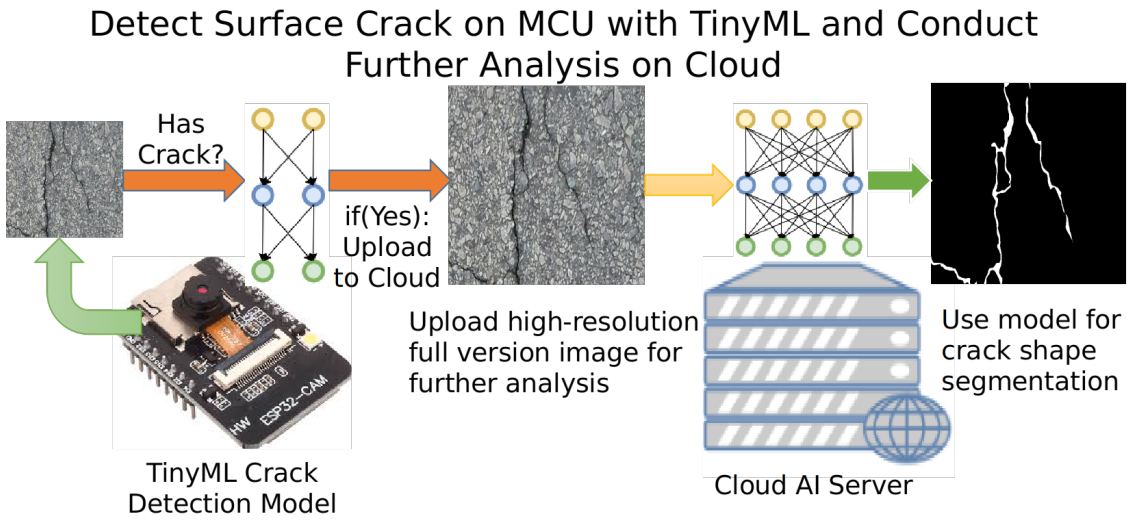

Figure 7.4: Crack analysis workflow of running binary crack detection on MCU using relative low-resolution images and further segmentation analysis on cloud using high-resolution images.

engine and attempt to conduct both continuous and discontinuous prognostics using the TinyML techniques. It is worth pointing out that the C-MAPSS jet engine is not a perfect example for this distributed prognostics case study and we use it under an ideal assumption for demonstration purposes only.

7.3.1 Prototypical Hardware Deployment Validation

The practical question is whether these application models become too difficult to deploy yet need to have acceptable performance. To validate the practicality of deploying models on the cloud as well as on the edge, we provide a prototypical hardware deployment of these models on a server GPU and an edge MCU computation platform as follows:

- **Cloud Hardware:** The cloud inference deployment was conducted on a workstation (server) with a 4 Core/8 Thread Intel I7-7700HQ processor, 64GB RAM and a 2560-core NVIDIA P5000 Mobile Pascal GPU with 16GB GDDR5X VRAM.
- **MCU Hardware:** The edge inference deployment was conducted on an ESP32-Cam Development Board with WiFi BT/BLE SoC module, a low-power dual-core 32-bit CPU with working frequencies of up to 240MHz, a built-in 520 KB SRAM with an external 4M PSRAM. It also supports OV2640 and OV7670 cameras with micro-sd card read/write options.

For the MCU hardware, we choose the ESP32 Wifi MCU family chip for our experiments. The ESP32 is a low-cost, low-power system on a chip series of microcontrollers integrated with Wi-Fi and Bluetooth capabilities. ESP32 uses a IoT-targeted architecture powered by a dual-core Tensilica Xtensa LX6 microprocessor,

which is more specific than ARM-Cortex series processors. ESP32 development is also officially supported by the popular MCU development tool of Arduino IDE so that many existing packages can be used on ESP32 without any barrier. Compared to Arduino MCUs, ESP32 has more hardware resources and communication methods embedded so it has become a popular choice for state-of-the-art IoT project development.

As we are only demonstrating the applicability of TinyML deployment on the device, we use the USB cord to power the device. But it is worth pointing out, in practice the low power consumption of these MCUs enable the possibility of working in totally wireless mode using solar power [143].

7.3.2 Case 1: Surface Crack Analysis

7.3.2.1 Problem Description

Crack occurrence and propagation are critical factors that affect the reliability of solid structures. Correspondingly, crack analysis plays a vital role in health monitoring and inspection of structures like buildings/bridges/roads/equipment. Traditional inspection on cracks require heavy manual load, and image processing and computer techniques help formulate automatic analysis pipelines [144].

We consider a crack analysis pipeline consisting of two phases of detection and segmentation [145]. Crack detection judges whether there exists cracks in the region of a given structural image [146]. Given an input image with likely damage, crack segmentation further detects crack locations on images in a more detailed and accurate manner [147]. For our study, the detection model training and evaluation is conducted using the Concrete Crack Images for Classification Dataset [148]. The segmentation model training and evaluation is conducted using the Crack500 Dataset [147].

7.3.2.2 System Model

Although past efforts have shows reliable analysis from static images, with the development of TinyML, we can imagine a more flexible deployment of crack detector such as on a MCU-assisted structure patrol robot and provide detection feedback as early warnings in a real-time manner and send images with potential cracks to the cloud for further analysis.

In order to realize the working pipeline proposed above, we need to allocate different computation burdens to suitable devices as shown in Fig 7.4. We propose the device workflow of running binary crack detection on MCU using relative low-resolution $(32, 32, 3)$ RGB images and segmentation analysis on cloud using high-resolution $(224, 224, 3)$ RGB images.

	MCU Detection	Cloud Detection	Cloud Segmentation
Model Size	584.9 kb	22.4 mb	40.8 mb
Precision	INT8	Float32	Float32
Input Type	(32,32,3) RGB	(224,224,3) RGB	(224.224.3) RGB
Output Type	Yes/No	Yes/No	Segmentation Mask
Latency	479 ms	2.4 ms	7.3 ms
Performance	Accuracy: 99.5%	Accuracy: 99.7%	mIOU: 0.5075

Table 7.1: Crack analysis performance on devices. MCU side runs a quantized model for this binary detection inference task. It gains good classification accuracy performance with only about $1/40$ model size. In contrast, segmentation has high computation burden and can only be deployed on the cloud.

7.3.2.3 Results

These years' development of deep learning has shown the success of convolutional neural networks [148]. Both the cloud and MCU side models are used for computer vision tasks and we make use of mature convolutional neural network(CNN) architectures for these tasks.

On the MCU side, we make use of the Edge Impulse cloud platform [139] to train a crack detection model. This model training is using transfer learning based on a pre-trained Mobilenet [149] for computer vision tasks. The training process takes less than 10 minutes with 6000 training images and 2000 testing images with a resolution of $32 * 32$. The cloud platform also generated C code for MCU deployments.[1]

On the cloud side, we use the Nvidia GPU for model training. The segmentation model is based on the U-Net [150] architecture. It takes in high-resolution surface images and returns segmentation masks to show the exact location and shape of cracks in the region.

We summarize experimental results in Table 7.1. We can see that for this binary detection task, the TinyML model on the MCU achieves as good classification accuracy as the cloud model with only about $1/40^{th}$ of model size. One thing worth noticing is that the model on the MCU side is an optimized compressed version with 8-bit integer weight/activation quantization [151]. Compared to full-precision 32-bit floating point representations, 8-bit representations save significantly on model size. The segmentation models need to compute and output the actual crack region. The computation burden and file read/write size is much higher than the binary detection. MCUs have neither enough computation power nor storage space for further segmentation analysis. Thus, it is necessary to put the segmentation model on the cloud side. The evaluation metric of this segmentation task is called Mean Intersection-Over-Union (mIOU), which refers to the average prediction bounding box overlapping for classes of objects [147]. Our implementation is based on the work presented in [152] and has achieved normal performance on this segmentation task.

[1] Project publicly available on EdgeImpulse: https://studio.edgeimpulse.com/public/32815/latest

7.3.3 Case 2: CMAPSS Jet Engine Prognostics

7.3.3.1 Problem Description

The C-MAPSS dataset [88] is a dataset for data-driven remaining useful life (RUL) prediction generated from detailed simulations of jet turbofan engines. The data trace for an engine starts from a degrading time point and ends (RUL=0) at the end of the running cycle. Apart from time label, there are 24 features for each data point as shown in Table 7.2. The first three are the three operational settings that have a global impact on engine performance.The remaining ones represent the 21 sensor values from different sub-components of the engine system.

No.	Parameter	Detail	Group
1	C1	Control input 1	0
2	C2	Control input 2	0
3	C3	Control input 3	0
4	T2	Total temperature at fan inlet	1
5	T24	Total temperature at LPC outlet	2
6	T30	Total temperature at HPC outlet	3
7	T50	Total temperature at LPT outlet	5
8	P2	Pressure at fan inlet	1
9	P15	Total pressure in bypass-duct	2
10	P30	Total pressure at HPC outlet	3
11	Nf	Physical fan speed	4
12	Nc	Physical core speed	6
13	epr	Engine pressure ratio (P50/P2)	0
14	Ps30	Static pressure at HPC outlet	3
15	phi	Ratio of fuel flow to Ps30	3
16	NRf	Corrected fan speed	4
17	NRc	Corrected core speed	6
18	BPR	Bypass Ratio	2
19	farB	Burner fuel-air ratio	5
20	htBleed	Bleed Enthalpy	0
21	Nf_dmd	Demanded fan speed	4
22	PCNfR_dmd	Demanded corrected fan speed	4
23	W31	HPT coolant bleed	5
24	W32	LPT coolant bleed	5

Table 7.2: Sensor feature grouping according to their semantic structural contexts. Different groups refer to different sub-components of the system.

For demonstration purpose, we only choose the first set $FD001$ for our experiments here. Data preparation steps are conducted on this dataset including all zero value feature removal and normalization into the 0.0-1.0 value range using MinMax [95]. The remaining 17 features can be divided into 4 groups based on their semantic structural locations.

1. Global: $1, 2, 20$

2. Fan+Splitter: $11, 16, 5, 9, 18$

3. HPC+Fuel: $6, 10, 14, 15$

4. Burner+HPT+LPT+CoreNozzle: $7, 23, 24, 12, 17$

We regard sensors of these different groups as spatially separated and have MCUs deployed in groups of $2, 3, 4$ and explore the cooperation of tiny prognostics models on MCUs and the holistic prediction model on the cloud. It is worth pointing out that this distributed prognostics system setting is based on several ideal assumptions. First of all, we regard these sensors as separable into distributed groups and can be connected to MCUs separately. In addition, we do not consider the extreme condition (like high temperature or pressure in the engine) working applicability of MCUs. Moreover, the experiments are conducted on stable Wifi connections. Overall, we regard this as a use case for demonstrating a scenario that aims to explore potential settings for distributed prognostics from different sensors (installed on sub-components) in a sensor network. We are using this dataset partly due to the fact that we failed to find a better dataset for flexible demonstration of use cases under diverse data division settings.

7.3.3.2 System Model

Based on the overall problem discussed above, we build models on both MCUs and cloud servers to conduct prognostics. From the functionality point of view, TinyML on-device prognostics provides the early warning from the sub-component level and the cloud prognostics provides the more accurate system-level analysis. Here we use TFLite-Micro [153] from Tensorflow team to transform machine learning models and execute models on MCUs with C-based interpreters [154].

For this RUL value regression problem, we use a relatively simple network architecture on the MCU side. We use a three-layer multilayer perceptron [155] with $16, 32, 16$ neurons in intermediate layers. This simple model structure is also shown in Fig 7.2. This specific network refers to the model architecture of MCU2, which absorbs data from Group2. The input of the model is flattened into one dimension and sent into the model for forward inference computations. The input of dimension 40 is first sent into an encoder model and encoded as a latent representation of dimension 32. Then this latent representation is further utilized for remaining useful life regression prediction and outputs one floating point value as the final result.

One thing worth mentioning is that the input data we are dealing with belong to time series values so it would be a more optimal choice to use network models with temporal considerations like LSTM [69] in recurrent neural networks. Unfortunately, these recurrent neural networks have not been supported as kernel operations in current TFLite-Micro [153] yet.

For the continuous working mode, all sensor data updates are sent to the cloud server. We build a standard MySQL server as the cloud database to store these incoming time series data. On the cloud side with full machine learning framework support, a more standard LSTM network is built for more accurate prognostics. Here the LSTM model has two layers with 100 and 50 units. Moreover, the output of the second LSTM layer is connected to one fully-connected neuron to output one final prediction value.

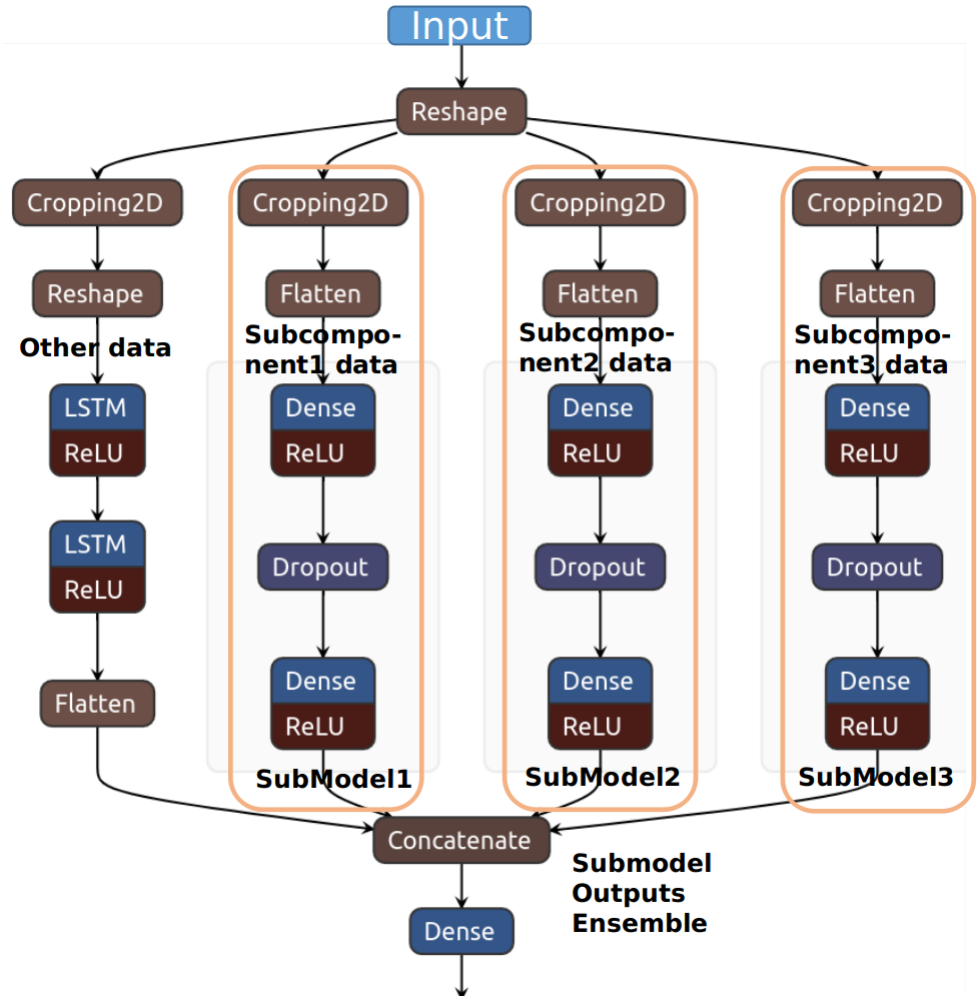

Figure 7.5: Model ensemble paradigm for the incontinuous working mode. Here we show the concatenation of latent space representations from three MCU submodels.

For the discontinuous working mode, we need to fetch a batch of data from a set window length of time to conduct prognostics tasks. We propose a submodel ensemble paradigm as shown in Fig 7.5. We first train MCU submodels for sub-components with different groups of sensor data. We then concatenate outputs of these tiny models but freeze parameter values in these tiny models. The global impact features are only used in the high-level holistic model. In this way, we retrain this holistic model for an overall prediction. One key point in the submodel is that we are only concatenating the outputs of the first half of the 'Input Encoder' of

	MCU1	MCU2	MCU3	Cloud
Model Size	57.9 kb	54.0 kb	58.0 kb	1005.6 kb
Precision	Float32	Float32	Float32	Float32
#Params	1905	1745	1905	35251
Latency /μs	638	588	647	94.2
Regression MAE	26.29	27.47	25.15	24.02
Data Transmission	fp32*6	fp32*5	fp32*6	fp32*17

Table 7.3: RUL Predictions Under Continuous Mode

	MCU1	MCU2	MCU3	Cloud
Model Size/kb	40.5+20.3	36.5+20.3	39.6+20.3	168.4
Model Precision	Float32	Float32	Float32	Float32
#Params	1360+545	1200+545	1360+545	9137
Latency/ μs	531+152	487+155	522+161	104
Regression MAE	26.29	27.47	25.15	24.7
Data Transmission Uncompressed	fp32*50	fp32*40	fp32*50	fp32*17*10
Data Transmission Compressed	fp32*32	fp32*32	fp32*32	fp32*(96+30)
Bandwidth Save	36%	20%	36%	25.90%

Table 7.4: RUL Predictions Under Incontinuous Mode

the Fig 7.2. In this way, we are only transmitting the latent space representation values with fewer dimensions than the model input and the communication bandwidth can be saved.

7.3.3.3 Results

Even though we are focused on the tiny machine learning model deployments on MCUs, current 32-bit MCUs like ESP32 can actually be used in more flexible ways. For this use case, the ESP32 MCU serves as a simple but multi-functional server. Fig 7.6 shows an example of such a simple server running on MCU. It can support TinyML prediction along with data visualizations. It stores sensor data for a past certain length of time. When new data comes in, it would post these data to the cloud server to make global RUL prediction using a more holistic model. But on the ESP32 itself it runs a lightweight RUL prognostics model for on-device inference, so if it reaches a threshold it should actively send alarm to the client user. Making use of these tiny MCU prognostics servers, we gather data and run models with these data.

We show experiment results from the continuous mode in Table 7.3. For value regressions, we cannot use quantized models to save model size. But still these tiny prediction models can run on low-power MCUs with low-latency. The metric of mean absolute deviation (MAE) is used for evaluation of the model performance.

Similarly, we show experiment results from the discontinuous mode in Table 7.4. As we have mentioned, to get access to the intermediate latent space representation, we divide the original predictor into the input encoder and the output predictor. The total number of model parameters remain the same. But some storage cost has been induced to store more model structural information. With the latent space representation, the

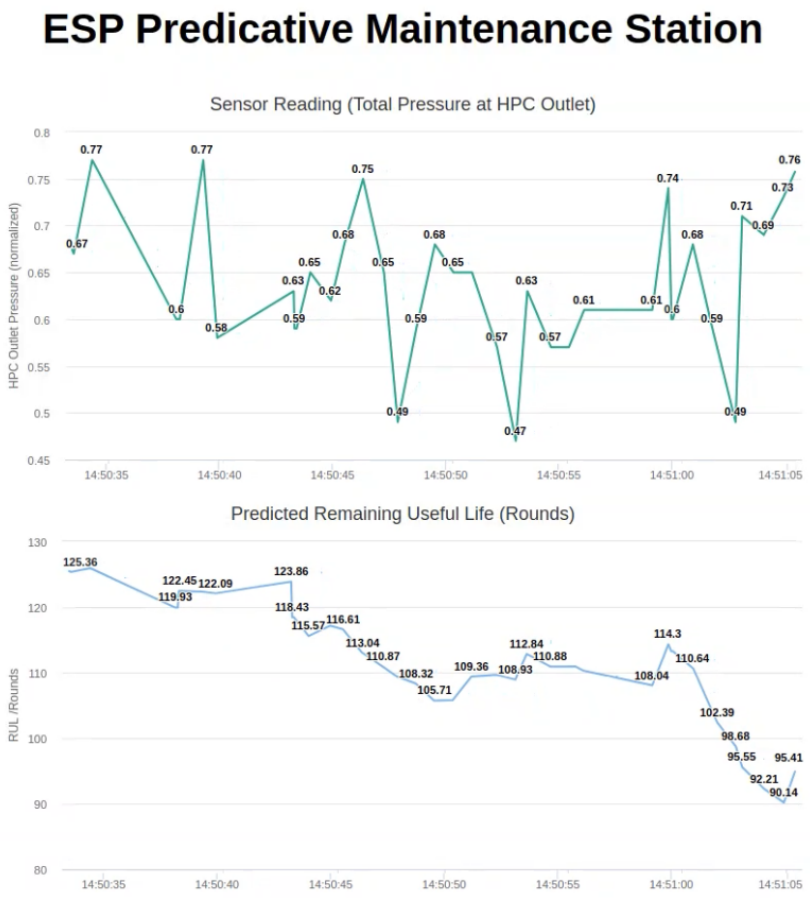

Figure 7.6: A tiny prognostics server deployed on MCU. It can support TinyML prediction along with data visualizations.

data transmission size can be greatly reduced. Again, we get an overall better system-level prediction with the information from all submodels and the global features.

7.4 Conclusion

Large scales and big data volumes in practice call for more resilient data-driven models in health management and prognostics scenarios. Development of current machine learning, especially deep learning enables the execution of prediction models on different levels of computation devices. This paper serves as a vision paper for the novel field of TinyML opportunities on MCU devices in PHM application scenarios. We not only emphasize the potential of TinyML but also the significance of combining it with high-level cloud computing from a more systematic point of view. We explore the question of how cloud platform can help build efficient TinyML models. We show the significance of inducing cloud resources for data storage and integration. Furthermore, we investigate the possible application patterns for more adaptive computation burden offloading from the edge MCU to the cloud. Our discussions and experimental implementations on

two case studies cover the most typical settings in PHM applications. These general settings would help formalize the system resilience testing in more scenarios.

Even though we have provided a widened vision of TinyML applications in CPS, it is worth pointing out that our current exploration also shows limitations in embedded machine learning on MCUs. First, our current investigation is based on the Wifi network as the communication method. Practical PHM application scenarios may be more complex for such a universal wireless connection. We may need to further investigate the impact of unreliable data communication on TinyML deployments for various CPS applications. Secondly, we demonstrate the possibility of using machine learning model for data analysis and use the latent space representation as the data compression to save bandwidth. But there are other methods that are also worth exploring like PCA and clustering. In addition, due to lack of support for RNNs in current TFLite-Micro [153], we have not achieved ideal performances on time series data. This also limited us from exploring unsupervised models like LSTM Autoencoders [136]. Thirdly, even though our system-level ensemble framework has shown promising performance on test settings, the strategy of model ensemble is still under manual configuration. Therefore, we need to find out ways to search for more flexible model ensemble strategies. Finally, even though many MCU models have shown to be capable of running TinyML, there is still a general lack of standardized prototypical hardware device and framework options [156]. Overall, the combination of TinyML and cloud computing would have a promising future for many CPS application scenarios.

CHAPTER 8

Motivation Case: Simple Black-box Adversarial Attack WITH&ON BNN

Based on practical edge hardware deep model deployment procedures discussed in Chapter 6, we propose to combine edge computing and adversarial machine learning together. This chapter shows a simple but efficient black-box adversarial attack test has been conducted on low precision neural networks (especailly binary neural networks) running on FPGAs. Experiments reveal the potential risk for edge machine learning models and serves as the motivation for further research. It is worth pointing out, even though this is not a systematic research work, this motivation case shows that the BNNs on FPGAs can be in danger as well as dangerous. The real-time inference speed makes it possible for an attacker to use a BNN on one device as a straightforward weapon to attack other BNN models.

8.1 Problem Overview

Advances in FPGA hardware accelerator technologies and their low power consumption and high flexibility have made them an attractive choice to realize a range of machine learning (ML) applications for edge/IoT systems. Despite this promise, FPGAs are prone to vulnerabilities, e.g., adversarial attacks, which are a significant threat to machine learning systems. Adversarial attacks are those where synthetic modifications are made to the original samples while training that then can misguide the neural network prediction systems. In other words, these are inputs to machine learning models that an attacker has intentionally intercepted and disguised in order for the prediction model to make mistakes.

Even though security has always been a popular topic in the FPGA field, the combination of programming logic and adversarial examples is still an area that remains vastly unexplored. Only recently has some research shown how hardware accelerators can be useful for real-time defense against some adversarial attack [157; 158]. With an increasing range of FPGA-based applications using deep learning models for edge/IoT systems, security issues like adversarial examples cannot be neglected thereby requiring more research on understanding the attack characteristics and defense mechanisms. This paper focuses on illustrating the adversarial attack case for FPGAs. Specifically, in this paper we present a simple black-box utilization of hardware accelerators in edge/IoT environments that are used to speedup the adversarial attack against neural network machine learning systems.[1]

The main contributions of this paper are the following:

- We show how it is possible to realize high-speed adversarial attacks using programming logic. We further show the potential risks of machine learning systems on the edge/IoT, especially for the popular

[1] code: https://github.com/paperAlterSubmission/BlackBoxAdvPynqBNN

quantized neural networks.

- We present a general procedure that can be utilized to implement black-box adversarial attacks using FPGA devices. This procedure is general enough to be easily adapted to other existing learning systems and can be regarded as a generalized safety testing procedure.
- We show a novel application setting of utlizing an existing hardware IP from an absolute high-level without going through the whole hardware design procedure. This can be regarded as a hardware-oriented algorithm design process that can be useful for designers that are more focused on the algorithm application side.

8.2 Background

Even though deep neural networks have been widely used, due to the complexity of current networks, the principles of deep networks remain a black box. Researchers have found some methods that make small modifications to an input image without harming original information, which then can lead the neural networks to wrongly classify the image [34].

8.2.1 Binary Neural Networks (BNNs)

New generations of programming logic devices are equipped with stronger specifications than before. However, the internal structures of FPGA devices determine that they are more efficient in fixed-point computations [159] rather than more widely used floating-point computations on normal CPUs. Therefore, quantized and binary neural networks with fix-point value represention have become the most suitable type of inference tool for FPGA devices. Current applications of BNNs show that they can perform hundreds or even thousands of inferences per second. This means the inference can almost be regarded as real-time. The most typical BNN realization by far comes from Xilinx [108] trained on MNIST and CIFAR10 datasets but can also be used on other tasks. One feature among all inference libraries is that the inference can return the results in a vector of confidence score values on classes and the final inference result class is the one with the largest confidence value. The returned vector would be used for estimating the sensitivity of pixels.

8.2.2 Gradient-based Attack

Many attack methods have been developed by far, but the most well-known and fundamental is still the FGSM (Fast Gradient Sign Method) [36].

$$\eta = \varepsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{8.1}$$

Here θ represents the parameters of a model, x is the input to the model, y refers to the targets associated with x (for tasks with targets) and $J(\theta,x,y)$ is the cost function used to train the neural network. Further, ε is the magnitude constraint added to the original sample. This method is quite simple and intuitive. The attacker makes modifications to maximize the loss function. As the modification is small enough it actually preserves the original information structure.

Since then other attack methods have also been designed [160]. All these methods are most optimal to be implemented on GPUs and can hardly be practical for edge/IoT environments, which have stringent limitations on device power and size. This paper shows how to implement a simple adversarial attack for BNN-based ML executing on FPGA. It shows how a small and low-power programming logic device can also be dangerous and universal weapon against neural network machine learning systems on the edge/IoT. The method implemented in this project can be viewed as a modified version of the Fast Gradient Sign Method (FGSM) algorithm specially designed for FPGA devices.

8.3 Overall Architecture

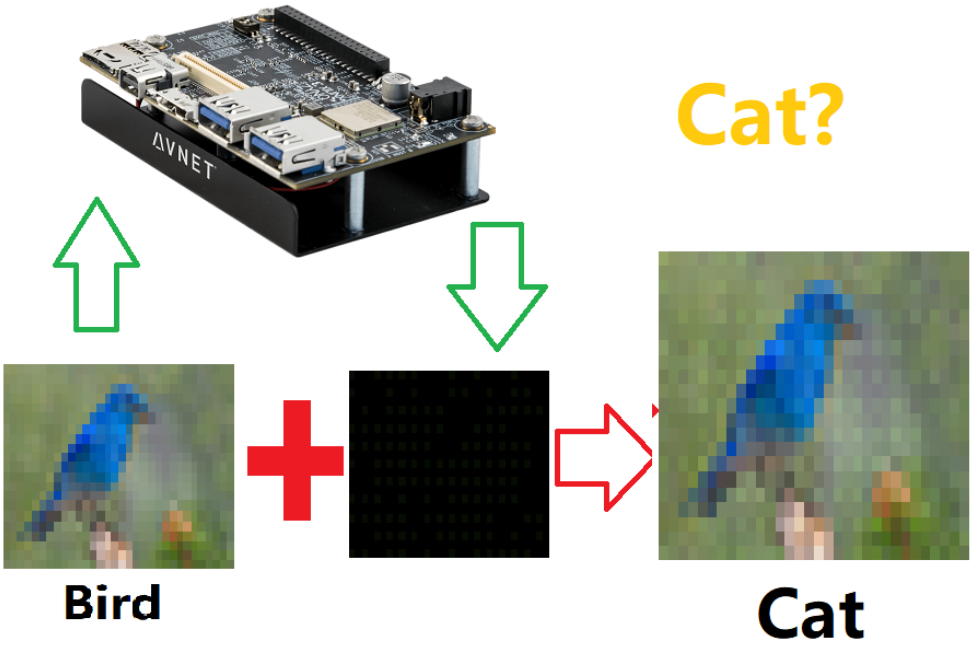

Figure 8.1: End-to-end Input and Output.

The overall attack procedure is designed as end-to-end. As shown in Figure 8.1, the attack framework inside the programming logic device receives an image and returns a same-size image with adversarial characteristics. This image is more likely to be mis-classified by end user's model. In this case, the modified image of a bird is classified as a cat by the BNN.

We implement a similar procedure to FGSM [36] discussed before. For FGSM, the execution can be divided into two parts. The first part retrieves the gradient of loss function for each pixel. The second part is to to obtain the sign of the gradient matrix and add perturbation. For our application, a more detailed flowchart is shown in Figure 8.2. We can see that the computations mainly lie in two parts:

- Estimate pixels' sensitivity to prediction confidence.
- Compute signs of sensitivity to add perturbations with a certain magnitude.

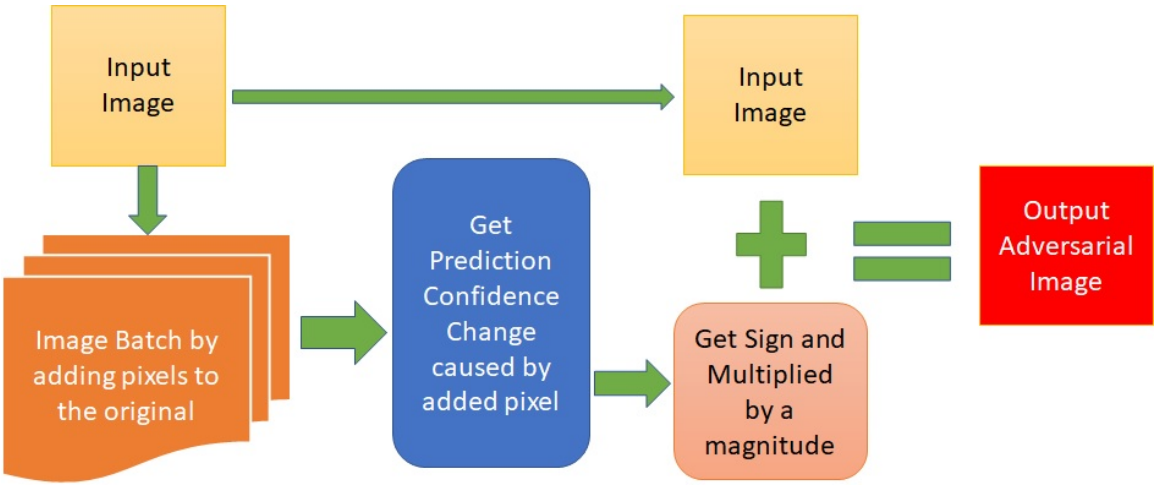

Figure 8.2: Overall Procedure of This Attack.

The first part is more difficult. For current BNN applications, the inferences are one-way and do not provide the function of computing arbitrary graphs like returning pixel-wise gradients to the loss function. As it returns confidence vectors, we can use these return values to construct a sensitivity distance function. As a result, we can regard this classifier as a black-box and use this metric to estimate importance of pixels instead of using gradients.

As the inference on the BNN is very fast, a very straightforward idea can be implemented here: modify one pixel in an image and measure the classification confidence change for this image. Iterating through pixels in an image and we can get the pixels' sensitivity to prediction confidence. Consequently, learning from the classic FGSM method, get the sign of these pixel changes and multiplied by Lambda to get noise added.

8.4 Attack Design and Optimization

In this section, we dive into the details of the black-box attack algorithm we designed and optimize for programmable logic devices.

8.4.1 General Idea

Current black-box adversarial attacks against neural networks comprise of two main technical paths. The first tries to obtain a substitute neural network to imitate the original network and the attack on this substitute network is highly likely to be transferred to the original neural network [8]. Transferability is an important feature of adversarial examples [161], which makes it possible to directly use accessible pre-trained neural network models to make adversarial attack in a direct or ensemble way against a new model with high success rate.

The second is more similar to popular paths of obtaining cost-sensitive samples. Using a number of queries to make estimations of the gradients to the target neural network is easier than training a totally new network from scratch [9; 162]. This can also be combined with some other optimization techniques like local search to get some accelerations [163].

Our attack method is closer to the second type of gradient estimation. However, there are two significant differences worth pointing out. First, we do not seek to accurately compute the gradient values with a general framework and we are only interested in what pixels would be more sensitive to the current prediction system. Second, as we are implementing this attack on a low latency FPGA device, we are only seeking a relative high successful attack rate for each image[164] but we are pursuing the consecutive adversarial impact in time. That means the attack frequency also matters. This would be a critical point for attacking internet of things devices in a streamline setting.

8.4.2 Pixel Sensitivity Estimation

The most important problem for us is how to define sensitivity of pixels without direct computations of gradients? A metric is required to do this measurement. As mentioned, the classifier returns an array containing prediction confidence for each class and the class with the highest confidence would be the classification result [108]. To further reduce the computation burden, we just use the confidence gap between the largest and second largest as the prediction robustness metric. We set a threshold value to filter out very small changes and the rest would be regarded as sensitive pixels need to be modified. That is, for an output confidence score vector:

$$S = \{score1, score2, ..., scoreN\} \tag{8.2}$$

where N is the number of output classes, we define the sensitivity metric in the following form:

$$Sensitivity = max(S) - max(S \setminus max(S)) \tag{8.3}$$

as a confidence variation metric for evaluating the sensitivity when the value of a pixel changes. The semantic meaning of this metric is direct: it refers to the difference between the most confident score and the second most confident score, which can be viewed as the absolute confidence of the prediction. This is partly highlighted by the difference form from Hinge Loss [165] but still holds the requirement of simplicity.

8.4.3 Multi-Channel Approximation

For color images, there are three color channels and normally in adversarial perturbations, the three color channels are aligned separately for their sensitivity values to the overall cost function. To further accelerate the computations throughout multiple color channels we further implement a simple but useful approximation technique here. Since the three different colors of red, green and blue have three different wavelengths and make their own contributions in the formation of a standard colored image. We can take average according to their contributions, rather than using average values of the three.

In common digital application, we use the standard equation below to convert RGB images to grayscale [166]:

$$Gray = 0.2989 * R + 0.5870 * G + 0.1140 * B \tag{8.4}$$

This means that the three colors contribute with different weights for natural light. An intuitive explanation is red color has the longest wavelength of all the three colors, green has the second longest wavelength but also gives more natural smoothing and blue contributed the least. To make it more compatible with fixed length computation, we further simplify this equation as a pure integer format as shown below:

$$Gray = 3 * R + 6 * G + 1 * B \tag{8.5}$$

From this approximation, we define perturbation with magnitude d as

$$[R', G', B'] = [R, G, B] + d * [3, 6, 1] \tag{8.6}$$

That is, a unit perturbation in our attack is a weighted modification added to the three channels in the vector form at the same time. To the best of our knowledge, we do not know of any other approach that implements this kind of color channel approximation technique for neural network perturbation analysis. We believe this would be useful for many applications involving multiple coherent information channels.

8.4.4 Final Attack Algorithm

In all, we get a simple integrated black-box adversarial attack framework. A more formal description is given in Algorithm 8. There are three hyperparameters here in the attack settings:

- Step. The sampling stride for choosing pixels, for examples, step=2 means choose pixels every x=x+2 or y=y+2. You can understand this as a sampling density;
- Thres. A threshold value for choosing sensitive pixels among sampled pixels;
- Lamda. The magnitude of perturbation added to the original image.

Algorithm 8 Simple Black-box Adversarial Attack using BNN

Require: $\boldsymbol{X}$: an input image consisting of n pixels $X = \{x_1, x_2, ..., x_n\}$. And $x_i = x_{i-1} + Step$; $Thres$: A threshold value for choosing sensitive pixels among sampled pixels; $Lamda$: The magnitude of perturbation added to the original image; d: the unit testing magnitude of adversarial deviation; $Lamda$: the maximum magnitude of adversarial deviation; $\boldsymbol{BNN}$: binary neural network predictor; max:return the maximum prediction score from a classifier.

Hardware Initialization(Download BNN to FPGA board)
$BNN(X) \leftarrow S_0$
$X_{modified} \leftarrow X$
for $x_i \in X$ **do**
  $X_{tmp} \leftarrow \{x_1, x_2, ..., x_i + d, ..., x_n\}$
  $S_{tmp} \leftarrow BNN(X_{tmp}) - S_0$
  $Sensitivity_i \leftarrow max(S_{tmp}) - max(S_{tmp} \setminus max(S_{tmp}))$
  if $Sensitivity > thres$ **then**
    $X_{modified}[i] = x_i + Lamda * sign(Sensitivity)$
  end if
end for
return $X_{modified}$

8.5 Experiment Results

Experiments of adversarial attack against neural network are implemented on an Ultra96 Zynq Ultrascale+ Development board with PYNQ operating system.

8.5.1 Experiment Settings

Ultra96 is an Arm-based, Xilinx Zynq UltraScale+ MPSoC development board based on the Linaro 96Boards specification [167]. It is a newer generation Zynq development board equipped with Ultrascale+ programmable logic chip. It comes with micron LPDDR4 memory providing 2 GB of RAM in a 512M x 32 configuration. Wireless options include 802.11b/g/n Wifi and Bluetooth 4.2 (provides both Bluetooth Classic and Low Energy (BLE)). This is a powerful prototypical programmable logic board with a balanced configuration which makes it an ideal test platform for internet of things applications.

As mentioned, since this is a simple black-box attack, the application involves only high-level utilization of programming logic. To simplify the algorithm realization process, the PYNQ [168] operating system is used and only Python is required here to implement the attack. The attack program is also tested working on Digilent PYNQ-Z1 baord and in theory should be working on all Zynq devices that can run PYNQ operating system and its corresponding version of binary neural network. We are using the officially supported BNN package [169] provided by Xilinx to conduct batch inference experiments. It can be executed at an optimized stable frequency of 300MHz.

8.5.2 Test Results

As batch testing is still quite time-consuming even when run with hardware acceleration, we choose the first 200 samples from the cifar10 test dataset to evaluate the performance of the attack. Given different hyper-parameters, we first evaluate the attack error rate under BNN on PYNQ itself. The meaning of black-box in the attack is that the attacker does not know the exact internal structure and values of the prediction model and can only get access to the output results given inputs.

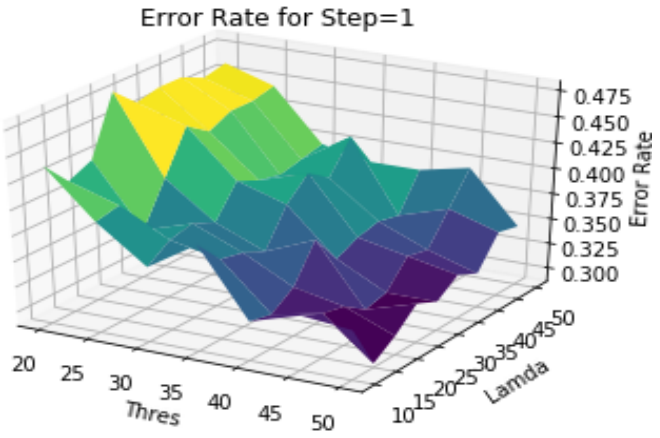

Figure 8.3: Error Rate for BNN Model(step=1)

Figure 8.3 gives the error rate under different settings. We can see the impact of this simple attack. The selection of hyper-parameter values also affects the attack success rate. We ran batch experiments on different hyper-parameters and use visualizations to give a more direction impression on their impacts.

One critical aspect of neural network vulnerability is that a same set of adversarial examples can be transferable [161] among different models. We also conduct experiments on a standard convolution neural network model [170] for comparison. It can be seen in Figure 8.4 that the error rates are a little lower than the BNN model under same settings but still show obvious impacts on the prediction model.

Table 8.1 shows the number of queries in one attack required for a 32*32 image and average attack

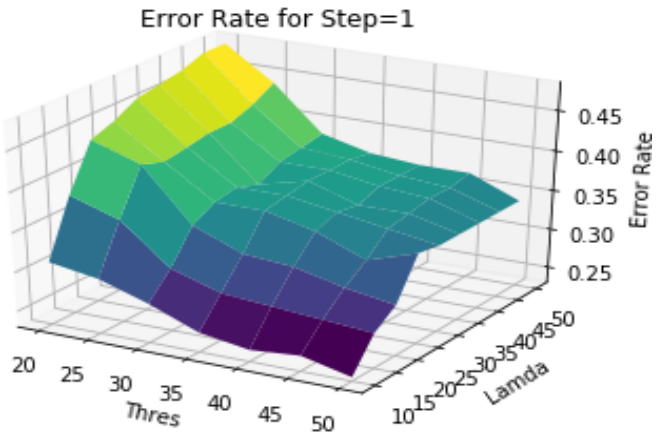

Figure 8.4: Error Rate for Standard Model(step=1).

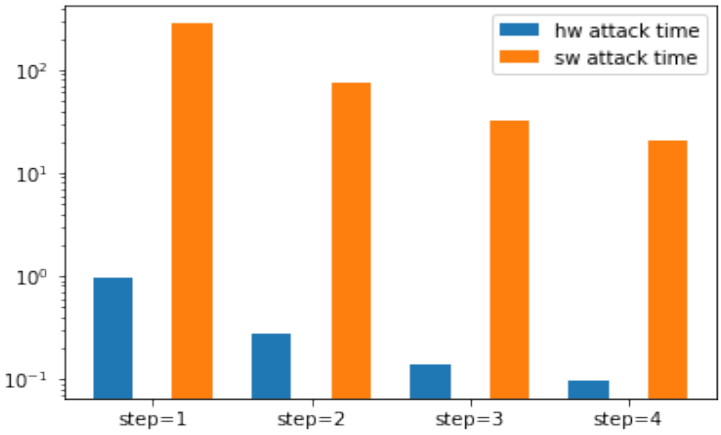

Figure 8.5: HW/SW Attack Speed Comparison.

time consumption for different step (sampling density) values on both host CPU(ARM CPU on board) and hardware accelerator executions. Ultra96 is equipped with an ARM Cortex-A53 CPU together with programming logic that supports pure hardware logic. The last row in the table gives an acceleration ratio of programmable logic execution to pure software execution. To make comparisons easier, Figure 8.5 provides visualizations on these time values.

For step=2, the hardware attack on one image takes only 0.27s while software attack takes 73.9s. Hardware executions show a 200x performance acceleration over software executions whatever the step value is. When the execution takes short enough, the attack can be more aggressive in a real-time manner.

Table 8.1: Time Consumption Comparison for Conducting One Adversarial Attack

PerAttack	step=1	step=2	step=3	step=4
NumQueery	1024	256	121	64
Hardware/s	0.97	0.27	0.16	0.096
Software/s	281.1	73.9	32.7	20.6
Acceleration	290x	273.7x	204.3x	214.6x

8.6 Conclusion

This paper shows a simple black-box adversarial attack against neural networks using BNN on FPGA. We show the process of making a hardware-oriented adaptive algorithm design and adoptation. We also show that potential applications of quantized neural networks might be under threat with a simple attack procedures. And with the development of more advanced hardware accelerators on the edge like Jetson Nano embedded GPUs [117] or ASICs like Coral TPU [109] or Intel NCS [118], the threshold of conducting such kind of adversarial attacks on the edge with a low latency would be even lowered.

This paper is just a preliminary step of showing the power of programming logic or other potential hardware accelerators in the field of adversarial machine learning. Current development of new synthesis and interacting tools provide better ways to make developers more focused on algorithmic issues. Due to their low latency and low power consumption levels, we believe hardware acclerators like FPGAs will play more and more important roles in edge/IoT scenarios.

One new type of threat worth mentioning against edge/IoT machine learning systems is the possibility of combining side-channel information leaks with adversarial attacks. A recent work from a research team in Cornell shows the possibility of reverse engineering the neural network internal structures and even weight information on FPGAs using side-channel information like power and cache [171]. This may under many realistic circumstances change the current adversarial attack settings in edge/IoT systems. In this way, some past black-box adversarial attacks can be boosted to gray-box or even white-box level and edge/IoT systems will be under more serious threat.

CHAPTER 9

Universal Adversarial Attacks on Quantized Neural Networks

Motivated by the edge model deployment procedure shown in Chapter 6 and potential potential vulnerabilities of edge LECs in Chapter 8, this chapter further tends to dive into the adversarial robustness of LECs on edge from a joint view. We study the potential threat of using a same universal adversarial perturbation to compromise all prediction models targeting the same task but with different quantization compressions across the cloud/edge pattern. To solve this issue, we also propose a defensive model deployment workflow.

9.1 Problem Overview

Deep learning-based machine learning (ML) models are becoming commonplace in cloud-based services, such as in object detection, image classification, speech recognition, etc. However, due both to real-time response requirements and privacy concerns, many of these services, particularly those used in video surveillance or proactive maintenance in industrial plants, must rely on edge computing [45]. Edge services are designed and deployed in such a way that the critical, time-sensitive components execute at the edge where ML models are used for prediction tasks [172] and the remainder components are deployed in the cloud.

Edge computing is, however, characterized by extreme heterogeneity in its resource types and significant constraints on their capacities (both compute and power). Consequently, the complex and large ML models that are traditionally utilized in cloud-based services cannot naïvely be deployed at the edge. To address these concerns, ML model compression techniques, such as model quantization [49], are attracting significant attention. These methods support high compression ratios while providing acceptable accuracy loss thereby enabling a practical cloud-edge application, where full-precision models are deployed on cloud servers and lightweight, quantized models on edge devices.

The resource heterogeneity at the edge, however, precludes a one-size-fits-all quantized ML model for a given full ML model; rather the quantized model's network architecture will vary based on the quantization settings [173]. To automate the transformation from full-scale to quantized models that are customized for the underlying edge hardware platform, solutions such as hardware-aware training [174] and model compilation [175] have been proposed. One recent work designs a systematic model compression pipeline to generate one neural network and then specializes it for efficient deployments across diverse platform configurations [176].

Despite the widespread use of ML models and their quantized counterparts at the edge [177], ML models are shown to be vulnerable to adversarial attacks [36], which are carefully designed inputs with bound-limited (i.e., small and almost undetectable) added perturbations that can greatly mislead the pre-trained models

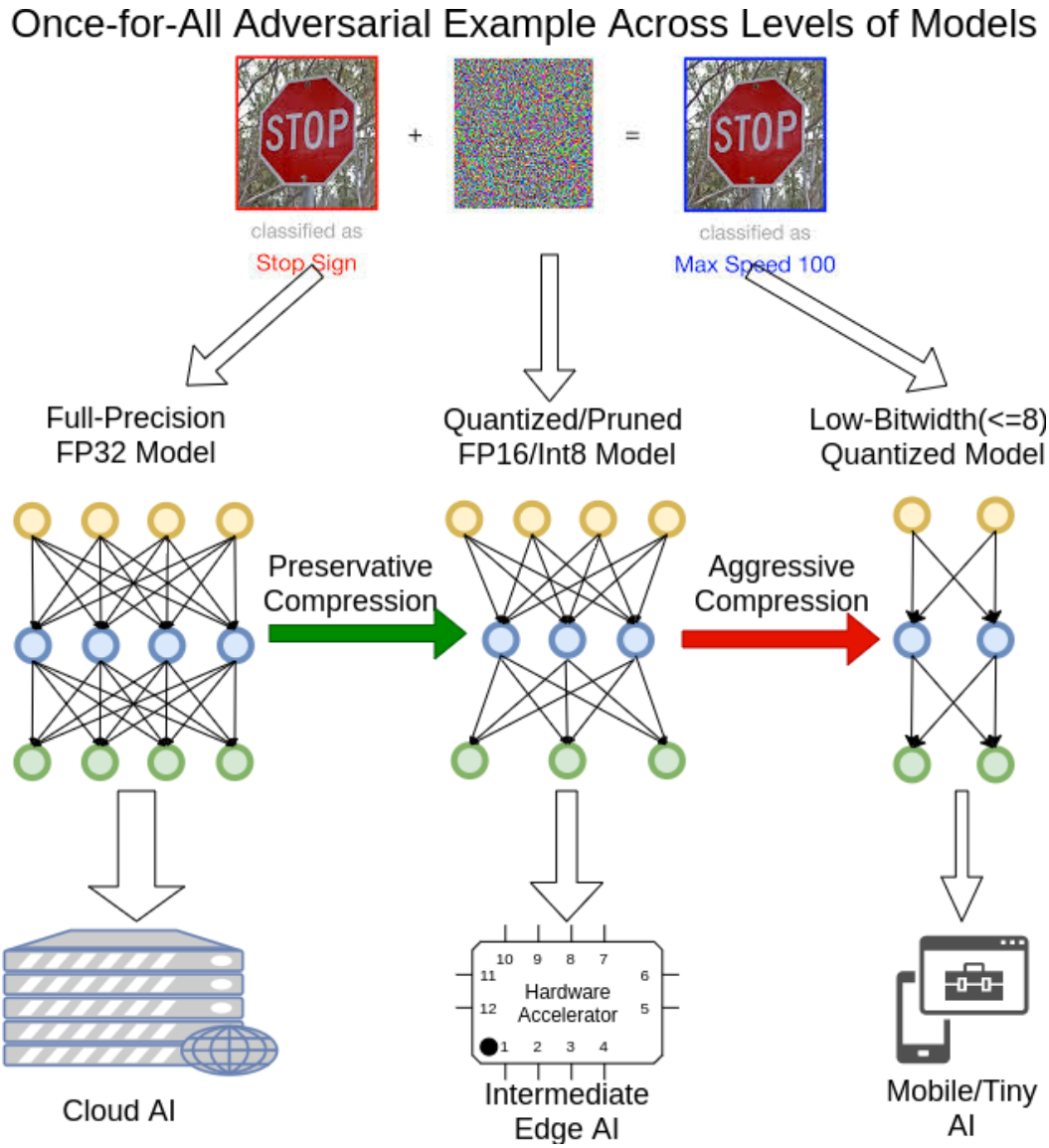

Figure 9.1: Practical Deployment of Hardware-aware Quantized Models under the Threat of Universal Adversarial Perturbations (UAP). The same attack perturbation may lead to model performance deviations on devices across multiple deployment levels.

into making suboptimal or incorrect inferences. Past research has even shown the existence of universal adversarial perturbations (UAP), where a single instance or type of perturbation has the ability to adversarially impact almost all data samples [70]. Moreover, these perturbations are also easily transferable to other models in an entirely black-box manner.

Although a significant amount of research on adversarial robustness (i.e., being able to defend against adversarial attacks) of traditional ML models exists, only limited work exists on evaluating the robustness of quantized ML models [178]. The available literature suggests that model compression techniques set a

limit on the amplitude of the classification prediction score[1] caused by perturbation and therefore improve robustness, while other works even propose compression-based defense solutions [179]. There are also some research works that combine adversarial robustness with the lower-level circuit design point of view [180]. But mostly, these defensive trials put attention on the adversarial training phase [181; 182; 183] rather than dealing with existing models, which might be costly in some application cases.

Thus, for applications using the cloud/edge ML model deployment pattern where adaptive model compression is used for different types of devices at the edge, we question whether the following potential threat assertion holds for a range of deployments as shown in Figure 9.1: *Quantized models with different precision levels are vulnerable to the same universal adversarial perturbations (UAP) as non-quantized models.* Validating this claim is important because the existence of the same adversarial perturbations that can compromise models at all precision levels would greatly reduce the threshold of potential adversarial threats in practice.

To that end, we propose a defensive deployment framework realized within a middleware that comprises design- and run-time capabilities for quantized models under universal adversarial perturbation [70]. We show that the adversary of UAPs is highly correlated to the model generation method of post-training quantization. To solve this problem, we then develop defense mechanisms based on a novel approach of post-training quantization of adversarially trained models, which are then deployed in the runtime infrastructure to realize adversarially robust edge services. Our middleware can be viewed in the same spirit as prior middleware efforts that provide fault tolerance or security to applications.

We evaluate our ideas for different precision levels of quantized neural networks using the object classification datasets of CIFAR10,CIFAR100 and SVHN, which are suitable for applications, such as surveillance or wildlife monitoring, and are the most widely used computer vision datasets. The experimental results using our evaluation workflow reveal two interesting observations. First, the natural quantized models (i.e., without adversarial robustness) are still vulnerable to universal perturbations and the perturbations from some quantized models are even transferable to other quantized models. Second, the state-of-art adversarial training methods are efficient against universal perturbations. Moreover, adversarially-trained quantized models via post-training quantization preserves the robustness against these kinds of attacks and hence can be realized within a reusable and configurable, adversarial defense middleware for edge-based services thereby obviating the need for *reinventing the wheel* in realizing adversarially robust, data-driven edge services.

In summary, this chapter makes the following contributions:

- We present a reusable and reproducible evaluation workflow to formalize the security and resilience

[1] A neural network makes a classification inference by choosing a class that has the highest amplitude of the prediction score. This is often realized by a Softmax function as the model output.

testing of quantized neural network models under universal adversarial perturbation (UAP) attacks.

- We show the consistency of robustness of adversarially trained models and propose a defensive quantization method towards more robust quantized models that can be codified into a configurable and reusable middleware solution that edge services can utilize.
- We evaluate the UAP attack and defense on the most widely used CIFAR10/CIFAR100/SVHN datasets. We show the vulnerability of normal quantized models and highlight the compactness and the efficiency of our proposed defense strategies.

9.2 Adversarially Robust Data-driven Services: Design and Implementation

We now describe the design of our proposed framework that provides a reusable service to realize adversarially robust data-driven services at the edge. Our approach comprises two phases as shown in Figure 9.2: an offline phase and an online phase of which the offline phase is more significant and drives the deployment of the appropriate adversarially robust quantized model in the runtime infrastructure.

In Step A of the offline phase, a user-supplied ML model along with training data is submitted to our system to undergo adversarial training, which is the novel aspect of our approach. In Step B, this adversarially trained full, robust model then undergoes quantization and robustness evaluation to form robust quantized models of different precision levels. Subsequently, in Step C, depending on the constraints imposed by the underlying edge hardware, i.e., deployment target, an appropriate robust, quantized model is synthesized and deployed in the runtime to perform the prediction/inference operations at the edge in the online phase.

Step A can make use of existing adversarial adversarial training techniques [96] and has a general compatibility. Step C involves hardware synthesis, which relies on hardware-specific toolchains from hardware producers such as PyTorch machine learning framework support on NVIDIA GPUs. Our work validates the efficiency and necessity of state-of-art adversarial training in Step A and further proposes the generation procedure of quantized robust models in Step B. These together bridge the gap between robust ML models and efficient hardware deployment in cloud/edge environments.

9.2.1 Generating Universal Adversarial Perturbations Across Quantized Models

The question we posed in Section 9.1 (see Figure 9.1) seeks to understand if and how much are quantized ML models vulnerable to UAPs. As we noted before, there have been prior efforts that focus on adversarial robustness evaluation of quantized neural networks like binary neural networks [164]. However, a systematic investigation into different attack settings for UAP involving quantized models is still missing.

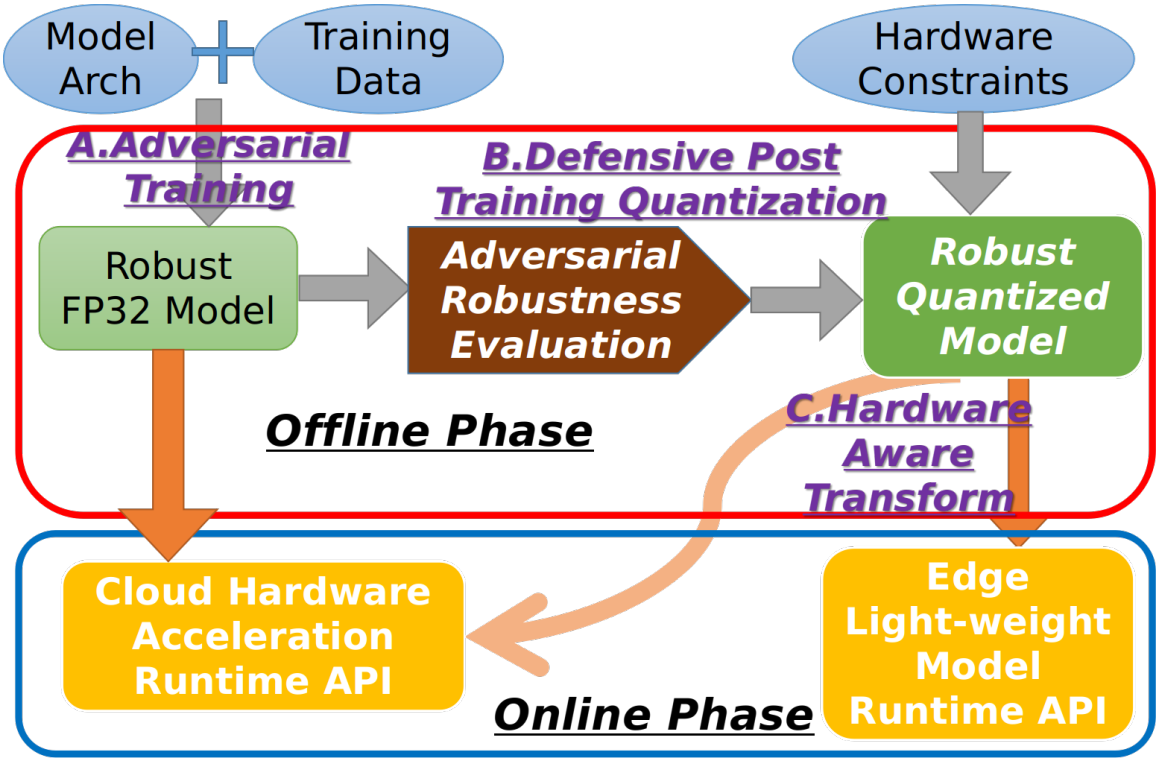

Figure 9.2: Proposed Workflow for deploying robust quantized models for cloud/edge computing. Full-precision models preserve performance on the cloud. Quantized models on the cloud are sometimes preferred for increasing computation throughouput. With limited hardware resources, quantized versions of robust models are preferred on the edge side. All compute intensive model preprocessing steps are conducted offline and only model inference is executed online and so no extra overhead is induced at runtime. Our work validates the efficiency of state-of-art adversarial training in Step A and further proposes the method towards quantized robust models in Step B. These together bridge the gap between robust ML models and efficient hardware deployment in cloud/edge environments (Step C).

We seek an automated attack generation across different precisions. A recent effort in universal perturbations has focused on *attack generation* using a single model [184] and then attempts to transfer the vulnerability to other models [161]. We extend this classical UAP generation algorithm and implement it in a more generalized way that supports exploration on multiple models as shown in *Algorithm 9*. epending on the amount of model architecture exposed by the user-supplied ML model to the attacker, Algorithm 9 is implemented on the white-box setting. This white-box setting assume that the attacker has full knowledge of the ML model like model architecture information and model weights.

Likewise, there could exist two settings based on the attack target. The *direct attacks* generate the adversarial perturbation from the testing data directly. *The indirect attacks* generate the adversarial perturbation from the training data and then add the perturbation to the testing data. To make this attack more realistic, we mostly consider the latter setting where the attack is implemented on the training data. Then the generated

UAPs are evaluated on the separate testing data.

Algorithm 9 Universal Perturbation using Model Ensemble

Require: $\boldsymbol{X}$: data points ; $\boldsymbol{F}$: predictor set containing predictors (with different quantization settings); $J(func, x, \Delta\boldsymbol{x})$: cost function of model $func$ according to input data x with a step size of α; $clip$: clips inputs in a range with a lower and an upper bound; α: modification step size for each iteration; ε: maximum bound distance allowed to be modified for each feature(L_∞ constraint).

1: $\Delta\boldsymbol{x} \leftarrow \mathbf{0}$, $i \leftarrow 0$
2: **while** $i < NumIter$ **do**
3: $\quad \boldsymbol{f} \leftarrow randomSelect(\boldsymbol{F})$
4: $\quad \boldsymbol{x} \leftarrow randomSelect(\boldsymbol{X})$
5: $\quad \Delta\boldsymbol{x} \leftarrow argmax \nabla J(\boldsymbol{f}, \boldsymbol{x}, \Delta\boldsymbol{x})$
6: $\quad \Delta\boldsymbol{x} \leftarrow clip\{\Delta\boldsymbol{x}, \Delta\boldsymbol{x} - \alpha, \Delta\boldsymbol{x} + \alpha\}$
7: $\quad i \leftarrow i + 1$
8: **end while**
9: $\Delta\boldsymbol{x} \leftarrow clip\{\Delta\boldsymbol{x}, \Delta\boldsymbol{x} - \varepsilon, \Delta\boldsymbol{x} + \varepsilon\}$
10: **return** $\Delta\boldsymbol{x}$

The algorithm uses an iterative attack procedure (line 2). Each prediction model is randomly selected (line 3) per iteration and the perturbation is gradually generated across the data points (line 4). A small attack step is taken per iteration (lines $5-6$). The final step projects the generated data back into its valid value range (line 9). This ensembles multiple predictors together for UAP generation and formulates an equivalent mixed-precision attack setting in the evaluation.

9.2.2 Deploying Adversarially Robust Quantized Models

From the adversarial attack in computer vision we know that an adversarial perturbation leads to large expansive increase in a target loss function with limited input modification and thus only invisible changes are made. As a result, to make a model more robust we can make its potential loss function non-expansive under constraints. In [179], the authors control quantized weights in adversarial training to suppress the network's amplification effect across layers so as to mitigate potential adversarial impacts. Similarly, researchers from [185] proposed an iterative model compression framework combining quantization and pruning in optimization to train more robust models.

These past efforts show possible ways to get more robust quantized prediction models. However, they are computationally expensive and cannot be combined with other defense techniques. Thus, we devise a novel approach where instead of training a model from scratch, we obtain a robust quantized model by directly quantizing a robust full model $\boldsymbol{f}_{rob}$ that is trained using techniques like adversarial training [186]. The consequence is that our approach is compatible with all state-of-art adversarial training defenses without any impediment. Obviously, the premise of this procedure is that the post-training quantization of the model preserves the adversarial robustness. To the best of our knowledge, the potential impacts of quantization on adversarially trained models have not been studied.

There is no formal definition of what is a robust model under adversarial attacks. One important reason is the diversity of defense policy deployment phases that take into account of the potential adversarial impacts. Given the evasion attack setting [35], defenses can be conducted as input data processing [187; 188], model internal robustness augmentation with adversarial training [96; 38] and more flexible model prediction procedures [44; 43]. Here, we are putting our focus on the second type of adversarial training which is based on the most straightforward model prediction procedure with one single prediction phase.

In this way, we are discussing the single model, single phase prediction robustness. As a result, under strong enough adaptive attacks [189; 94]. Generally speaking, robust models are generated by the technique of adversarial training. Based on their design principles and actual efficiency, adversarial training has evolved for three generations. The first generation tries to insert adversarial examples into the training dataset directly [36]. This proves to be efficient for single-step FGSM attacks but vulnerable to multi-step attack methods. As a result, the second generation adversarial training induces the iterative attacks for adversarial training sample generation [96]. In addition, the adversarial examples are updated in training epochs for data batches and intermediate models. These days, researchers put more attention on the empirical trade-off between adversarial robustness and natural robustness [190; 191]. This leads to the state-of-the-art regularization based adversarial training method that is seeking smoother decision boundaries for a more balanced performance between adversarial robustness and natural robustness [38].

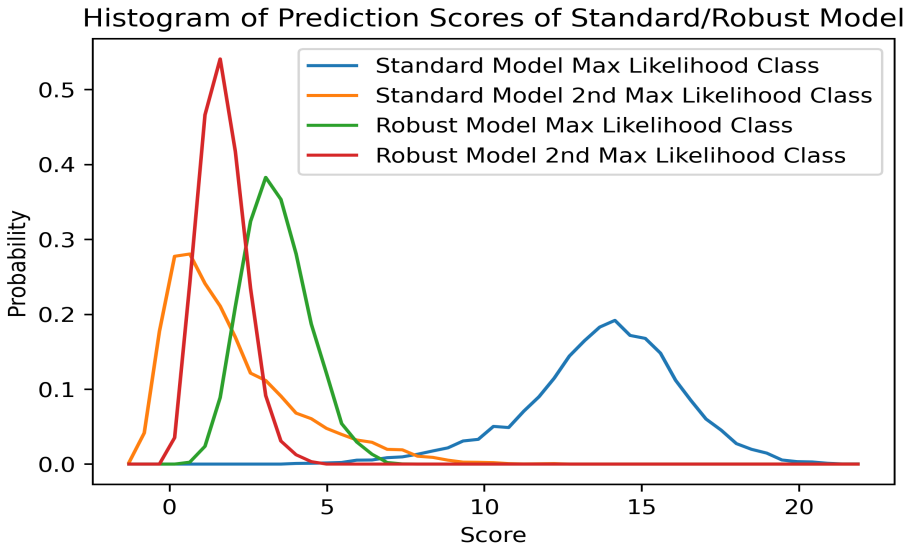

Figure 9.3: Comparison of prediction scores for 1st and 2nd most likely classes between a standard WideResNet model and a robust model based on WideResNet architecture. A neural network classifier would compute likelihood scores for classes and the class with the highest score would be the classification output.

As mentioned above, there have been many adversarial training methods developed so far [186] that are

guided by diverse design ideologies. As past research works have shown the error amplification of neural network models through their layers under adversarial attacks [187; 192], researchers are seeking defense in the opposite way. It has been shown in both theory and practice that balanced robust models should be achievable using methods that can impose tightened local Lipschitzness [193] in layers and augmenting them with generalization techniques like dropout [194]. Empirically, current robust models have shown the feature of being non-expansive [179] in prediction outputs. That is, given a bound-limited input variation, the output magnitude changes from robust models are also bound-limited. It is worth pointing out that even though this is hard to be formally proved, in practice the amplitude control effect can be visualized through empirical experiments as shown in Figure 9.3 and Figure 9.4.

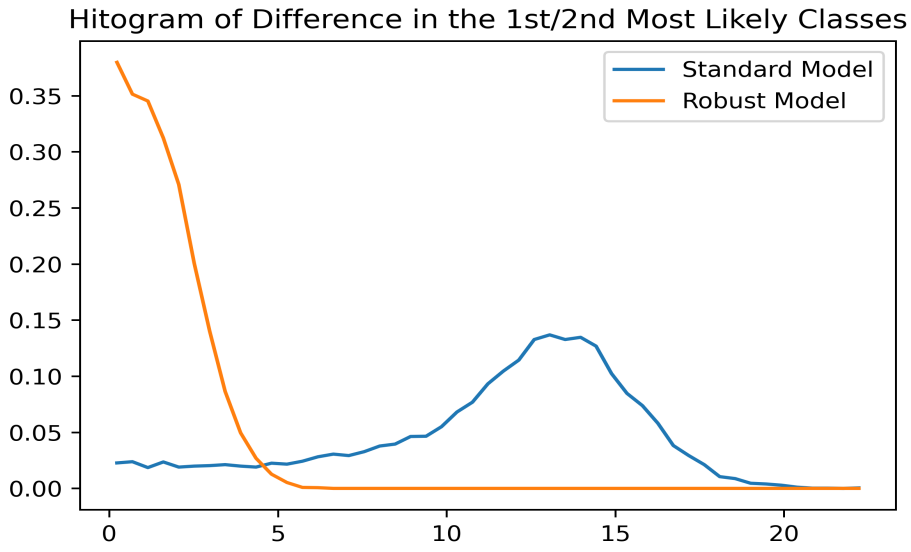

Figure 9.4: Prediction score differences between 1st and 2nd most likely classes of a standard WideResNet model and a robust model based on WideResNet architecture. This difference reflects the distance from the data point to its nearest decision boundary.

Here the robust model [195] is based on the same network architecture as the standard model. As seen from the Figure 9.3, the prediction scores for the maximum and the second maximum likelihood classes are more separated for the standard model but the scores are smoothed for robust models. Figure 9.4 shows the prediction score differences between 1st and 2nd most likely classes on natural and robust models. This difference reflects the distance from the data point to its nearest decision boundary. This further illustrates the trade-off between the natural robustness (prediction confidence maximization) and adversarial robustness (prediction confidence smoothening) achieved by adversarial training techniques [38].

Based on these insights, we illustrate the impact of quantization on robust models. The theoretical analysis and proof is provided in the next paragraphs. We make use of general characteristics of current defensive

robust models and further discuss the impact of quantization operations on them. It is worth mentioning that there has been research work that already made trials on the impact of quantization operations on robust models[196], their analysis does not show a clear definition of robust models.

As part of our approach, we present two propositions for which the theoretical proofs are provided below. The first provides the non-expansive characteristic of current robust neural network models. Further, based on this general characteristic, we prove that this robustness property can be preserved under quantization operations. This becomes the theoretical basis of our defense framework.

Proposition 1 (Robust Models are Non-expansive) *Robust models can suppress the amplification effects of deep networks by controlling the neural network's Lipschitz constant.*

For a function $f : X \to Y$, if it satisfies:

$$D_Y\left(f\left(x_1\right), f\left(x_2\right)\right) \leq \mathrm{Lip}(f) D_X\left(x_1, x_2\right), \forall x_1, x_2 \in X \tag{9.1}$$

for a real-valued $k \geq 0$ and some metrics D_X and D_Y , then we call f Lipschitz continuous and $\mathrm{Lip}(f)$ is defined as the Lipschitz constant of f. The Lipschitz constant describes how much the function output would change given a certain input variation and therefore can be used as a metric for robustness under bound-limited perturbations.

Without loss of generosity, a feed-forward network is composed of a series of functions:

$$f(x) = \left(\phi_l \circ \phi_{l-1} \circ \ldots \circ \phi_1\right)(x) \tag{9.2}$$

where ϕ_l denotes the function of one layer. In this way, the function of the ith layer has its corresponding Lipschitz constant of $\mathrm{Lip}\left(\phi_i\right)$. For the input and output of this specific layer, we have the following property:

For a neural network layer computation function $\phi_i : X_i \to Y_i$, it would satisfy:

$$D_Y\left(\phi_i\left(x_i\right), \phi_i\left(x_i + \Delta x_i\right)\right) \leq \mathrm{Lip}(\phi_i) D_X\left(x_i, x_i + \Delta x_i\right), \forall x_i, x_i + \Delta x_i \in X_i \tag{9.3}$$

Here Δx_i denotes the perturbation caused by the adversarial input. For the very first layer, this variation equals to the input data adversarial perturbation, which means $\Delta x_i = \eta$. As a result, the overall Lipschitz constant of the whole feed-forward network is the product of its L individual layer Lipschitz constants $\mathrm{Lip}\left(\phi_i\right)$.

$$\mathrm{Lip}(f) \leq \prod_{i=1}^{L} \mathrm{Lip}\left(\phi_i\right) \tag{9.4}$$

We focus on the difference between clean inputs $x_1 = X$ and its corresponding adversarial inputs $x_2 = X_{adv}$. From this point of view, we can regard a robust model as a serial combination of layer functions with suppressed Lipchitz constants so that the total amplification under bound-limited adversarial inputs can be controlled. We can imagine the most ideal case [179] when Lipchitz constants of all layers are strictly compressed ($\text{Lip}(\phi_i) \leq 1.0$) so that the adversarial robustness can be fully guaranteed.

Proposition 2 (Quantization Preserves Adversarial Robustness) *Given the robust neural network model formulation shown in Proposition 1, post-training quantization (PTQ) of this robust model will generate a robust quantized model with a high probability of* $\left(1 - \frac{1}{R^2}\frac{\Delta^{2L}}{12^L}\right)$*, where* $R > 1$*: a constant of prediction score ratio between two most likely classes,* Δ*: quantization step size, and L:number of individual layers.*

Model quantizations seek fewer representation bit-widths for model parameters θ like weights and activations. We can use a linear range mapping function as shown in Equation 9.5 to quantize the floating point parameters θ_f into fixed-width N_{bits} integer parameters θ_q.

$$\theta_q = \text{ round } \left\{ \left(\theta_f - min_{\theta_f}\right) \frac{2^{N_{bits}} - 1}{max_{\theta_f} - min_{\theta_f}} \right\} \tag{9.5}$$

As the neural network layer computations are dominated by the multiply-accumulate (MAC) operations on layer inputs, the output variation of layer i caused by the quantization step above can be bounded by a Uniform Distribution determined by the rounding step as shown:

$$Lip\left(\phi_i'\right) \sim U(Lip\left(\phi_i\right) - \frac{\Delta}{2}, Lip\left(\phi_i\right) + \frac{\Delta}{2}) \tag{9.6}$$

Here Δ refers to the quantization step size. In linear quantization this is determined by the number of representation bits used $\Delta = \frac{1}{2^{N_{bits}-1}}$. We denote the original Lipschitz constant for the $i'th$ layer in the network before quantization as $\mu_i = Lip\left(\phi_i\right)$.

To estimate the function amplification on prediction output class change, we investigate the Lipschitz constant change of prediction scores between two most probable classes R_i for the single $i'th$ layer in the network before ($Lip\left(\phi_i\right)$) and after ($Lip\left(\phi_i'\right)$) quantization using the Chebyshev's inequality:

$$\Pr(|Lip\left(\phi_i'\right) - \mu_i| \geq R_i) \leq \frac{\sigma_i^2}{R_i^2} \tag{9.7}$$

Here σ_i^2 refers to the quantization error of layer parameters. In probability, Chebyshev's inequality guarantees that for a given probability distribution only a certain fraction of values can deviate more than a distance from the mean.

From the quantization equation Equation 9.5, we can see that quantizations incur an inevitable information loss with the value rounding operation. For this linear quantization setting, we can compute the quantization error power as:

$$\sigma^2_{\text{QNoise}} = E\left(e^2\right) = \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \frac{1}{\Delta} e^2 de = \frac{\Delta^2}{12} \tag{9.8}$$

Moreover, this error variance should be the same for layers with same quantization settings so we would have:

$$\sigma_i^2 = \sigma^2_{\text{QNoise}} = \frac{\Delta^2}{12} \tag{9.9}$$

To make the current class prediction score to become invalid (no longer maximum), the perturbation needs to at least deviate the score by the ratio of R so this can be distributed to each layer as $R_i = \sqrt[L]{R}$. Here, the required R is determined by the gap between prediction scores of the maximum and $2nd$ maximum likelihood class but we can guarantee $R > 1$. Further, we get the overall perturbed probability as:

$$Pr[|Lip(f) - Lip(f')| \geq R] \leq P_i^L = \frac{1}{R^2} \frac{\Delta^{2L}}{12^L} \tag{9.10}$$

Consider the value range of quantization step Δ (for example $\Delta = \frac{1}{2^8-1} = \frac{1}{255}$ for the 8-bit quantized representation), even consider the extreme case of only one layer $L = 1$ and very close prediction scores for the two classes $R \approx 1$, the flipping probability computed from the above equation would be $\frac{1}{255} * \frac{1}{255} * \frac{1}{12}$. Obviously, the possibility of classification change after quantization is strictly compressed if the original model is robust under adversarial attacks.

Thus we have proved that quantization preserves the non-expansiveness of robust models by preserving the overall Lipschitz constant of the network. Furthermore, the flexbile applications of robust models could be a potential technical path towards more robust quantized models for cloud/edge applications.

It is worth pointing out this analysis might also indicate the impact of quantization operations of models in the reverse way. It indicates simply using quantization on non-robust machine learning models would not help improve the adversarial robustness.

Compared to past methods that aim to get a robust quantized model from the extra training, we propose a simpler and more general approach based on the proposition, which can be implemented in a flexible way as shown below.

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \max_{x'} J\big(\boldsymbol{f}_{rob}(\boldsymbol{\theta}), x, \Delta\boldsymbol{x}\big) \\ &\text{subject to:} x' = x + \Delta\boldsymbol{x} \\ &\|\Delta\boldsymbol{x}\|_p < \varepsilon \\ &W_b, A_b \in \boldsymbol{\theta} \end{aligned} \tag{9.11}$$

In other words, instead of training a model from scratch, we quantize a robust model $\boldsymbol{f}_{rob}$ from adversarial training [186]. We consider this relatively simple problem to solve for an optimal quantization setting of weight bitwidth W_b and activation bitwidth A_b to maintain the robustness. Such an approach greatly reduces the search space and computational burden while maintaining model robustness.

Algorithm 10 illustrates the approach, which uses a simple search process to iteratively test potential settings for model weights and activations. We randomly select a batch of data (or full data) for evaluation (line 5) and compare the natural (line 6) and adversarial performance (lines 7-8) for a selected quantization setting (lines 3-4) on this set of data. The selection is made to maximize the adversarial robustness (classification accuracy in this case) while maintaining the natural robustness at an acceptable level (lines 9-11).

Algorithm 10 Robustness-Driven Optimal Model Quantization Setting Search

Require: $\boldsymbol{X}$: data points ; $\boldsymbol{f}_{rob}$: full precision robust predictor; $J(func, x, \varepsilon)$: cost function of model $func$ according to input data x; ε: maximum bound distance allowed to be modified for each feature (like L_∞ constraint); ***quant***: post-training quantization function, *QuantList*: quantization setting list for weight bitwidth W_b and activation bitwidth A_b; $eval(func, x)$:evaluation metric for model $func$ according to input data x.

1: $best \leftarrow 0$,$optBit \leftarrow empty$
2: **while** QuantList **do**
3: $W_b, A_b \leftarrow pop(QuantList)$
4: $\boldsymbol{f}_{robQT} \leftarrow \boldsymbol{quant}(\boldsymbol{f}_{rob}, W_b, A_b)$
5: $\boldsymbol{x} \leftarrow randomBatch(\boldsymbol{X})$
6: $natural \leftarrow eval(\boldsymbol{f}_{robQT}, \boldsymbol{x})$
7: $\boldsymbol{x}_{adv} \leftarrow argmax \nabla J(\boldsymbol{f}_{robQT}, \boldsymbol{x}, \varepsilon)$
8: $current \leftarrow eval(\boldsymbol{f}_{robQT}, \boldsymbol{x}_{adv})$
9: **if** $natural > threshold$ **and** $current > best$ **then**
10: $best \leftarrow current$
11: $optBit \leftarrow [W_b, A_b]$
12: **end if**
13: **end while**
14: **return** $optBit$

In summary, we propose a systematic evaluation framework for quantized neural network models under the threat of universal perturbations. A simple defensive post-training quantization method is then proposed as a defense solution for these kind of attacks.

9.3 Research Evaluation

We evaluate our framework on the widely-used CIFAR10/CIFAR100/SVHN object classification datasets. Our evaluations first illustrate the vulnerability of normal quantized neural networks under UAPs followed by the efficiency of obtaining robust quantized neural networks using post-training quantization methods. We then show the feasibility of deploying robust quantified models at the edge.

9.3.1 Model and Quantization Tool

We specifically consider the quantized model generation technique of post-training quantization (***PTQ***). To implement post-training quantization, we use BrainChip Akida CNN2SNN Toolkit [197],[2] which is a light-weight model quantization toolkit that is compatible with the Keras machine learning framework [170].

We consider three typical computer vision datasets of CIFAR10/CIFAR100/SVHN. CIFAR10 and CIFAR100 datasets [125] are 32x32 colour images in 10/100 object classes, with 6000/600 images per class. There are 50000 training images and 10000 test images. SVHN [198] is a benchmark dataset containing over 600,000 labeled digits cropped from Street View images. We chose the most widely used CIFAR10 object classification dataset from computer vision for a detailed investigation of the adversarial impacts on various settings. To further validate the efficiency of the proposed defense framework, we also conduct experiments on CIFAR100/SVHN datasets. In these datasets, included images are RGB images with 8-bit pixel values ranging from $0-255$. Perturbation magnitudes can be added in this range and a change of $8/255$ leads to approximately 3.1% of value deviation in all pixels. We incorporate different ML model architectures as follows:

- Akida DS-CNN [197] is a MobileNet [149] architecture model with fewer layers. It utilizes depthwise separable convolutions to reduce the model size and complexity. This model is used for CIFAR10 attack evaluation.
- ResNet [112] induces skip connections allowing efficient training of very deep network models. This is used as the natural model for SVHN dataset.
- WideResNet [199] shows state-of-the-art performance on many object classification datasets and from which many robust models are also adversarially trained [186]. This architecture is used for all three CIFAR10/CIFAR100/SVHN datasets.

[2]We thank Brainchip Inc for granting us academic use of codes and models from the Akida toolkit.

9.3.2 UAP Adversarial Attack Results

In this part, we conduct a detailed investigation of the adversarial impacts on the CIFAR10 dataset. Using our universal attack procedure (see Algorithm 9), we show the existence of universal adversarial perturbations across different quantization precision levels. Examples of these kinds of perturbations are shown in Figure 9.5. The attack strengths (L_{inf}) increase from left to right: *original*, 1.6%,3.1%, 4.7%,6.3%, 7.8%,10.2%, 12.5%. We can observe that the perturbations become relatively obvious when the attack strength is higher than 6% level but even on the strongest 10% level we can only observe some texture noise but the original image structural information is still well-preserved.

Figure 9.5: Example Universal Perturbations on CIFAR10 Classes: airplanes, cars, birds, cats. Attack strengths (L_{inf}) increase from left to right: *original*, 1.6%, 3.1%, 4.7%, 6.3%, 7.8%, 10.2%, 12.5%. We can observe some texture noise from images with the strongest perturbations from comparisons with raw images. But they still preserve original data structural information and are easy to recognize by human eyes.

Experimental results under white-box settings are shown in Figure 9.6 and Figure 9.7. From these results we can observe that model quantization does not really provide a predictable guarantee in adversarial robustness improvement. The diversity in the results stemming from different settings shows the significant need for designing a systematic evaluation workflow that we have developed in this research.

We analyze these results under different predictor and attack settings. To ease the notation, we use the abbreviation "wXaY" to indicate the quantization setting with X bits of *weight* and Y bits of *activation.* For comparison purposes, the accuracy of the "ensemble" is defined as the mean of the classification accuracy of multiple models for white-box experiments.

The post-training quantization (PTQ) model prediction results are shown in Figure 9.6 and Figure 9.7.

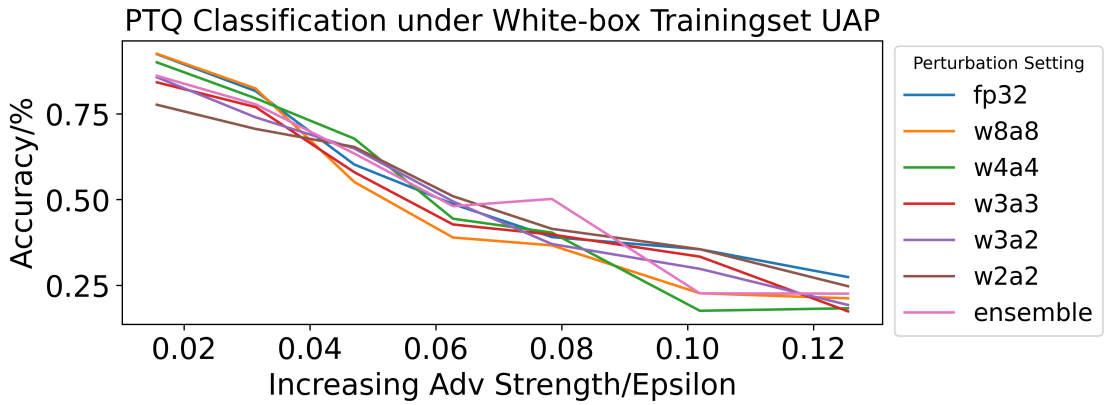

Figure 9.6: Accuracy of PTQ Model under White-box Training Data Attack

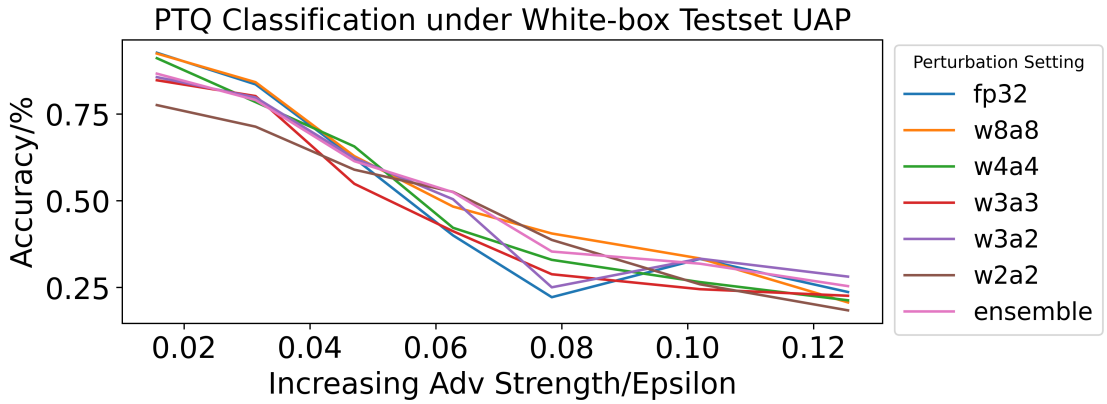

Figure 9.7: Accuracy of PTQ Model under White-box Testing Data Attack

Under white-box settings, the vulnerability expands for both direct testset or indirect trainingset attacks under increasing attack strengths. Quantized models of w2a2, w3a2 and w3a3 show higher accuracy on a wide range of adversarial perturbations. The results also indicate that for the full-precision (fp32) model, the universal perturbation from the trainingset data seems to be more aggressive than the testset data.

9.3.3 Defensive Post-training Quantization Results

We further investigate proposed defense settings on more datasets. We show the impacts of perturbations generated from white-box settings. We then show the efficiency of defending against such kinds of adversarial perturbations using post-training quantization of robust models. It is worth pointing out that we focus on defense in trainingset adversarial settings because these attacks are shown to be stronger and more practical in realistic settings.

Fortunately, even though the state-of-the-art defensive adversarial training methods [96] are based mostly on iterative white-box attack settings, the robustness of these models are highly preserved under universal adversarial perturbations. We consider adversarially trained robust models with state-of-the-art perfor-

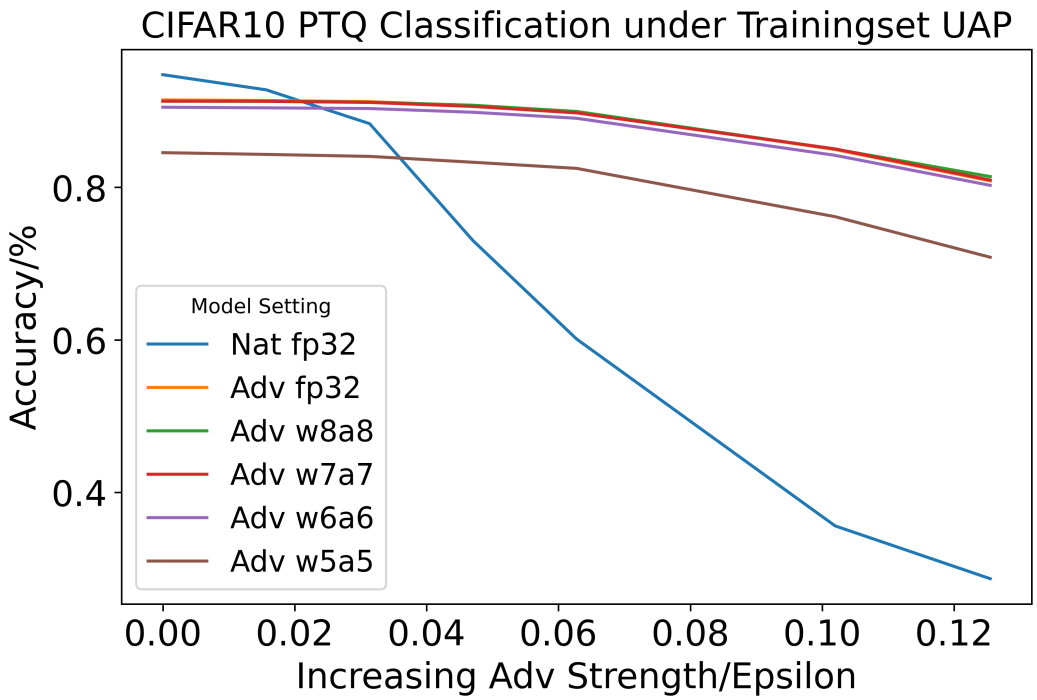

Figure 9.8: Defensive Post-training Quantization on CIFAR10.

mances [186] as follows:

- *CIFAR10:* Pre-trained robust model from research work [200] induces helper perturbations to regularize the decision boundary of the classifier.
- *CIFAR100:* Pre-trained robust model from research work [201] demonstrates how adversarial robustness can benefit from combining larger models, Swish/SiLU activations and model weight averaging..
- *SVHN:* Transfer learning from a pre-trained robust CIFAR10 model from research work [202] demonstrates how adversarial robustness can benefit from semisupervised learning with some extra data.

It is worth pointing out we try to include robust models from different sources. The natural model of CIFAR10 is a standard WideResnet network without adversarial training. The natural model of CIFAR100 is an adversarially-trained model on a different norm distance metric, which shows the incompatibility of robust models from different norm metrics. The robust model of SVHN is based on the transfer learning of a robust model of CIFAR10. We test our proposed defensive quantization approach using these robust models on corresponding datasets. Experimental results of these three models are shown in Figure 9.8, Figure 9.9 and Figure 9.10. We see that the linear quantization of weight and activation values preserve model robustness. Moreover, for these robust image classification models, optimal quantized representations vary across different datasets. On CIFAR10 and CIFAR100, the quantized representation using $optBit =$

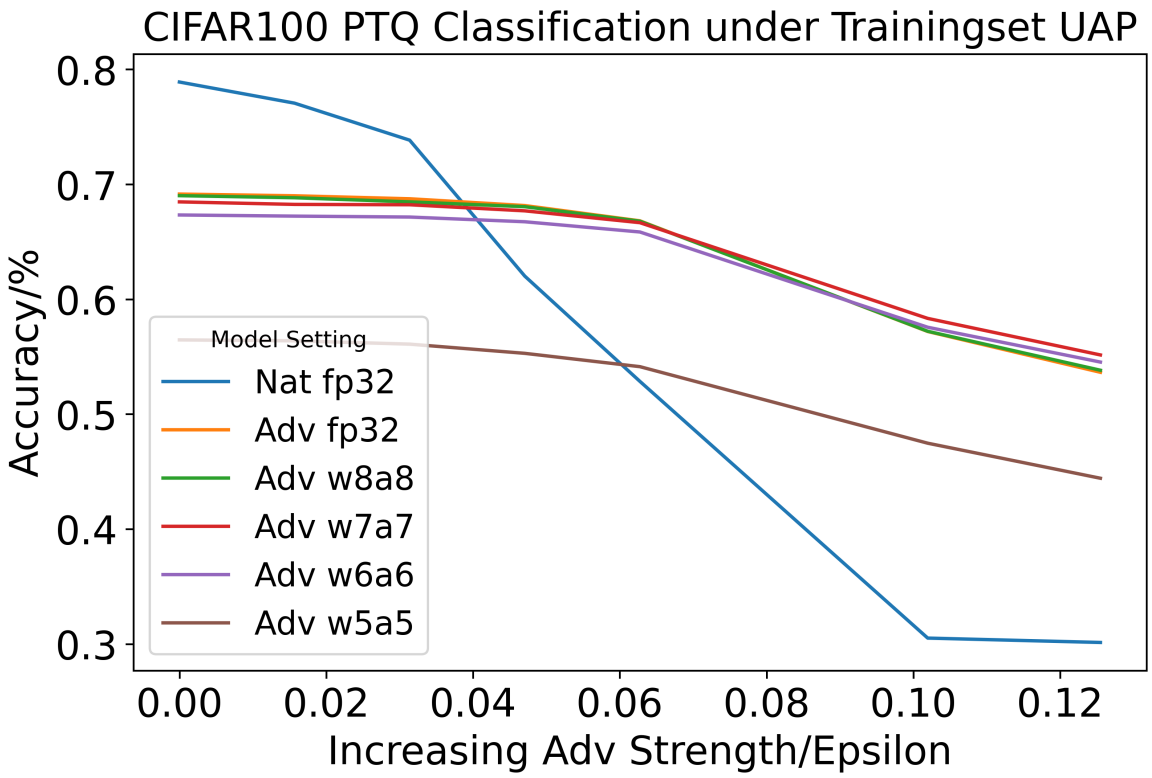

Figure 9.9: Defensive Post-training Quantization on CIFAR100.

$[W_b \leftarrow 7, A_b \leftarrow 7]$ bits in weights and activation shows the best adversarial robustness with the acceptable natural robustness. On SVHN, the quantized representation using $optBit = [W_b \leftarrow 6, A_b \leftarrow 6]$ bits in weights and activation shows the best adversarial robustness with a good natural robustness. These results show the efficiency and scalability of adversarial training with post-training quantization approach proposed in our defensive model deployment framework for cloud/edge environments.

9.3.4 Prototypical Hardware Deployment Validation

The results above show the online accuracy of the defensive quantization settings supported in our framework. The next question is whether they become too difficult to deploy even though they have shown high robustness? To validate the practicality of deploying robust models on the cloud as well as edge, we provide a prototypical hardware deployment of these models on a server GPU and an edge computation platform as follows:

- **Cloud Hardware:** The cloud inference deployment was conducted on a mobile workstation (server) with a 4 Core/8 Thread Intel I7-7700HQ processor, 64GB RAM and a 2560-core NVIDIA P5000 Mobile Pascal GPU with 16GB GDDR5X VRAM.

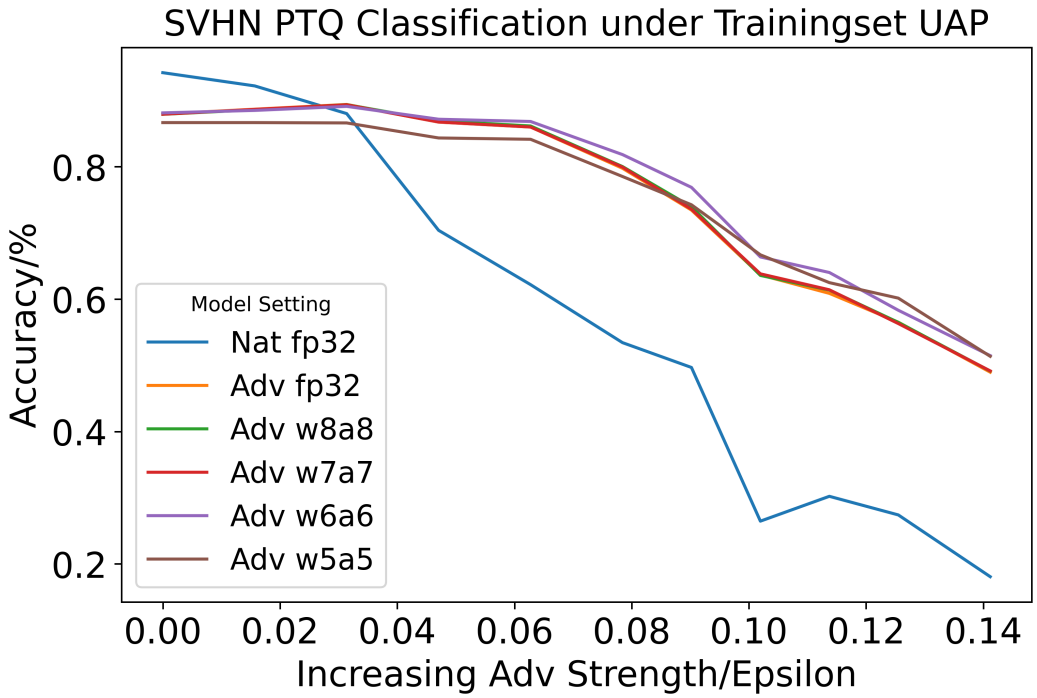

Figure 9.10: Defensive Post-training Quantization on SVHN.

- **Edge Hardware:** The edge inference deployment was conducted on an NVIDIA Jetson Nano Development Board with an ARM A57 Quad-core CPU, 4GB RAM and an embedded 128-core Maxwell GPU.

Even though hardware configurations for cloud and edge deployment are relatively different, the hardware accelerations are all based on NVIDIA GPUs. In this way, some cross-platform ML engines like PyTorch can be reused. We conduct model deployment profiling on a batch inference of 32 data inputs and obtain the average value of 100 executions when measuring the memory usage and the latency. Experimental results are shown in Table 9.1.

Table 9.1: Prototypical Hardware Deployment

Platform	Cloud GPU		Jetson Nano	
Model Setting	WideRes28-10	Rob1(w8a8,w7a7)	WideRes28-10	Rob1_w7a7_FP16
Compute Core	GPU	GPU	GPU	GPU
Model Format	Pytorch	Pytorch	Pytorch	Pytorch
Precision	FP32	FP32	FP32	FP16
Model Size /MB	292.1	292.1	292.1	219.2
#Param	38.12M			
#MACs	257.26G			
Memory Usage /MB	396	436	686	186
Latency/ms	1.4	1.42	12.7	1.19

There still exist some key differences in the prototypical deployment workflows. On the cloud side, we only consider 32-bit full-precision(FP32) models to guarantee the completeness of model performance. Here, the quantized robust model settings shown on the cloud side are based on the 'fake' quantization simulation just as quantization-aware training does. In this way, model parameters are still stored as 32-bit floating point values that are rounded to the corresponding quantization value range. That is the reason why the model size of standard 'WidRes28-10' and quantized robust models are the same on the cloud side. Since we are deploying the same model on the cloud and edge side, they share the same number of model parameters leading to the same level of computations (MACs).

It is worth pointing out that computations with quantization settings are strictly limited by the underlying hardware support. For the cloud side P5000 Pascal GPU, it has internal precision computation support of FP32, FP16 and as low as INT8 (8-bit integer). But INT8 computations are based on the TensorRT optimization framework [203] developed by NVIDIA and are not supported by all devices. For the edge side Jetson Nano, it only has FP32 and FP16 support given its hardware architecture. Thus, better quantization support has become a demanding trend for new edge devices. Newer devices like Google Coral Stick or Board have incorporated INT8 computations. Computation devices like FPGAs (field programmble gate array) have the most flexible bit-width support but they also need deep knowledge and expertise in FPGA deployment.

For the relatively flexible quantization settings like 7-bit or 6-bit that we consider, we store models in higher precisions (FP32,FP16,INT8) and then round these parameter values to lower bitwidth range for equivalent computations. Even though Jetson Nano devices do not have INT8 hardware support, their half-float FP16 represention are shown to be efficient. According to the results in Table 9.1, the inference latency of FP16 model already reaches the inference latency level of the full precision model from the cloud server side. This means that the deployment of robust models either on cloud or edge would not incur extra cost. We also leverage the memory usage of the model inference, where on the cloud side this memory usage is measured by the peak VRAM usage in the GPU. On the edge side, Jetson Nano has a unified shared memory for CPU and GPU so the memory usage on the edge belongs to both RAM and VRAM. This explains why the same 32-bit full-precision model requires more memory usage on the edge side.

In summary, we conducted a prototypical model deployment workflow to show the applicability of the model on both cloud and edge devices as shown in Figure 9.2. We show the potential for deploying robust models in both these kinds of application environments. We choose popular commercial hardware devices trying to show a prototype of practical model deployment workflow. In contrast, the lack of good hardware support also limits the capability of deploying these models. Moreover, it can be anticipated that the high demand for reliable model deployment on the edge side will leverage a large market in more efficient edge hardware accelerators and more user-friendly software tools [173].

9.4 Conclusions

This chapter presented a reusable and configurable LEC deployment framework that transforms a user-supplied deep learning machine learning (ML) model into an adversarially robust quantized model (offline phase) that is amenable to be deployed and operated in the heterogeneous and resource-constrained edge (online phase). The middleware's offline phase supports a systematic adversarial robustness evaluation workflow for quantized models under universal adversarial perturbations (UAPs). Experimental evaluations reveal that traditional quantized models are vulnerable to universal perturbations and moreover, the perturbations from some quantized models are even transferable. The work highlights the need for state-of-art adversarial training and validates its efficiency. Further, this work is the first to propose the generation procedure of quantized robust models combining model evaluation and quantization using adversarially trained models. Such adversarially trained quantized neural networks continue to be compact and efficient making them useful in realizing edge-based services. These together bridge the gap between robust ML models and efficient hardware-aware deployment in cloud/edge environments.

Our future work will address current limitations in the presented work as follows: (1) Even though we have shown an efficient method to obtain robust quantized neural network models, the post-training compression methods cannot generate models for aggressive quantization settings like binary neural networks for mobile or tiny devices. Thus, our current work provides only a partial solution comprising relatively conservative quantization settings. Our future work will therefore explore more systematic quantization-aware adversarial training for more flexible model settings [204]; (2) The deployable quantized models in the current work are generated offline and hence cannot be adapted dynamically as operating conditions may change. Accordingly, we will investigate dynamic adaptations extending to the further decision making phase; (3) Distributed model inferencing is increasingly prevalent in edge scenarios and hence there is a need for distributed, interacting quantized models with potentially different precision levels; (4) We only put attention on the computer vision task of object image classification. This only occupies a very small fraction of different potential application scenarios in CPS. We may need to further investigate classifiers for different applications and other types of models beyond classifiers; (5) An emerging trend of LEC is the development of large foundation models [205] that are very difficult to train from scratch. The deployment of this kind model might lead to more general adversarial risks in both model optimization and deployment phases; and (6) Investigating the use of machine learning compiling and optimization tools like TVM (which is available in open-source) for generating more flexible robust quantized models remains an open problem.

CHAPTER 10

Conclusion

10.1 Discussion

Our work investigates the problem of how to evaluate and improve the adversarial robustness of learning-enabled components(LECs) in cyber-physical system applications. We show the potential risk of adversarial impacts for several application scenarios from both software algorithm and hardware deployment levels. We propose multiple resilient solutions to address firstly, the evaluation difficulty and, then the vulnerabilities of the practical learning-based component deployment.

To wrap up, we again list out the challenges we are facing:

- Challenge 1: Define threat on system resilience for CPS ML application
- Challenge 2: Evaluate the resilience of ML deployment in cyber-physical systems
- Challenge 3: Adversarial impacts be mitigation in a resilient way
- Challenge 4: Learning deployment hardware impact the system resilience

Also, the tasks we have conducted or proposed to help solve these challenges in the Table 10.1 below.

The central topic of this thesis is "how to deploy machine learning predictors resiliently in adversarial environment". Even though we can claim this is one of the most generally seeking goal for all system designer, this is still an open question that often requires taking into account of semantic and domain-specific or even case-specific knowledge.

In this thesis, we first consider the scenario of smart grid load prediction and develop a dynamic data repair framework to address adversarial vulnerabilities from partial compromise of sensor network data. Our research defines and explores adversarial threats on a sensor network where a portion of sensor reading values are adversarially modified. Based on this setting, we prove the vulnerability of machine learning predictor even under this limited perturbation. Moreover, with the assumption that the attacker has knowledge of the anomaly detector in the system, we further prove stealthy attacks taking into account of anomaly detectors (auto-encoder based identity mapping type) are also possible without much barrier. To mitigate such potential threats, randomization techniques are induced for adversary detection, we reuse the identity mapping model of auto-encoder not only for anomaly detection but also for potentially adversarial data recontruction. Taking into account of the DDDAS mechanism, we make use of feed-back data repair for incoming data in an iterative manner.

Table 10.1: Contribution Summary

Task	Scope	Target Challenge	Main Technical Path
1	Power system resilience evaluation	1,2	Model-driven, Domain-specific abstraction
2	Load forecast robustness improvement	3	Dynamic data repair
3	Adverary in PHM application	1,2,3	Adversarial classificationregression, Semantic structure
4	Hardware accelerator evaluation	2,4	Hardware acceleration, Reliability-driven anlysis
5	TinyML Patterns for CPS	2,4	TinyML, Edge Computing
6	FPGA BNN Black-box Adversarial Attack	1,4	Hardware acceleration, Adversarial Attack
7	UAPs on compressed quanitzed models	1,3,4	Model Quantization, Universal Adversarial Perturbation

We further study the adversarial robustness threat in prognostics and health management systems and investigate whether incorporating semantic knowledge into LEC deployments would help overcome this kind of threat. To the best of our knowledge, our work should be the first to conduct a systematic investigation on the vulnerability of deep learning data-driven prognostics. We show both the classification (health status prediction) and the regression (remaining useful life prediction) can be greatly affected by limited adversarial perturbations. Meanwhile, more resilient model deployments with context-information can reduce the adversarial impacts. This in turn shows that domain-specific knowledge is effective for adversarial robustness improvement.

As all LEC deployments eventually have to be placed on certain hardware platforms, then we put attention on the hardware availability of LEC executions on various cloud/edge scenarios. Several guideline principles for efficient LEC deployments on edge hardware platforms ranging from CPU,GPU, FPGA, ASIC and even tiny MCU are revealed and proposed. As in edge computing, different accelerator technologies can illustrate different traits for different application types that run at the edge, there is a critical need for effective mechanisms that can help developers select the right technology (or a mix of) to use in their context, which is currently lacking. To address this critical need, we propose a recommender system to help users rapidly and cost-effectively select the right hardware accelerator technology for a given compute-intensive task. Our framework comprises the following workflow. First, we collect realistic execution traces of computations on real, single hardware accelerator devices. Second, we utilize these traces to deduce the achievable latencies and amortized costs for device deployments across the cloud-edge spectrum, which in turn provides guidance in selecting the right hardware. Apart from these hardware options, a rapid developing platform is microcontroller(MCU). A corresponding LEC deployment technique which has aroused more attention these days

is called TinyML. Unfortunately, pure MCUs lack the capacity to conduct complex analytics with heterogeneous data sources in CPS. To address these limitations, we propose the solution of using TinyML deployed at the edge in cooperation with system-level machine learning executing in the cloud. Specifically, we study applications in which sensor data is collected and used to predict system health status and perform remaining useful life regression. We also show how edge MCU devices and cloud computing can be combined and adapted to satisfy diverse requirements, such as latency, power and communication. We also describe the limitations of the current MCU-based deep learning in data-driven prognostics. To the best of our knowledge, this is the first work to systematically investigate the TinyML-Cloud cooperation for data-driven prognostics. We target this as a vision paper and aim to provide a high-level guideline for future PHM application designs involving smart MCU-based decision making.

Finally, we propose a robustness-driven model deployment workflow mitigating the adversarial impacts across these hardware platforms. Inspired by a simple black-box adversarial attack with/on binary nerual networks(BNN) on FPGAs after the hardware availability work, we investigate whether quantized models with different precision levels are vulnerable to the same universal adversarial perturbation (UAP). We then present a workflow that generates adversarially robust compressed models using a novel defensive post-training quantization approach and deploys them at the edge. Experimental evaluations reveal that although quantized models are vulnerable to universal perturbations, post-training quantization on middleware-synthesized adversarially trained models are effective against UAPs. Furthermore, deployments on heterogeneous edge devices with flexible quantization settings are efficient thereby paving the way in realizing adversarially robust data-driven cloud/edge services.

10.2 Future Work

Based on the accomplished research works, we believe there are two potential directions that may play important roles in robust deployment of LECs. On one hand, with more research conducted on adversarial attack and defense we would expect more automated procedures in both attack evaluation and defensive deployments. On the other hand, we have to admit the vulnerability of black-box deep neural network models may remain inevitable in the near future, this in turn means maybe we can actually make use of adversarial examples in some occasions. As a result, we discuss two potential future directions in the following paragraphs.

10.2.1 Automated Robustness-Driven Model Evaluation/Optimization

The results in attack exploration and defense mechanism design show the gap in the performance between the natural model and protected model. Deep Learning (DL) techniques are widely used in a number of classification and regression problem settings. However, it is often the case that the complexity of these DL models

force users to treat the DL models as black boxes making them vulnerable to adversarial attacks.Although many attack and defense strategies have been proposed over the years, the diversity of the solution space makes it challenging for users in practice to rapidly test the robustness of a trained DL model under adversarial settings [206]. To overcome this problem, we propose exploiting model-driven generative techniques to construct an automated adversarial robustness testing and model strengthening framework for DL models. To that end we have defined a multi-stage approach comprising the following steps, which are described in subsequent sections:

1. **Basic predictor training/testing**: Existence of a DL model is a necessary condition. If for some reason there is no pre-trained model and data uploaded from the user, we provide predictor training and evaluation pipelines for demonstration or further use.

2. **Adversarial attack on the prediction model**: This step provides the user with worst-case white-box attack settings to enable testing the robustness of the predictor. This attack investigates one single attack configuration that could be defined by the user.

3. **Empirical Robustness Testing**: This step is effectively a batch testing equivalent of the previous phase. It investigates the model robustness on a series of increasing attack strength and returns empirical error rates and loss sensitivity as output artifacts.

4. **Defense Policy Search**: This step automatically conducts a grid search on the input space of defenses. Two types of defense policies are considered. The search needs to determine the optimal hyper-parameter values in these defense policies.

5. **Deployment of Defense Policy**: In this step the framework applies a chosen defense policy for a complete inference and allows the user to test the efficiency of the output defense policy on more data and more flexible settings.

10.2.2 Utilizing Adversarial Examples for GOOD

Another direction is the utilization of adversarial attacks from the reverse point of view. That is, given the current inevitability of adversary in ML, is it reasonable to make use of adversarial examples for good?

With the rapid development of edge computing and machine learning, edge-based solutions for video streaming analytics have emerged as a popular system paradigm for video-centric applications [207]. Compared to fully centralized solutions of pushing all computation burdens to the cloud, such edge-assisted systems offer tremendous benefits for more flexible deployment.

Meanwhile, video streaming in scenarios like smart cities may bring inevitable privacy concerns [208]. In an application like smart city neighborhood analysis, we can make use of several cameras in the neighborhood to make analysis on vehicles and people and give suggestions on safety improvement. However, a neighborhood resident may feel uncomfortable if video contents including his/her facial/clothing details or vehicle routing details, are constantly streamed to a centralized platform performing the video analysis that may reveal private habit or internal conditions.

As a result, we can get an obvious conflict of interest from the privacy perspective. On one hand, clients hope they can make use of some video analysis services to improve daily life. On the other hand, the potential privacy leakage problem cannot be neglected. Essentially, this becomes a question of how much trust the end client user should put on the overall service provider.

It is worth pointing out that privacy has always been of significant concern in the field of edge computing [209]. However, current research on edge privacy mainly assumes attacker from the outside try to intercept the data from the communication between the edge and the cloud. This kind of threat has been mostly solved by stronger data encryption guarantee which prevents the attacker from getting access to the data [210]. But this still assumes the complete trust on the cloud side service provider who may still abuse the information in some way.

Assuming the existence of adversarial attacker from the cloud side, it is reasonable to assume that the attacker would also have enough computation capability to run video analysis machine learning algorithms to extract information from the large volume of video data in practical applications at a high speed. And most intuitively, the edge can choose to give zero trust on the cloud side [211] and every data request would require verification and authentication from the edge client user. But as we have pointed out the data volumes from state-of-art video analysis tasks could be huge this model needs too much manual work in practice. To seek a trade-off between full and zero trust, the edge can choose to give the cloud side limited trust and only provide the cloud with data whose sensitive information part has been filtered [212; 213; 214]. In this way, the cloud side could make use of the information provided by the edge side without being able to revealing all sensitive information.

Based on all these discussions above, we believe the privacy issue in edge-assisted video analysis is still a novel field worth diving into based on the following challenges:

- Only the edge side is in full control of the end client user and the client may not have full trust on the center service provider
- Even though we are talking about privacy all the time, what is the exact metric for privacy in edge-assisted settings?

- To improve privacy, the edge side tries to hold and analyze data on edge without sending data to cloud. How can efficient analysis in such resource constrained environments be guaranteed?
- The attacker may run video analysis machine learning algorithms to extract information from video streams for large-scale abuse.

This overall would formulate a privacy-preserving problem in a general edge setting. Overall, we are facing a problem with a big picture like this: given a limited set of hardware devices and corresponding prediction tasks, we need to design a strategy to decide when and how we should make use of machine learning models to analyze data to minimize potential privacy leakage risks.

One interesting problem we first need to specify is how to define the privacy. For a traffic video stream with people and vehicles moving around from a camera, the privacy can be hierarchical. For one video frame, privacy refers to whether people and vehicles exist and what attributes they have. High-level attributes like color or make can only be used for identification in a rough manner. Low-level detailed attributes like license plate or face would enable accurate vehicle or person re-identification. Furthermore, for a traffic video stream, vehicles and people are coming in and going out continuously, which means temporal relations matter for these objects in the frame.

Without loss of generality, we provide the definition of privacy of a video stream from three levels:

- Tier0-Privacy: the most sensitive and accurate information about an object that has unique identification property. An example would be a car with license plate 'XXXX' past the road at the last minute.
- Tier1-Privacy: objects with certain attributes appear in a certain time slot. An example would be a white Nissan car went past the road at the last 15min time slot.
- Tier2-Privacy: high-level condition for the target road segment. An example would be 10 cars went past the road in the last hour.

We can see that this division is based on the granularity and uniqueness of object attributes. And of course, more accurate and unique information means more privacy information within.

A novel privacy-preserving data processing technique is based on adversarial attacks. Without breaking the original information structure hidden in the images, adding a bound-limited adversarial perturbation to an image could prevent the abuse of information by machine learning models from the attacker in the data distribution [215; 216; 217].

Based on the discussion above, we set the following guidelines for privacy-preserving data migration in the deployment of machine learning models in an edge-based video analysis system:

- Full raw high-resolution images would only be sent to the cloud if authorized by the edge user.
- For the performance checking, randomization in data processing should be considered to break the temporal dependencies.
- Adversarial perturbations would be added to the relative low-resolution images sent to the cloud to stop the cloud side to make further use of the image to do license plate recognition or face recognition.

We believe this adversary-based privacy protection would become a powerful tool to enhance information security in a more general sense.

BIBLIOGRAPHY

[1] E. Lee, "The past, present and future of cyber-physical systems: A focus on models," *Sensors*, vol. 15, no. 3, pp. 4837–4869, 2015.

[2] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *2008 The 28th International Conference on Distributed Computing Systems Workshops*, pp. 495–500, IEEE, 2008.

[3] S.-T. Chen, C. Cornelius, J. Martin, and D. H. P. Chau, "Shapeshifter: Robust physical adversarial attack on faster r-cnn object detector," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 52–68, Springer, 2018.

[4] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1625–1634, 2018.

[5] M. Valtierra-Rodriguez, R. de Jesus Romero-Troncoso, R. A. Osornio-Rios, and A. Garcia-Perez, "Detection and classification of single and combined power quality disturbances using neural networks," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 5, pp. 2473–2482, 2013.

[6] A. Ghafouri, Y. Vorobeychik, and X. Koutsoukos, "Adversarial regression for detecting attacks in cyber-physical systems," *arXiv preprint arXiv:1804.11022*, pp. 3769–3775, 2018.

[7] Y. Vorobeychik and M. Kantarcioglu, "Adversarial machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–169, 2018.

[8] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 506–519, ACM, 2017.

[9] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," *arXiv preprint arXiv:1804.08598*, 2018.

[10] Y. Chen, Y. Tan, and B. Zhang, "Exploiting vulnerabilities of load forecasting through adversarial attacks," in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, pp. 1–11, ACM, 2019.

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

[12] D. Zhang, X. Han, and C. Deng, "Review on the research and practice of deep learning and reinforcement learning in smart grids," *CSEE Journal of Power and Energy Systems*, vol. 4, no. 3, pp. 362–370, 2018.

[13] R. Sevlian and R. Rajagopal, "A scaling law for short term load forecasting on varying levels of aggregation," *International Journal of Electrical Power & Energy Systems*, vol. 98, pp. 350–361, 2018.

[14] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on lstm recurrent neural network," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2017.

[15] H. Shi, M. Xu, and R. Li, "Deep learning for household load forecasting—a novel pooling deep rnn," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 5271–5280, 2017.

[16] W. He, "Load forecasting via deep neural networks," *Procedia Computer Science*, vol. 122, pp. 308–314, 2017.

[17] C. Tong, J. Li, C. Lang, F. Kong, J. Niu, and J. J. Rodrigues, "An efficient deep model for day-ahead electricity load forecasting with stacked denoising auto-encoders," *Journal of parallel and distributed computing*, vol. 117, pp. 267–273, 2018.

[18] M. Mishra, "Power quality disturbance detection and classification using signal processing and soft computing techniques: A comprehensive review," *International Transactions on Electrical Energy Systems*, vol. 29, no. 8, p. e12008, 2019.

[19] S. Wang and H. Chen, "A novel deep learning method for the classification of power quality disturbances using deep convolutional neural network," *Applied energy*, vol. 235, pp. 1126–1140, 2019.

[20] E. Balouji and O. Salor, "Classification of power quality events using deep learning on event images," in *2017 3rd International Conference on Pattern Recognition and Image Analysis (IPRIA)*, pp. 216–221, IEEE, 2017.

[21] J.-G. Kim and B. Lee, "Appliance classification by power signal analysis based on multi-feature combination multi-layer lstm," *Energies*, vol. 12, no. 14, p. 2804, 2019.

[22] N. Mohan, K. Soman, and R. Vinayakumar, "Deep power: Deep learning architectures for power quality disturbances classification," in *2017 International Conference on Technological Advancements in Power and Energy (TAP Energy)*, pp. 1–6, IEEE, 2017.

[23] A. Saxena, J. Celaya, E. Balaban, K. Goebel, B. Saha, S. Saha, and M. Schwabacher, "Metrics for evaluating performance of prognostic techniques," in *2008 International Conference on Prognostics and Health Management*, pp. 1–17, IEEE, 2008.

[24] J. F. Broenink, "Introduction to physical systems modelling with bond graphs," *SiE Whitebook on Simulation Methodologies*, 1990.

[25] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *2017 IEEE international conference on prognostics and health management (ICPHM)*, pp. 88–95, IEEE, 2017.

[26] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Reliability Engineering & System Safety*, vol. 183, pp. 240–251, 2019.

[27] J. Wang, G. Wen, S. Yang, and Y. Liu, "Remaining useful life estimation in prognostics using deep bidirectional lstm neural network," in *2018 Prognostics and System Health Management Conference (PHM-Chongqing)*, pp. 1037–1042, IEEE, 2018.

[28] M. A. Chao, C. Kulkarni, K. Goebel, and O. Fink, "Fusing physics-based and deep learning models for prognostics," *arXiv preprint arXiv:2003.00732*, 2020.

[29] G. Karsai and J. Sztipanovits, "Model-integrated development of cyber-physical systems," in *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*, pp. 46–54, Springer, 2008.

[30] X. Koutsoukos, G. Karsai, A. Laszka, H. Neema, B. Potteiger, P. Volgyesi, Y. Vorobeychik, and J. Sztipanovits, "Sure: A modeling and simulation integration platform for evaluation of secure and resilient cyber–physical systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 93–112, 2017.

[31] H. Neema, P. Volgyesi, B. Potteiger, W. Emfinger, X. Koutsoukos, G. Karsai, Y. Vorobeychik, and J. Sztipanovits, "Sure: An experimentation and evaluation testbed for cps security and resilience: Demo abstract," in *Proceedings of the 7th International Conference on Cyber-Physical Systems*, pp. 1–1, IEEE Press, 2016.

[32] B. Broll and J. Whitaker, "Deepforge: An open source, collaborative environment for reproducible deep learning," 2017.

[33] Z. Lattmann, T. Kecskés, P. Meijer, G. Karsai, P. Völgyesi, and Á. Lédeczi, "Abstractions for modeling complex systems," in *International Symposium on Leveraging Applications of Formal Methods*, pp. 68–79, Springer, 2016.

[34] 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.

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

[36] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples (2014)," *arXiv preprint arXiv:1412.6572*, 2014.

[37] A. T. Nguyen and E. Raff, "Adversarial attacks, regression, and numerical stability regularization," *arXiv preprint arXiv:1812.02885*, 2018.

[38] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan, "Theoretically principled trade-off between robustness and accuracy," *arXiv preprint arXiv:1901.08573*, 2019.

[39] A. Shafahi, M. Najibi, M. A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein, "Adversarial training for free!," in *Advances in Neural Information Processing Systems*, pp. 3358–3369, 2019.

[40] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2107–2116, 2017.

[41] V. Zantedeschi, M.-I. Nicolae, and A. Rawat, "Efficient defenses against adversarial attacks," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 39–49, 2017.

[42] S. Wang, X. Wang, P. Zhao, W. Wen, D. Kaeli, P. Chin, and X. Lin, "Defensive dropout for hardening deep neural networks under adversarial attacks," in *Proceedings of the International Conference on Computer-Aided Design*, p. 71, ACM, 2018.

[43] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," *arXiv preprint arXiv:1711.01991*, 2017.

[44] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh, "Towards robust neural networks via random self-ensemble," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 369–385, 2018.

[45] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[46] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[47] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[49] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[50] C. Khona, "Key Attributes of an Intelligent IIoT Edge Platform." http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/, 2017. [Online; accessed 19-July-2019].

[51] M. Najafi, K. Zhang, M. Sadoghi, and H.-A. Jacobsen, "Hardware acceleration landscape for distributed real-time analytics: Virtues and limitations," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1938–1948, IEEE, 2017.

[52] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards qos-aware fog service placement," in *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pp. 89–96, IEEE, 2017.

[53] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Profit-aware application placement for integrated fog–cloud computing environments," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 177–190, 2020.

[54] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 751–766, 2018.

[55] J. G. Boubin, N. T. Babu, C. Stewart, J. Chumley, and S. Zhang, "Managing edge resources for fully autonomous aerial systems," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pp. 74–87, 2019.

[56] J. Tang, S. Liu, L. Liu, B. Yu, and W. Shi, "Lopecs: A low-power edge computing system for real-time autonomous driving services," *IEEE Access*, vol. 8, pp. 30467–30479, 2020.

[57] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, "Enabling fpgas in the cloud," in *Proceedings of the 11th ACM Conference on Computing Frontiers*, p. 3, ACM, 2014.

[58] H. Zhu, D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and M. Erez, "Kelp: Qos for accelerated machine learning systems," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 172–184, IEEE, 2019.

[59] M. Isakov, V. Gadepally, K. M. Gettings, and M. A. Kinsy, "Survey of attacks and defenses on edge-deployed neural networks," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, IEEE, 2019.

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

[61] 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 *Proceedings of the 34th Annual Computer Security Applications Conference*, pp. 393–406, 2018.

[62] X. Hu, L. Liang, L. Deng, S. Li, X. Xie, Y. Ji, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "Neural network model extraction attacks in edge devices by hearing architectural hints," *arXiv preprint arXiv:1903.03916*, 2019.

[63] X. Koutsoukos, G. Karsai, A. Laszka, H. Neema, B. Potteiger, P. Volgyesi, Y. Vorobeychik, and J. Sztipanovits, "Sure: A modeling and simulation integration platform for evaluation of secure and resilient cyber–physical systems," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 93–112, 2018.

[64] H. Neema, J. Sztipanovits, M. Burns, and E. Griffor, "C2wt-te: A model-based open platform for integrated simulations of transactive smart grids," in *2016 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pp. 1–6, IEEE, 2016.

[65] A. R. Khan, A. Mahmood, A. Safdar, Z. A. Khan, and N. A. Khan, "Load forecasting, dynamic pricing and DSM in smart grid: A review," *Renewable and Sustainable Energy Reviews*, vol. 54, pp. 1311–1322, 2016.

[66] A. P. Douglas, A. M. Breipohl, F. N. Lee, R. Adapa, and W. B. R. Norman, "Risk Due to Load Forecast Uncertainty in Short Term Power System Planning * School of Electrical and Computer Engineering , University of Oklahoma," *IEEE Transactions on Power Systems*, vol. 13, no. 4, pp. 1493–1499, 1998.

[67] R. Billinton and D. Huang, "Effects of load forecast uncertainty on bulk electric system reliability evaluation," *IEEE Transactions on Power Systems*, vol. 23, no. 2, pp. 418–425, 2008.

[68] O. Kosut, L. Jia, R. J. Thomas, and L. Tong, "Malicious data attacks on smart grid state estimation: Attack strategies and countermeasures," in *2010 First IEEE International Conference on Smart Grid Communications*, pp. 220–225, IEEE, 2010.

[69] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[70] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

[71] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," *arXiv preprint arXiv:1702.04267*, 2017.

[72] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 3–14, ACM, 2017.

[73] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135–147, ACM, 2017.

[74] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," *arXiv preprint arXiv:1805.06605*, 2018.

[75] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Advances in Neural Information Processing Systems*, pp. 7167–7177, 2018.

[76] M. Cui, S. Member, J. Wang, S. Member, and M. Yue, "Machine Learning Based Anomaly Detection for Load Forecasting Under Cyberattacks," *IEEE Transactions on Smart Grid*, vol. PP, no. c, p. 1, 2018.

[77] J. Goh, S. Adepu, M. Tan, and L. Z. Shan, "Anomaly Detection in Cyber Physical Systems using Recurrent Neural Networks," *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, pp. 140–145, 2017.

[78] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," 2017.

[79] D. Li, D. Chen, L. Shi, B. Jin, J. Goh, and S.-K. Ng, "MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks," pp. 1–17, 2019.

[80] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding," 2018.

[81] 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.

[82] K. P. Schneider, Y. Chen, D. P. Chassin, R. G. Pratt, D. W. Engel, and S. E. Thompson, "Modern grid initiative distribution taxonomy final report," tech. rep., Pacific Northwest National Laboratory, 2008.

[83] https://github.com/gridlab-d/Taxonomy_Feeders, 2015. Accessed: Oct. 2019.

[84] E. Blasch, D. Bernstein, and M. Rangaswamy, "Introduction to dynamic data driven applications systems," in *Handbook of Dynamic Data Driven Applications Systems*, pp. 1–25, Springer, 2018.

[85] X. Zhou, Y. Li, C. A. Barreto, J. Li, P. Volgyesi, H. Neema, and X. Koutsoukos, "Evaluating resilience of grid load predictions under stealthy adversarial attacks," in *2019 Resilience Week (RWS)*, vol. 1, pp. 206–212, IEEE, 2019.

[86] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[87] J. Echauz, K. Kenemer, S. Hussein, J. Dhaliwal, S. Shintre, S. Grzonkowski, and A. Gardner, "Adversarial campaign mitigation via roc-centric prognostics," in *Proceedings of the Annual Conference of the PHM Society*, vol. 11, 2019.

[88] A. Saxena and K. Goebel, "C-mapss data set," *NASA Ames Prognostics Data Repository*, 2008.

[89] Y. Li, H. Zhang, C. Bermudez, Y. Chen, B. A. Landman, and Y. Vorobeychik, "Anatomical context protects deep learning from adversarial perturbations in medical imaging," *Neurocomputing*, vol. 379, pp. 370–378, 2020.

[90] E. Ramasso and A. Saxena, "Performance benchmarking and analysis of prognostic methods for cmapss datasets.," *International Journal of Prognostics and Health Management*, vol. 5, no. 2, pp. 1–15, 2014.

[91] J. Zhang and J. Lee, "A review on prognostics and health monitoring of li-ion battery," *Journal of power sources*, vol. 196, no. 15, pp. 6007–6014, 2011.

[92] P. Baraldi, F. Cadini, F. Mangili, and E. Zio, "Model-based and data-driven prognostics under different available information," *Probabilistic Engineering Mechanics*, vol. 32, pp. 66–79, 2013.

[93] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.

[94] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *arXiv preprint arXiv:1902.06705*, 2019.

[95] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[96] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[97] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14410–14430, 2018.

[98] T. Wang, J. Yu, D. Siegel, and J. Lee, "A similarity-based prognostics approach for remaining useful life estimation of engineered systems," in *Prognostics and Health Management, 2008. PHM 2008. International Conference on*, pp. 1–6, IEEE, 2008.

[99] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Reliability Engineering & System Safety*, vol. 183, pp. 240–251, 2019.

[100] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[101] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pp. 88–95, IEEE, 2017.

[102] F. O. Heimes, "Recurrent neural networks for remaining useful life estimation," in *2008 international conference on prognostics and health management*, pp. 1–6, IEEE, 2008.

[103] Umberto Griffo, "Recurrent Neural Networks for Predictive Maintenance." https://github.com/umbertogriffo/Predictive-Maintenance-using-LSTM, 2019. [Online; accessed 19-May-2020].

[104] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine prognostics," *NASA Technical Reports*, 2008.

[105] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," 2018.

[106] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, *et al.*, "Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 269–278, ACM, 2018.

[107] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*, pp. 77–84, IEEE, 2016.

[108] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74, ACM, 2017.

[109] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, IEEE, 2017.

[110] B. Varghese, N. Wang, D. Bermbach, C.-H. Hong, E. de Lara, W. Shi, and C. Stewart, "A survey on edge benchmarking," 2020.

[111] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, *et al.*, "Mlperf inference benchmark," *arXiv preprint arXiv:1911.02549*, 2019.

[112] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[113] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[114] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga?," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 232–239, IEEE, 2012.

[115] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, and J. Zambreno, "An fpga implementation of decision tree classification," in *2007 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1–6, IEEE, 2007.

[116] 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.

[117] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[118] Intel Incorporation, "Intel® Movidius™ Neural Compute Stick." https://software.intel.com/en-us/movidius-ncs, 2018. [Online; accessed 19-Jan-2020].

[119] R. Kastner, J. Matai, and S. Neuendorffer, "Parallel programming for fpgas," *arXiv preprint arXiv:1805.03648*, 2018.

[120] S. Jiang, D. He, C. Yang, C. Xu, G. Luo, Y. Chen, Y. Liu, and J. Jiang, "Accelerating mobile applications at the network edge with software-programmable fpgas," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 55–62, IEEE, 2018.

[121] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang, "Software-hardware codesign for efficient neural network acceleration," *IEEE Micro*, vol. 37, no. 2, pp. 18–25, 2017.

[122] W. Foster, "Leverage Multiple Intel® Neural Compute Sticks with Intel® Distribution of OpenVINO™ Toolkit." https://software.intel.com/en-us/articles/leverage-multiple-intel-neural-compute-sticks-with-intel-distribution-of-openvino-toolkit, 2019. [Online; accessed 20-Feb-2020].

[123] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[124] U.S. Energy Information Administration, "Electric Power Monthly with Data for November 2019." https://www.eia.gov/electricity/monthly/current_month/epm.pdf, 2020. [Online; accessed 06-Feb-2020].

[125] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.

[126] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[127] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. " O'Reilly Media, Inc.", 2019.

[128] R. I. Grosvenor and P. W. Prickett, "A discussion of the prognostics and health management aspects of embedded condition monitoring system," in *Annual Conference of the PHM Society*, vol. 3, 2011.

[129] Q. Song, F. Luan, Z. Shi, T. Li, and M. Wang, "Design of turbidity remote monitoring system based on fx-11a optical fiber sensor," in *2020 Prognostics and Health Management Conference (PHM-Besançon)*, pp. 291–294, IEEE, 2020.

[130] K. M. Farinholt, A. Chaudhry, M. Kim, E. Thompson, N. Hipwell, R. Meekins, S. Adams, P. Beling, and S. Polter, "Developing health management strategies using power constrained hardware," in *PHM Society Conference*, vol. 10, 2018.

[131] H. Ren, D. Anicic, and T. Runkler, "Tinyol: Tinyml with online-learning on microcontrollers," *arXiv preprint arXiv:2103.08295*, 2021.

[132] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, 2021.

[133] G. Crocioni, D. Pau, J.-M. Delorme, and G. Gruosso, "Li-ion batteries parameter estimation with tiny neural networks embedded on intelligent iot microcontrollers," *IEEE Access*, vol. 8, pp. 122135–122146, 2020.

[134] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, *et al.*, "Benchmarking tinyml systems: Challenges and direction," *arXiv preprint arXiv:2003.04821*, 2020.

[135] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, *et al.*, "Tensorflow lite micro: Embedded machine learning on tinyml systems," *arXiv preprint arXiv:2010.08678*, 2020.

[136] G. Crocioni, G. Gruosso, D. Pau, D. Denaro, L. Zambrano, and G. Di Giore, "Characterization of neural networks automatically mapped on automotive-grade microcontrollers," *arXiv preprint arXiv: 2103.00201*, 2021.

[137] C. Vuppalapati, A. Ilapakurti, K. Chillara, S. Kedari, and V. Mamidi, "Automating tiny ml intelligent sensors devops using microsoft azure," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 2375–2384, IEEE, 2020.

[138] Chris Knorowski, "Building a TinyML Application with TF Micro and SensiML." https://blog.tensorflow.org/2021/05/building-tinyml-application-with-tf-micro-and-sensiml.html, 2021. [Online; accessed 19-June-2021].

[139] Louis Moreau, Mihajlo Raljic, "Enhancing Health and Safety in Industrial Environments with Embedded Machine Learning." https://www.edgeimpulse.com/blog/enhancing-health-and-safety-in-industrial-environments-with-embedded-machine-learning, 2021. [Online; accessed 19-June-2021].

[140] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance evaluation of iot data management using mongodb versus mysql databases in different cloud environments," *IEEE Access*, vol. 8, pp. 110656–110668, 2020.

[141] S. Amghar, S. Cherdal, and S. Mouline, "Which nosql database for iot applications?," in *2018 international conference on selected topics in mobile and wireless networking (mownet)*, pp. 131–137, IEEE, 2018.

[142] M. Nasar and M. A. Kausar, "Suitability of influxdb database for iot applications," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 10, pp. 1850–1857, 2019.

[143] A. Priambodo and A. Nugroho, "Design & implementation of solar powered automatic weather station based on esp32 and gprs module," in *Journal of Physics: Conference Series*, vol. 1737, p. 012009, IOP Publishing, 2021.

[144] A. Mohan and S. Poobal, "Crack detection using image processing: A critical review and analysis," *Alexandria Engineering Journal*, vol. 57, no. 2, pp. 787–798, 2018.

[145] Q. Zou, Z. Zhang, Q. Li, X. Qi, Q. Wang, and S. Wang, "Deepcrack: Learning hierarchical convolutional features for crack detection," *IEEE Transactions on Image Processing*, vol. 28, no. 3, pp. 1498–1512, 2018.

[146] L. Zhang, F. Yang, Y. D. Zhang, and Y. J. Zhu, "Road crack detection using deep convolutional neural network," in *2016 IEEE international conference on image processing (ICIP)*, pp. 3708–3712, IEEE, 2016.

[147] X. Zhang, D. Rajan, and B. Story, "Concrete crack detection using context-aware deep semantic segmentation network," *Computer-Aided Civil and Infrastructure Engineering*, vol. 34, no. 11, pp. 951–971, 2019.

[148] Ç. F. Özgenel and A. G. Sorguç, "Performance comparison of pretrained convolutional neural networks on crack detection in buildings," in *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 35, pp. 1–8, IAARC Publications, 2018.

[149] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[150] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[151] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[152] J. Chen, G. Liu, and X. Chen, "Road crack image segmentation using global context u-net," in *Proceedings of the 2019 3rd International Conference on Computer Science and Artificial Intelligence*, pp. 181–185, 2019.

[153] Tensorflow Development Team, "TFLite Micro Kernel Ops." https://github.com/tensorflow/tflite-micro/tree/main/tensorflow/lite/micro/kernels, 2021. [Online; accessed 10-July-2021].

[154] Simone, "EloquentTinyML Arduino Library." https://github.com/eloquentarduino/EloquentTinyML, 2020. [Online; accessed 10-July-2021].

[155] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[156] P. Warden, "Leaving google, starting stanford," 2022.

[157] B. D. Rouhani, M. Samragh, M. Javaheripi, T. Javidi, and F. Koushanfar, "Deepfense: Online accelerated defense against adversarial deep learning," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[158] C. Song, H.-P. Cheng, H. Yang, S. Li, C. Wu, Q. Wu, Y. Chen, and H. Li, "Mat: A multi-strength adversarial training method to mitigate adversarial attacks," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 476–481, IEEE, 2018.

[159] R. Kastner, J. Matai, and S. Neuendorffer, "Parallel programming for fpgas," *CoRR*, vol. abs/1805.03648, 2018.

[160] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[161] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[162] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 15–26, ACM, 2017.

[163] N. Narodytska and S. P. Kasiviswanathan, "Simple black-box adversarial perturbations for deep networks," *arXiv preprint arXiv:1612.06299*, 2016.

[164] A. Galloway, G. W. Taylor, and M. Moussa, "Attacking binarized neural networks," *arXiv preprint arXiv:1711.00449*, 2017.

[165] P. L. Bartlett and M. H. Wegkamp, "Classification with a reject option using a hinge loss," *Journal of Machine Learning Research*, vol. 9, no. Aug, pp. 1823–1840, 2008.

[166] C. Recommendation, "601, encoding parameters of digital television for studios," *Digital Methods of Transmitting Television Information*, pp. 95–104, 1990.

[167] Avnet Incorporation, "Avnet Ultra96 Zynq Ultrascale+ Development Board ." http://zedboard.org/sites/default/files/product_briefs/5354-pb-ultra96-v3b.pdf, 2018. [Online; accessed 19-July-2019].

[168] A. G. Schmidt, G. Weisz, and M. French, "Evaluating rapid application development with python for heterogeneous processor-based fpgas," *arXiv preprint arXiv:1705.05209*, 2017.

[169] Xilinx Incorporation, "Quantized Neural Networks (QNNs) on PYNQ ." https://github.com/Xilinx/BNN-PYNQ/, 2017. [Online; accessed 19-July-2019].

[170] Keras Team, "Train a simple deep cnn on the cifar10 small images dataset.," 2019. [Online; accessed 19-Mar-2020].

[171] 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)*, pp. 1–6, IEEE, 2018.

[172] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[173] X. Zhou, R. Canady, S. Bao, and A. Gokhale, "Cost-effective hardware accelerator recommendation for edge computing," in *3rd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[174] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8612–8620, 2019.

[175] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 578–594, 2018.

[176] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[177] Z. Xu, H. S. Shah, and U. Ramachandran, "Coral-pie: A geo-distributed edge-compute solution for space-time vehicle tracking," in *Proceedings of the 21st International Middleware Conference*, pp. 400–414, 2020.

[178] K. Duncan, E. Komendantskaya, R. Stewart, and M. Lones, "Relative robustness of quantized neural networks against adversarial attacks," in *2020 IEEE International Joint Conference on Neural Networks*, IEEE, 2020.

[179] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," *arXiv preprint arXiv:1904.08444*, 2019.

[180] Y. Fu, Y. Zhao, Q. Yu, C. Li, and Y. Lin, "2-in-1 accelerator: Enabling random precision switch for winning both adversarial robustness and efficiency," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 225–237, 2021.

[181] Z. Tang, Y. Dong, and H. Su, "Error-silenced quantization: Bridging robustness and compactness.," in *AISafety@ IJCAI*, 2020.

[182] K. Gupta and T. Ajanthan, "Improved gradient-based adversarial attacks for quantized networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 6810–6818, 2022.

[183] Y. Fu, Q. Yu, M. Li, V. Chandra, and Y. Lin, "Double-win quant: Aggressively winning robustness of quantized deep neural networks via random precision training and inference," in *International Conference on Machine Learning*, pp. 3492–3504, PMLR, 2021.

[184] H. Hirano and K. Takemoto, "Simple iterative method for generating targeted universal adversarial perturbations," *Algorithms*, vol. 13, no. 11, p. 268, 2020.

[185] S. Gui, H. N. Wang, H. Yang, C. Yu, Z. Wang, and J. Liu, "Model compression with adversarial robustness: A unified optimization framework," in *Advances in Neural Information Processing Systems*, pp. 1285–1296, 2019.

[186] F. Croce, M. Andriushchenko, V. Sehwag, N. Flammarion, M. Chiang, P. Mittal, and M. Hein, "Robustbench: a standardized adversarial robustness benchmark," *arXiv preprint arXiv:2010.09670*, 2020.

[187] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense against adversarial attacks using high-level representation guided denoiser," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1778–1787, 2018.

[188] C. Guo, M. Rana, M. Cisse, and L. Van Der Maaten, "Countering adversarial images using input transformations," *arXiv preprint arXiv:1711.00117*, 2017.

[189] F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks," in *International conference on machine learning*, pp. 2206–2216, PMLR, 2020.

[190] A. Fawzi, H. Fawzi, and O. Fawzi, "Adversarial vulnerability for any classifier," *Advances in neural information processing systems*, vol. 31, 2018.

[191] A. Raghunathan, S. M. Xie, F. Yang, J. Duchi, and P. Liang, "Understanding and mitigating the tradeoff between robustness and accuracy," *arXiv preprint arXiv:2002.10716*, 2020.

[192] P. Panda, "Quanos: adversarial noise sensitivity driven hybrid quantization of neural networks," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 187–192, 2020.

[193] Y. Huang, H. Zhang, Y. Shi, J. Z. Kolter, and A. Anandkumar, "Training certifiably robust neural networks with efficient local lipschitz bounds," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22745–22757, 2021.

[194] Y.-Y. Yang, C. Rashtchian, H. Zhang, R. R. Salakhutdinov, and K. Chaudhuri, "A closer look at accuracy vs. robustness," *Advances in neural information processing systems*, vol. 33, pp. 8588–8601, 2020.

[195] D. Wu, S.-T. Xia, and Y. Wang, "Adversarial weight perturbation helps robust generalization," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[196] C. Song, R. Ranjan, and H. Li, "A layer-wise adversarial-aware quantization optimization for improving robustness," *arXiv preprint arXiv:2110.12308*, 2021.

[197] BrainChip Incorporation, "The akida neural processor cnn2snn toolkit," 2020.

[198] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[199] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[200] R. Rade and S.-M. Moosavi-Dezfooli, "Helper-based adversarial training: Reducing excessive margin to achieve a better accuracy vs. robustness trade-off," in *ICML 2021 Workshop on Adversarial Machine Learning*, 2021.

[201] S. Gowal, C. Qin, J. Uesato, T. Mann, and P. Kohli, "Uncovering the limits of adversarial training against norm-bounded adversarial examples," *arXiv preprint arXiv:2010.03593*, 2020.

[202] Y. Carmon, A. Raghunathan, L. Schmidt, P. Liang, and J. C. Duchi, "Unlabeled data improves adversarial robustness," *arXiv preprint arXiv:1905.13736*, 2019.

[203] NVIDIA Incorporation, "8-bit inference with tensorrt," 2017.

[204] Y. Guo, T. Ji, Q. Wang, L. Yu, and P. Li, "Quantized adversarial training: An iterative quantized local search approach," in *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 1066–1071, IEEE, 2019.

[205] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[206] M. Pintor, L. Demetrio, A. Sotgiu, G. Manca, A. Demontis, N. Carlini, B. Biggio, and F. Roli, "Indicators of attack failure: Debugging and improving optimization of adversarial examples," *arXiv preprint arXiv:2106.09947*, 2021.

[207] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[208] C. Tsigkanos, C. Avasalcai, and S. Dustdar, "Architectural considerations for privacy on the edge," *IEEE Internet Computing*, vol. 23, no. 4, pp. 76–83, 2019.

[209] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy-preserving in edge computing paradigm: Survey and open issues," *IEEE Access*, vol. 6, pp. 18209–18237, 2018.

[210] B. S. Gu, L. Gao, X. Wang, Y. Qu, J. Jin, and S. Yu, "Privacy on the edge: Customizable privacy-preserving context sharing in hierarchical edge computing," *IEEE Transactions on Network Science and Engineering*, 2019.

[211] E. Gilman and D. Barth, *Zero Trust Networks*. O'Reilly Media, Incorporated, 2017.

[212] H. Wu, X. Tian, M. Li, Y. Liu, G. Ananthanarayanan, F. Xu, and S. Zhong, "Pecam: privacy-enhanced video streaming and analytics via securely-reversible transformation," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pp. 229–241, 2021.

[213] S. Taghavi and W. Shi, "Edgemask: An edge-based privacy preserving service for video data sharing," in *Proceedings of the 3rd Workshop on Security and Privacy in Edge Computing (EdgeSP)*, pp. 2327–4662, 2020.

[214] J. He, B. Liu, D. Kong, X. Bao, N. Wang, H. Jin, and G. Kesidis, "Puppies: Transformation-supported personalized privacy preserving partial image sharing," in *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp. 359–370, IEEE, 2016.

[215] S. Shan, E. Wenger, J. Zhang, H. Li, H. Zheng, and B. Y. Zhao, "Fawkes: Protecting privacy against unauthorized deep learning models," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pp. 1589–1604, 2020.

[216] M. Xue, C. Yuan, C. He, Z. Wu, Y. Zhang, Z. Liu, and W. Liu, "Use the spear as a shield: A novel adversarial example based privacy-preserving technique against membership inference attacks," *arXiv preprint arXiv:2011.13696*, 2020.

[217] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, "Memguard: Defending against black-box membership inference attacks via adversarial examples," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 259–274, 2019.