1-2021

Development of a Reference Design for Intrusion Detection Using Neural Networks for a Smart Inverter

Ammar Mohammad Khan
University of Arkansas, Fayetteville

Development of a Reference Design for Intrusion Detection Using Neural Networks for a Smart Inverter

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Engineering

by

Ammar Mohammad Khan
Portland State University
Bachelor of Science in Computer Engineering, 2019

July 2021
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

ABSTRACT

The purpose of this thesis is to develop a reference design for a base level implementation of an intrusion detection module using artificial neural networks that is deployed onto an inverter and runs on live data for cybersecurity purposes, leveraging the latest deep learning algorithms and tools. Cybersecurity in the smart grid industry focuses on maintaining optimal standards of security in the system and a key component of this is being able to detect cyberattacks. Although researchers and engineers aim to design such devices with embedded security, attacks can and do still occur. The foundation for eventually mitigating these attacks and achieving more robust security is to identify them reliably. Thus, a high-fidelity intrusion detection system (IDS) capable of identifying a variety of attacks must be implemented. This thesis provides an implementation of a behavior-based intrusion detection system that uses a recurrent artificial neural network deployed on hardware to detect cyberattacks in real time. Leveraging the growing power of artificial intelligence, the strength of this approach is that given enough data, it is capable of learning to identify highly complex patterns in the data that may even go undetected by humans. By intelligently identifying malicious activity at the fundamental behavior level, the IDS remains robust against new methods of attack. This work details the process of collecting and simulating data, selecting the particular algorithm, training the neural network, deploying the neural network onto hardware, and then being able to easily update the deployed model with a newly trained one. The full system is designed with a focus on modularity, such that it can be easily adapted to perform well on different use cases, different hardware, and fulfill changing requirements. The neural network behavior-based IDS is found to be a very powerful method capable of learning highly complex patterns and identifying intrusion from different types of attacks using a single unified algorithm, achieving up to 98% detection

accuracy in distinguishing between normal and anomalous behavior. Due to the ubiquitous nature of this approach, the pipeline developed here can be applied in the future to build in more and more sophisticated detection abilities depending on the desired use case.

The intrusion detection module is implemented in an ARM processor that exists at the communication layer of the inverter. There are four main components described in this thesis that explain the process of deploying an artificial neural network intrusion detection algorithm onto the inverter: 1) monitoring and collecting data through a front-end web based graphical user interface that interacts with a Digital Signal Processor that is connected to power-electronics, 2) simulating various malicious datasets based on attack vectors that violate the Confidentiality-Integrity-Availability security model, 3) training and testing the neural network to ensure that it successfully identifies normal behavior and malicious behavior with a high degree of accuracy, and lastly 4) deploying the machine learning algorithm onto the hardware and having it successfully classify the behavior as normal or malicious with the data feeding into the model running in real time. The results from the experimental setup will be analyzed, a conclusion will be made based upon the work, and lastly discussions of future work and optimizations will be discussed.

ACKNOWLEDGMENTS

My gratitude goes to my advisor, Dr. H. Alan Mantooth, who has been my guide throughout my graduate education providing me with a deeper understanding of cybersecurity as well as many other opportunities to learn and grow. His approach to mentorship as both made me grow as a professional and as a person. I would also like to thank Dr. Qinghua Li, Dr. Jia Di, and Dr. Chris Farnell for being on my advising committee, directing me along the way. It has also been a pleasure to work with Nicholas Blair, Estefano Soria, Brandon Meyers Ortiz and many other students in the cyber security group who have helped me progress.

TABLE OF CONTENTS

Chapter 1 1

Introduction 1

1.1 Motivation 1

1.2 Intrusion Detection 3

1.4 Thesis Objectives 6

1.4.1 Thesis Organization 7

1.5 References 8

Chapter 2 9

Background 9

2.1 Introduction 9

2.2 The Inverter Architecture Overview 9

2.3 The Chosen Neural Network Algorithm 20

2.4 References 27

Chapter 3 29

Test Setup and Experimental Results 29

3.1 Introduction 29

3.2 Data Collection Process 29

3.3 Implementing the Neural Network Model 44

3.4 Training and Testing the Neural Network 45

3.5 Deploying the Neural Network Onto the Hardware 46

3.5 References 48

Chapter 4 49

Evaluation of The Neural Network Based IDS 49

4.1 Introduction 49

4.2 Evaluation of the Train and Test Datasets 49

4.3 Results of the Attack Simulated On the Deployed Model 55

4.4 Hardware Usage of the Artificial Neural Network Algorithm 57

4.5 References 57

Chapter 5 59

Conclusions and Future work 59

5.1 Conclusions 59

5.2 Recommendations and Future Work 60

LIST OF FIGURES

Figure 2.1. Reference Design Architecture. 10

Figure 2.2. Physical Device Architecture Recognition. 11

Figure 2.3. Communication layer topology. 12

Figure 2.4. Raspberry Pi 4 Model B board layout. 13

Figure 2.5. Intrusion Detection Module. 16

Figure 2.6. Neural Network Nodes [9]. 20

Figure 2.7. Equation for Input Node [9]. 22

Figure 2.8. Equation for Output Node [9]. 22

Figure 2.9. Gradient Dissent [9]. 23

Figure 2.10. LSTM Auto-encoder Model. 26

Figure 3.1. Communication path diagram. 31

Figure 3.2. Wireshark Capture. 32

Figure 3.3. Example of a Dataset. 33

Figure 3.4. Normal Collected Data Plot. 35

Figure 3.5. Malicious Collected Data Plot. 36

Figure 3.6. Simulated Normal Behavior. 37

Figure 3.7. Normal Collected Data. 38

Figure 3.8. Denial of Service Attack Data. 39

Figure 3.9. Faulty Injection Data Attack Data. 40

Figure 3.10. Sudden Voltage Spike Attack Data. 41

Figure 3.11. Webserver GUI. 43

Figure 4.1. Losses Over Each Epoch. 50

Figure 4.2. Losses Over each Epoch for Improved Dataset. 53

Figure 4.3. Losses Over each Epoch for First Dataset.. 54

Figure 4.4. IDS Detection Screenshot.. 56

LIST OF TABLES

Table 2.1. STRIDE Table Definitions 17

Table 2.2. STRIDE Threat Model[4]. 18

Table 2.3. List of Machine Learning Algorithms Benefits and Costs. 24

CHAPTER 1

INTRODUCTION

1.1 Motivation

Providing electricity through power grids is a crucial component for the function of society in terms of comfort, leisure, and functionality. As technology has advanced, grid connected devices have also evolved in their levels of complexity and computation. Grid connected devices have been put on a network that allows for more communication and analyses of the devices. The benefits of integrating such devices on a network connected to the internet include efficient communication to alter settings and system output and being able to monitor inverter behavior more easily. However, securing the inverters has now become of paramount importance because they may now be vulnerable to cyber-attacks such as interference, modifications, and damage. Some of the cyber-attacks include spoofing the data, denial of service, or alterations of data to damage the inverter. For instance, a section of the grid in Ukraine during 2016 was shut down by cyber attackers [1]. The metrics of confidentiality, integrity, and authenticity need to be maintained when implementing security strategies for grid connected devices.

Cyber-attacks can come in different forms and at various levels for grid-connected devices. Inverters generally have multiple layers such as the communication layer, control layer, and hardware layer where each layer has its own vulnerabilities [2]. Attackers can gain access at the communication layer, which is the top layer of the inverter, through means such as Man in the middle, spoofing, phishing, Denial of service, Structured Query Language injection, and other types of methods. After gaining access at the communication layer, the attacker can deploy various attacks by sending malicious commands to the inverter or by sending faulty data to the inverter, which may result in damaging the power electronic device. There are various approaches to addressing attacks and vulnerabilities at the inverter level. Many cybersecurity methods attempt to mitigate such vulnerabilities by initially designing the inverter in a secure manner. However, with the rapid advancement of technology and new methods, additional vulnerabilities are always a threat.

In recent years, there was a cyberattack on IT service provider Solar Winds and of its victims were electricity and oil companies. The Ukraine attack on the power grid in 2016 and even this recent attack on Solar Winds both demonstrate the importance of cybersecurity, as such attacks affect not only the companies themselves but large portions of the population. With the advancements of IOT devices, grid connected devices have now been put on a network, and although that has brought immense benefit it has also made these devices vulnerable to cyberattacks. Moreover, while researchers and engineers will always attempt to make their designs secure, there is no guarantee that the inverter will not be attacked. There is additionally no guarantee that the existing security systems will be able to handle the attack, due to the evolving nature of technology. As a result, a crucial step in addressing security issues involves intrusion detection, and in particular an intrusion detection approach that will be adaptable to newly discovered attacks without having to re-implement the base algorithm.

This work attempts to help secure grid-connected devices by implementing a machine learning based intrusion detection system at the communication layer of the inverter. A neural network behavior-based detection algorithm serves as the base of this work, due to its wide applicability in computer science in learning to perform a wide variety of identification tasks using data. The deployment of the neural network algorithm is to make intrusion detection more efficient at identifying a diverse range of anomalous behavior that could have resulted from an adversary. The goal of this research is to detail the process of training and deploying a neural network to detect and learn behaviors that could result from various types of attacks on an inverter without the operator having to consistently monitor the inverter, along with implementing an intrusion detection algorithm that is robust to new cyberattacks because new attacks are always being developed and as a result the solution needs to be adaptable as well as modular.

1.2 Intrusion Detection

Cybersecurity involves numerous principles in order to sustain the standards necessary for secure practices. If a system is methodically secure, then undesired influences would have a minimal impact on the system. However, no matter how secure a system is, an intrusion or attack is always a real threat. Therefore, in order to be able to mitigate attacks it is of paramount importance to understand the intrusion event and be able to detect such an event should it occur. In order to be able to detect malicious behavior, one security method deployed is implementing an intrusion detection system. An intrusion detection system, also known as an IDS, is a software application or a device that monitors a network for malicious activity or cyberattacks [3]. Generally, any malicious activity or violation would be reported or flagged in the system.

There are multiple types of intrusion detection depending on the application of the device. Two types of intrusion detection systems are signature-based detection and behavior-based detection. Signature-based detection is the traditional approach, which generally involves observing the packet's signature or length of the bytes in network traffic. More recent is the second type of intrusion detection: a behavior-based approach, wherein machine learning is used to observe patterns and trends in the behavior of a certain metric in order to detect an attack. This technique uses a machine learning algorithm that is trained to identify the range of what constitutes normal behavior. If incoming behavior does not possess qualities considered normal, this is then classified as an anomaly and potentially malicious. This approach is superior to the signature-based approach as it determines anomalies based on trends and behavior, and not just packet length or size, and can therefore detect more complex attacks and also attacks that bypass signature-based detection. Furthermore, there is a need for this approach because there are always newer types of cyberattacks that are being developed and a behavioral based approach offers a solution that is adaptable and robust to newer technology. The intrusion detection method implemented in this work is a behavior-based approach that detects anomalies in a voltage time series from a photovoltaic inverter.

1.3 Machine Learning and Neural Networks

Machine learning has been at the forefront of the artificial intelligence boom in recent years. Machine-learning algorithms analyze statistics on tremendous amounts of data in order to find patterns, often matching and even surpassing human pattern recognition abilities [4]. The type of data can range from alphabets, numerical values, words, images, clicks, voltages, and even more types of data. Most modern intelligent systems on the Internet such as “Duck Duck Go” or “Bing” use machine learning extensively. Additionally, voice assistant modules found in various phones and devices also use machine learning in assisting their users. Each of these systems leverage large amounts of data collected from users and other sources to fuel their performance.

Neural networks and deep learning form a key branch of machine learning that has enabled many of the newest advancements in artificial intelligence. Artificial neural networks consist of node layers that include an input layer, additional hidden layers depending on the complexity of the algorithm, and an output layer [5]. Deep learning uses many layers of these computational nodes working in conjunction to sift through data and yield a final result that is a prediction. A deep learning algorithm fundamentally consists of multiple neural networks or a single network with many layers [6]. Neural network layers are composed of neurons, connections and weights. The neuron, a type of node based loosely on a biological neuron, computes the weighted average of the data input and passes that information through a non-linear function [6]. The connection and weights connect the neurons in the same layer and to other layers and each one has a weight value that represents the strength of the connection. The goal is to tweak the weights such that the output produces low error relative to the desired task. There are two propagation functions; a forward propagation function that delivers a predicted value, and a backward propagation function that yields the error value [6]. The learning rate is a tunable metric that is used to determine how quickly the value of the weights should be updated. The weights are used to decrease the amount of loss that is incurred when the model attempts to make a prediction based on its data inputs. Furthermore, there are other machine learning algorithms that can be used for intrusion detection and the one chosen for this research is the recurrent neural network approach because of its highly accurate detection rate, ability to accept multiple metrics, and additional factors that will be elaborated upon in Chapter 2.

This thesis utilizes the artificial neural network approach for the research considered because it can learn complex patterns from any arbitrary dataset and continually improve its ability to distinguish between normal and anomalous behavior. Identifying attacks in cybersecurity is of paramount importance and although there are various ways of identifying threats, neural networks are particularly powerful due to their quality of being universal function approximators, meaning that they can learn any arbitrary function – and hence almost all pattern recognition tasks – given enough data. Additionally, since the model learns based on behaviors, it can detect different types of attacks, independent of the way the attack was caused. Furthermore, using this type of software approach allows for the algorithm to be built upon to and improved with varying metrics without having to make modifications to the hardware. As a result, adopting a behavioral based approach to intrusion detection offers a robust solution to cyberattacks. Attacks that result in a breach of any of the members of the CIA triangle can be identified with an artificial neural network. For example, in the case of a man-in-the-middle data spoof attack wherein an attacker spoofs the data at the communication layer, the integrity of the data has been violated [7]. The attacker might be sending hazardous commands to the inverter, altering the voltage to damage the hardware, or shut it down. The behavior-based IDS would identify that the behavior is abnormal and malicious, concluding that an intrusion has occurred and flagging it as such. This approach aids in the process of mitigating a cyberattack by identifying the attack and notifying the system that a breach has occurred. Additionally, the behavior-based detection is not dependent upon the hardware, and hence is flexible to different types of inverters. Lastly, it addresses the need for a solution that is adaptable to detecting newer types of attacks.

1.4 Thesis Objectives

The purpose of this thesis is to investigate and explore a method for intrusion detection of cyberattacks for a power electronics device and an ARM processor using an artificial neural network algorithm in real time. This work is a reference design that incorporates collecting data to train and test a neural network algorithm to detect malicious behavior, deploys the neural network model onto an ARM processor, and also updates the algorithm on the hardware in a manner similar to patching. This reference design will portray the process of collecting data, simulating data, training the neural network, deploying it onto hardware and updating the algorithm. This thesis is divided into sections including the background information necessary to understand the concepts in this research, the approach of the design and the selection of the specific algorithm over others, the implementation of the security concepts, and the experimental results that analyze the features of the design.

1.4.1 Thesis Organization

The thesis is organized sequentially based on the listing below:

- Chapter 1 serves as an introduction to the concepts associated with this research such as the chosen approach which includes intrusion detection and neural networks, and the objectives.
- Chapter 2 provides an explanation of the background concepts needed to understand the functionality of the intrusion detection system at the inverter level using neural networks, the inverter layers, where the neural network sits in the design, the chosen algorithm, and the attack vectors considered.
- Chapter 3 explains the system implementation, including the process of collecting and simulating data, training the model, and fine-tuning its performance. Additionally, it outlines the process of deploying the neural network onto the hardware and being able to update the code.
- Chapter 4 summarizes the results and evaluates the neural network based on performance metrics. It analyzes the datasets and the efficacy of the model used for intrusion detection and explains the CPU usage incurred by the algorithm when deployed on hardware.

- Chapter 5 summarizes the conclusion and introduces potential future work and recommendations for building upon the current neural network implementation of the intrusion detection system at the inverter level.

1.5 References

[1] Q. Wang, W. Tai, Y. Tang and M. Ni, "Review of the false data injection attack against the cyber-physical power system", IET Cyber-Physical Systems: Theory & Applications, vol. 4, no. 2, pp. 101-107, 2019.

[2] "National Vulnerability Database," NVD, Jun-2021. [Online]. Available: https://nvd.nist.gov/vuln/full-listing. [Accessed: 07-Jun-2021].

[3] B. Networks, "Barracuda Networks," *What is an Intrusion Detection System? | Barracuda Networks*. [Online]. Available: https://www.barracuda.com/glossary/intrusion-detection-system. [Accessed: 01-Jul-2021].

[4] K. Hao, "What is machine learning?," *MIT Technology Review*, 05-Apr-2021. [Online]. Available: https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/. [Accessed: 01-Jul-2021].

[5] By: IBM Cloud Education, "What are Neural Networks?," *IBM*. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks. [Accessed: 01-Jul-2021].

[6] Larry Hardesty | MIT News Office, "Explained: Neural networks," *MIT News | Massachusetts Institute of Technology*. [Online]. Available: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. [Accessed: 02-Jul-2021].

[7] S. Deepika and P. Pandiaraja, "Ensuring CIA triad for user data using collaborative filtering mechanism," 2013 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, 2013, pp. 925-928.

CHAPTER 2

BACKGROUND

2.1 Introduction

It can be noted from the first chapter that the motivation for this research is to implement intrusion detection using a neural network as a reference design for a grid connected device. It was mentioned in the previous chapter that a key part in maintaining the security of a device -- namely confidentiality, integrity, and authenticity -- is to be able to detect cyberattacks. Cyberattacks can occur at different links in the communication of the inverter or device. The primary focus of this intrusion detection approach is at the communication layer of the inverter. The concepts related to the artificial neural network algorithm chosen for this intrusion detection implementation will also be mentioned and elaborated upon. This chapter will detail the architecture of the inverter, the various layers that are relevant to the intrusion detection process, and the location of the intrusion detection system in the design of the inverter. Furthermore, the communication path of the inverter will be shown and the protocol it uses will be explained in this chapter. A focus on the communication layer of the inverter will be explained and elaborated upon, while the other various layers will be mentioned briefly to explain the architecture of the complete device. Additionally, the attack vectors considered for training the neural network will be mentioned to setup up the explanation for it in the following chapter.

2.2 The Inverter Architecture Overview

The development of smart inverters with embedded computing has added many useful features for increased utility. However, adding these features has also increased the complexity of the design as additional layers will have varying components depending on design decisions made by engineers. The architecture chosen for the implementation of this research is an inverter with three layers, as can be seen below in Figure 2.1, including the communication layer, control layer, and hardware layer.

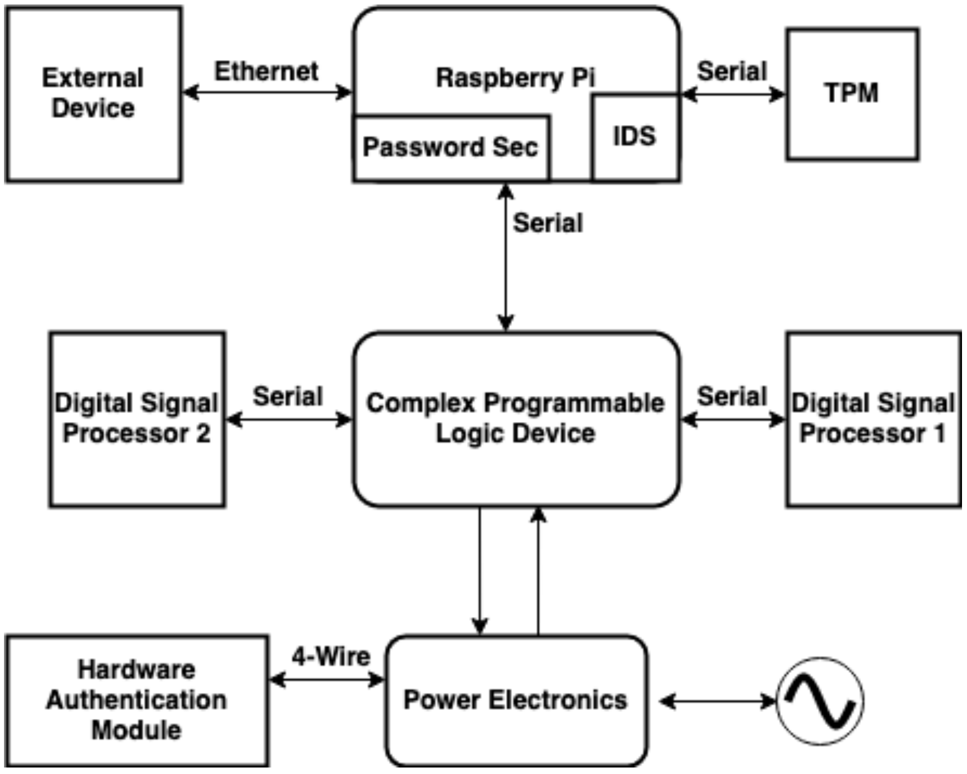

External Device
Ethernet
Raspberry Pi
Password Sec
IDS
Serial
TPM
Serial
Digital Signal Processor 2
Serial
Complex Programmable Logic Device
Serial
Digital Signal Processor 1
Hardware Authentication Module
4-Wire
Power Electronics

Figure 2.1. Reference design architecture of the topology of the inverter.

The top layer, known as the communication layer, consists of an ARM processor -- in this case the Raspberry Pi 4 -- which hosts an Intrusion Detection Module, a Password Security module, an External Device connected through Ethernet, and a Trusted Platform Module that is connected via serial. The next layer is the control layer which consists of a Complex Programmable Logic device, Digital Signal Processor 1, and a Digital Signal Processor 2 that are accessed through serial communication. The bottom layer consists of the Power Electronics Hardware and a Hardware Authentication Module. The physical components of each layer will be described in further detail below to understand the design. However, since the intrusion detection module exists at the top layer, the communication layer will be the primary focus.

For a more demonstrative understanding of the physical makeup of the reference design architecture, the components within the system are displayed in Figure 2.2. As can be seen here, the Raspberry Pi is separate from the UCB controller which hosts the digital signal processors as well as the field programmable logic device. The intrusion detection module that exists inside the Raspberry Pi communicates to the DSP through Modbus TCP, which is flashed with C code that configures and maps the appropriate registers to then send commands to adjust the values of the power electronics, along with receiving values that contain the updated metrics of the device.

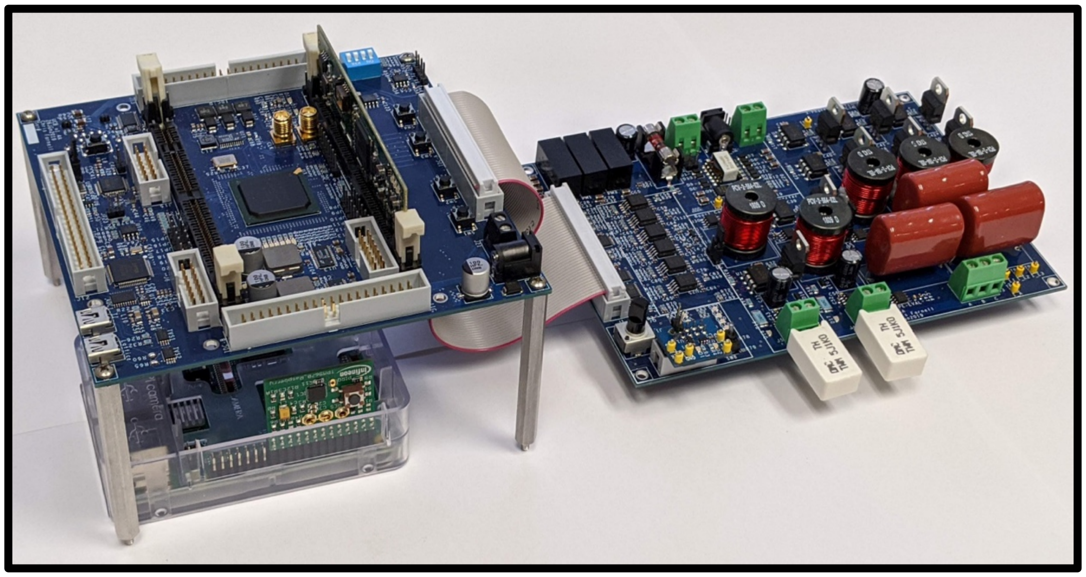


Figure 2.2. Physical device architecture of the inverter.

2.2.1 Communication Layer

Within the inverter or grid connected device, a method of communication typically exists in order to communicate aquired data into the system as well as out of the system in order to be processed and utilized. It is crucial to verify that the data reported from within the device is correct and realistic. The data delivers opportunities of growth for companies within the power industry that are aiming to develop a robust smart grid [1]. This occurs by observing the data received from the rest of the layers of the system to address any needs or concerns. Any issues that arise from within the system can then be documented and addressed by correcting the issue in the current system with an update or by developing a revision that corrects the design. In order to have better data mangement, data analysis, visualization, and processing of the data, having a specific communication layer is crucial. Figure 2.3 defines the communication layer topology used for this inverter. Some of the components such as the TPM or External device can be swapped out depending on the designer's needs, and even the Raspberry Pi could be swapped out depending on the implementation strategy. The key factor to remember, however, is that the device chosen must be able to sustain an intrusion detection module that implements a real time neural network algorithm. The main features of the Raspberry Pi will be described below, followed by why it is an effective device for implementing a machine learning algorithm for intrusion detection.

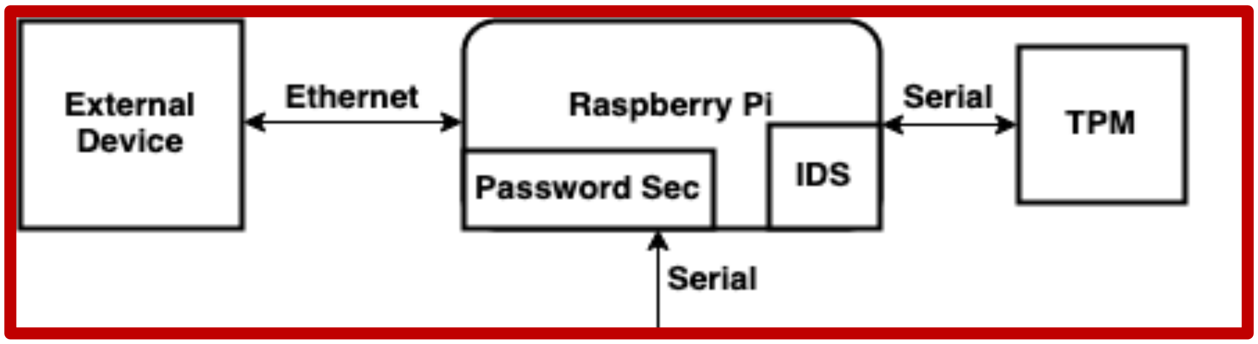


Figure 2.3. Communication layer topology.

The Raspberry Pi is the primary component of the communication layer in this inverter topology. The Raspberry Pi consists of multiple inputs and outputs for the user to interface with such as a keyboard and mouse for input and outputs, along with being able to directly access the General Purpose Input/Output (GPIO) registers to the central processor. Additionally, the

Operating System for the Raspberry Pi is Linux based, however it has its own name which is known as Raspbian. The GPIO registers provide direct communication with the processor over numerous standardized protocols in order to connect other devices such as the Trusted Platform Module or other devices. In order to understand what the Raspberry Pi looks like it can be seen in Figure 2.4. The primary reasons for choosing the Raspberry Pi 4 in this instance are because of the quad core ARM processor that can run a neural network algorithm, the Linux kernel, Gigabit Ethernet port, serial interface, being able to host a web-based server, and extensive documentation found in the community.

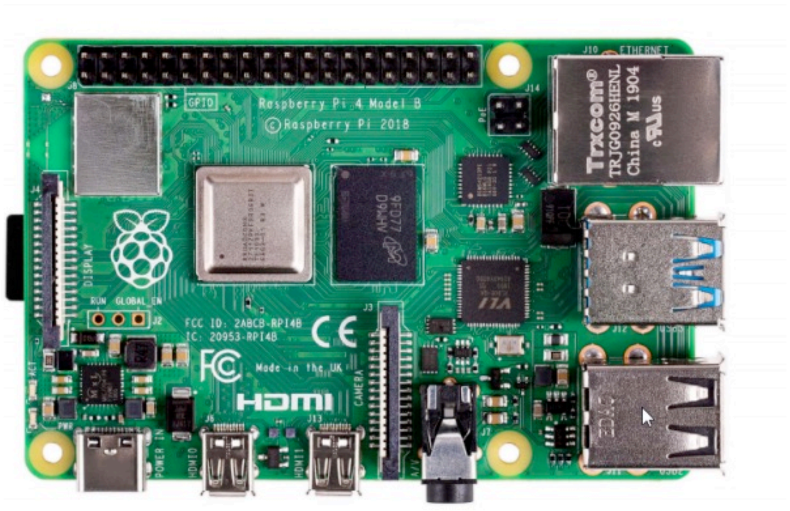


Figure 2.4. Raspberry Pi 4 Model B board layout.

For the communication layer, other ARM processors could be used that support a Linux kernel. The Linux kernel is a vital component for the communication layer and the artificial neural network algorithm because the kernel is compatible with the Python libraries used for implementing neural networks. The kernel is the base foundation of an operating system, which the particular operating system is the software implemented for the supporting the computer's base functionality. The Raspbian operating system with the Linux kernel is used because it is open source, compatible with data collection software, extremely well documented, and is versatile enough to accommodate different modules that can run simultaneously.

The core feature of the kernel is to serve as a hub for the required communication link between the inverter and the outside world. The kernel provides the necessary setup to allow the system to execute programs for inputting commands and transferring data. Furthermore, it has the necessary packages and libraries that are used by various algorithms, including collecting data, intrusion detection, communicating with register, and hosting a webserver. Being able to access the data from the control and hardware layers efficiently is important for the intrusion detection algorithm because that data is what will be fed into the artificial neural network algorithm. Before being able to access the raw data it must be scaled and formatted correctly to be comprehensible to both the user and the algorithm. This formatting occurs when communicating with the lower level registers in the system. The webserver, along with other scripts that are running within the system manage and manipulate the data. The webserver is a hub that routes the critical data to the correct destination. The webserver can then be used to display the data in various ways and provide the user with the ability to send and receive commands to and from the device. Additionally, the webserver ensures the data is accessible through an external communication device, such as the computer. The data that is sent is securely encrypted using a TLS encryption protocol. Also, the webserver has username and password required for a user to be able to access the Graphical User Interface for accessing the data. Additionally, the password management module allows the admin to control what type of users can be created to be able to read and write to the server, as well as ensuring that a strong password is selected for the user. Other forms of authentication such as multi-factor authentication are being researched as protecting passwords from attacks is important, however this is not the focus of this paper. The intrusion detection machine learning model runs in a Python script and it monitors the data entering the Raspberry Pi from the DSP in real time to determine whether the behavior of it is malicious or normal.

2.3 The Intrusion Detection Module

The intrusion detection module exists at the communication layer of the inverter and is running inside the Raspberry Pi. The intrusion detection module exists at this layer because typically an attacker will attempt to intercept communication at this layer as this is where the data is processed and managed. As a result, it is crucial to be able to detect an intrusion at this layer because an attack at this layer could result in downstream effects to the remaining layers in the topology. As mentioned in Chapter 1 there are multiple types of intrusion detection methods and algorithms that can be deployed. Signature based intrusion detection relies on the signature of the packets, packet length or byte size of the data, but is less robust as these can be falsified by a skilled attacker. On the other hand, behavior-based intrusion analyzes the behavior of the data it is monitoring and uses metrics and patterns in the data to determine whether that behavior is malicious or not and flag it as an intrusion if the behavior does not match what the model has been trained to understand as normal [2]. As can be seen in Figure 2.5 below any sort of malicious behavior could occur at the communication link from the grid or even during the data collection process. To account for this, the intrusion detection module has been implemented at the corresponding level in the design. The intrusion detection module deploys an artificial neural network algorithm that serves as the base of the intrusion detection system. This algorithm is fed data, is trained on that data, and then is deployed to run in real-time on a stream of data from the inverter. The details of the algorithm implementation will be elaborated upon in the next section.

From a high level design perspective, it is sufficient to note that the artificial neural network is the core component of the intrusion detection module.

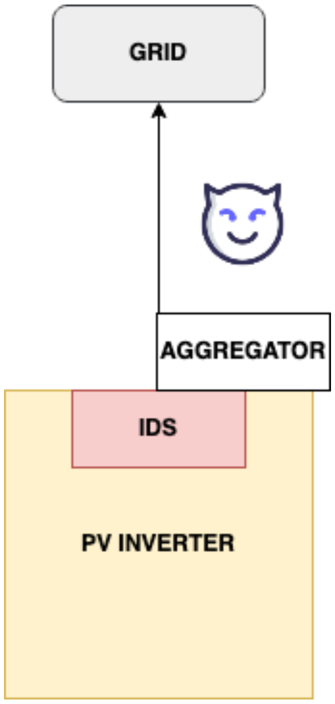


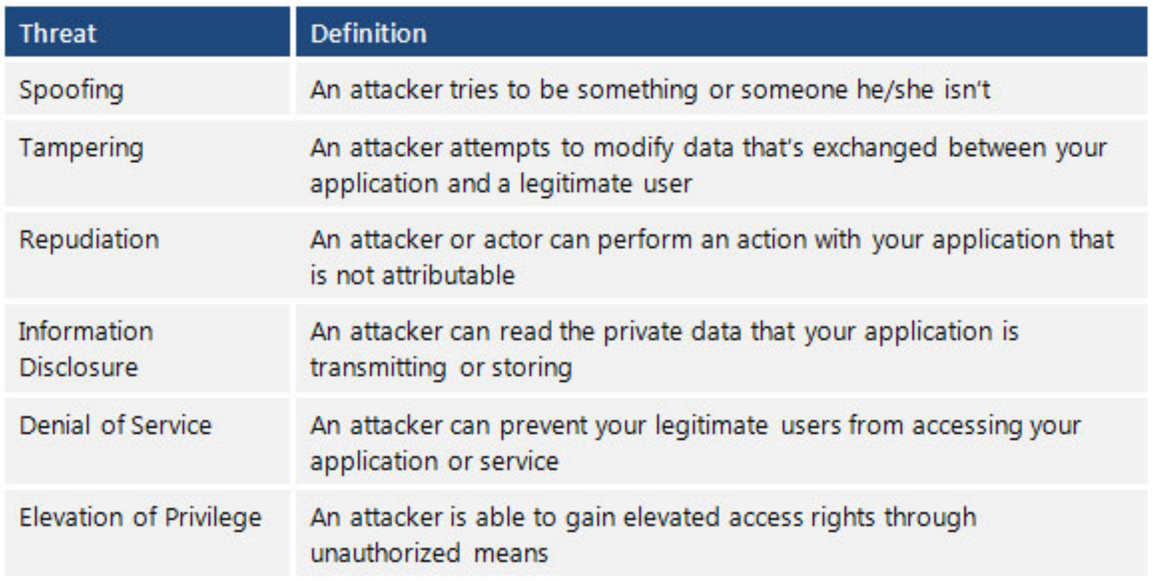
Figure 2.5. Intrusion Detection module.

The intrusion detection module is currently implemented in Python 3 because Python has various libraries for implementing machine learning algorithms and is also compatible with the Raspberry Pi 4. The script is consistently running on the Raspberry Pi 4, collecting data and analyzing it through its neural network algorithm. However, deploying this algorithm onto the Raspberry Pi is the last step. First, data needs to be collected, processed, and prepared for training. The model must then undergo a training scheme to optimize its parameters. Afterwards, the artificial neural network algorithm is fed a test dataset to validate that its performance. Once it has been validated, it is then saved and loaded onto hardware for deployment. It is at this stage that the model no longer takes in datasets but rather is fed real time data and then classifies the behavior as malicious or normal. This is merely a fundamental outline of how the intrusion detection algorithm works; it is important to understand which attack vectors were considered when implementing the intrusion detection module and how that influenced the design. This will be detailed in the next chapter.

2.3.1 Threats Considered and Modeling the Attack

There are numerous approaches that an assailant might deploy when looking to attack a system. Having a communication layer in the inverter topology allows for efficient data management and monitoring, yet this added benefit comes at a cost of a larger attack surface. As a result, when designing a smart inverter and implementing an intrusion detection module, it is important to consider various attack vectors. Cyberattacks are aimed at violating one or all components of the CIA triangle, namely confidentiality, integrity, and availability. One of the approaches of assessing the risk of potential attacks is to model that behavior using what is called a STRIDE Model [3]. As can be seen in Table 2.1 below, a table of the potential threats, the STRIDE Model consists of spoofing, tampering, repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege.

Table 2.1. STRIDE Table Definitions



Threat	Definition
Spoofing	An attacker tries to be something or someone he/she isn't
Tampering	An attacker attempts to modify data that's exchanged between your application and a legitimate user
Repudiation	An attacker or actor can perform an action with your application that is not attributable
Information Disclosure	An attacker can read the private data that your application is transmitting or storing
Denial of Service	An attacker can prevent your legitimate users from accessing your application or service
Elevation of Privilege	An attacker is able to gain elevated access rights through unauthorized means

This model is used to help find threats and determine what is the goal for a particular type of attack. A denial-of-service attack for example is when the attacker attempts to shut down the application of the service. In the case of a smart grid connected device, an attacker might want to shut down the power completely so the user is unable to access the device. In another instance, an attacker might tamper with the data and modify it to result in a denial of service or to damage the hardware of the device. Each threat listed in the STRIDE model is a violation of a particular security property, as can be seen below in Table 2.2.

Table 2.2. STRIDE Threat Model [4]

Threat	Security Property
Spoofing	**Authenticity**
Tampering	**Integrity**
Repudiation	**Non-repudiability**
Information Disclosure	**Confidentiality**
Denial of Service	**Availability**
Elevation of Privilege	**Authorization**

Each security property mentioned in the table above is aimed at preserving a particular component of security. Authenticity is the property of being able to verify that the data that is transmitted is valid and unaltered from the trusted source [5]. Spoofing would violate that as the data could be altered to cause the inverter to behave in a manner that is detrimental to the device. Additionally, the data is no longer verifiably from the intended sender. Integrity refers to guarding against improper tampering of the data or even preventing the destruction of it [6]. Non-repudiability refers to the principle that the owner or sender of the message cannot deny that they have sent the data and this is used to prove that the receiver received the data from specified sender. Confidentiality refers to ensuring that the data is known only by the authorized parties and no other entities. This policy is even more crucial than others as any secure link of communication must ensure this policy is fulfilled. Availability is ensuring that the data is accessible to the authorized users and in the case of a denial of service this policy would be violated. Authorization refers to the fact that only users authorized to access the data can access data, and for instance, if a non-admin user gains access to administrative data then this policy is violated. Although there are five security principles mentioned, generally only three are mentioned with the CIA triangle, as some of these policies are subsets of Confidentiality, Integrity, and Availability.

Different types of attacks can result in different policies being violated. One such attack would be a man in the middle data spoof attack. In such an attack, a third party can listen in on the communication link and alter the data before it arrives on the receiving end. This is an important attack vector to consider as some threat intelligent agencies state that 35% of exploitation activities involve these attacks [7]. When implementing the intrusion detection module this is the main attack vector to consider and this attack violates both policies of authenticity and integrity. The attacker could disrupt the communication link and alter the data. The next attack vector that is considered is the false data injection attack, which could result from an attacker injecting false data into the communication link to show incorrect data monitoring. This type of attack is different from merely a man in the middle attack, as the false data involves altering the data and attempts to deceive the operator. Additionally, detecting this type of attack is impossible using a signature-based intrusion detection approach, as the behavior of the data has been altered, while the packets, bytes, and other such traits remain the same [8]. As a result, implementing an artificial neural network that is trained to detect cyberattacks based on malicious behavior is crucial and this is one of the primary justifications for implementing an artificial neural network machine learning algorithm for intrusion detection instead of implementing signature-based detection. The final attack scenario considered is the Denial-of-Service attack, which acts with the intention of shutting down the data needed to keep the device operating. The following section will detail the specifically chosen artificial neural network algorithm that serves as the core of the intrusion detection module.

2.3 The Chosen Neural Network Algorithm

There are various types of machine learning algorithms that exist in the world of information technology. As mentioned in Chapter 1, an artificial neural network is a specific type of machine learning algorithm. Artificial neural networks consist of node layers, containing an input layer, multiple inner hidden layers, and an output layer as can be seen below in Figure 2.6.

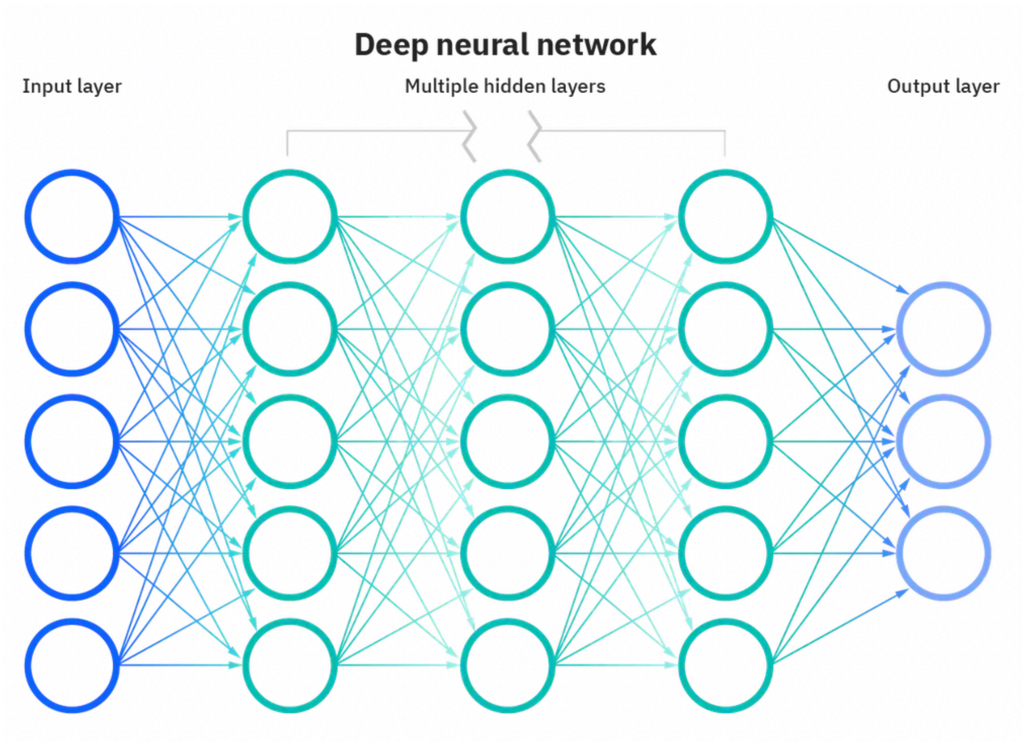


Figure 2.6. Neural Network Nodes [9].

One of the key advantages to using a neural network for the behavior-based intrusion detection task, as opposed to other machine learning methods, is that neural networks can fulfill the Universal Approximation Theorem. This means that with certain caveats, for any arbitrary function, a neural network configuration exists that can approach it. In practice, this suggests that even very complex input-output mapping can theoretically be accomplished by a neural network. In the context of the IDS, this is favorable because it implies that there are little to no inherent limits to the complexity of the behavior that can be analyzed and classified. Thus, it should be possible to use this single algorithm to train the IDS to be able to encapsulate normal and anomalous activity of arbitrary complexity.

Neural networks improve their accuracy through training on diverse datasets that encompass a wide range of possible examples. In this work, we use a neural network for a supervised learning task. This means that data is provided in the form of the raw data itself consisting of many examples of the data relevant for the given task, i.e. voltage trend examples in this case, as well as a corresponding label that denotes which category each example falls into, i.e. normal or anomalous in this case. The network then learns to match raw data to the correct classification based on the examples it has seen. It is only after they have been trained to a satisfactory degree of accuracy with data from the real world can neural networks then be deployed onto hardware to be used in real time to perform predictions on data points they have not been trained on. Each node in the neural network consists of a linear regression model and it consists of input data, weights, a threshold, and an output [9]. After the input layer is determined, weights are assigned. These weights are used to help determine the importance of a variable and the larger weights serve as a larger factor than inputs with smaller weights. All the inputs are multiplied by their individual weights and are summed together as can be seen in the neural network equation below in Figure 2.7 [9].

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

Figure 2.7. Equation for input node [9].

The output of that node is then passed through an activation function which yields a specific output. If that particular output exceeds the chosen threshold, it activates the node, and then passes the data to the next layer in the network. The general equation of the output function can be seen below in Figure 2.8 where it yields a binary 1 or 0 to classify the value based on the calculation of the function [9].

$$\text{output} = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases}$$

Figure 2.8. Equation for output node [9].

This description is of a feedforward neural network which is the more common neural network, however there are also neural networks that propagate data backwards as well. When trained successfully, the neural network must simply feed forward and compute the values at the output layer. These values then represent the prediction it made.

When training neural networks it is crucial to be able to evaluate the accuracy of the model by using an equation. A fundamental component of the training process is an equation that calculates the loss, which quantifies the correctness of the model. The model adjusts its weights and biases to optimize this loss function towards a minimum, also known as a point of convergence. This optimization is done through gradient descent, an efficient technique for optimizing high-dimensional multivariable functions, as loss functions typically have a dimensionality that scales with the number of features in the network. Gradient descent calculates the gradient vector with respect to the different variables in the cost function and tweaks the parameters such that the function moves in small jumps towards a minimum, as can be seen in Figure 2.9. The size of these “jumps” is quantified as the learning rate and is a key hyperparameter to consider in the training of a neural network. A lower learning rate is tantamount to smaller steps taken by gradient descent, which usually results in better long-term optimization of the function but slower and more computationally taxing training. A high learning rate means larger gradient descent steps and faster training, but a higher chance to get stuck in a suboptimal state for the network.

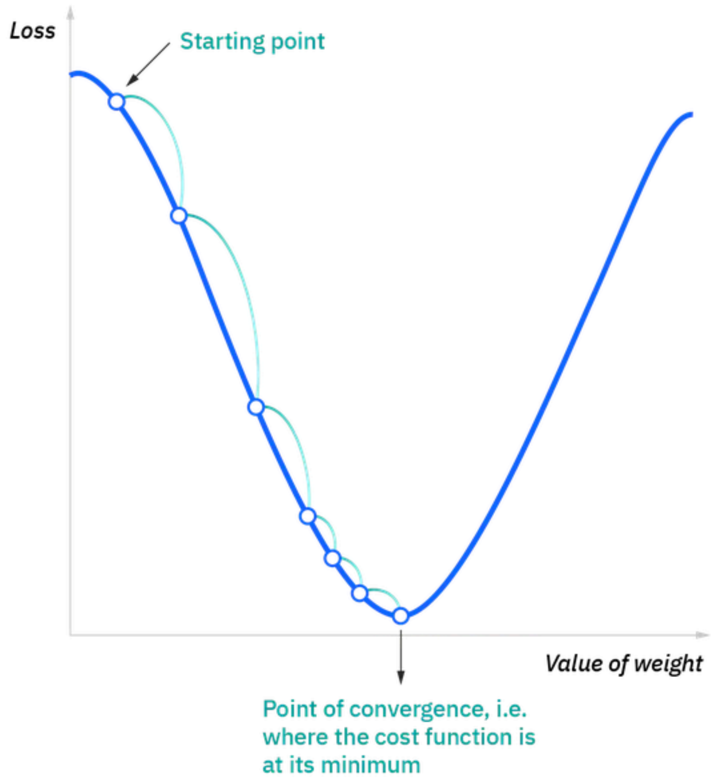

Loss
Starting point
Value of weight
Point of convergence, i.e. where the cost function is at its minimum

Figure 2.9. Illustration of gradient descent method [9].

The goal is to decrease the value of the loss with each epoch until it reaches the point of convergence seen in Figure 2.9, as that is where the model will be the most accurate. An epoch refers to training the neural network with all of the training data for a single cycle [9]. Gradient descent is used for optimization and the reason this model is chosen is because it is computationally too expensive for thousands of inputs in the model and hence gradient descent is used for this model.Training for different numbers of epochs will result in varying accuracy of the model. Typically, the larger and more diverse the training dataset, the less epochs are needed to achieve reasonable performance.

There are various other machine learning algorithms that can be implemented to detect anomalies in a time series data and some of the most recent algorithms with their benefits and costs can be seen below in Table 2.3.

Table 2.3. List of Machine Learning Algorithms Benefits and Costs.

Algorithm Name:	ARIMA	SARIMA	LSTM
Benefits:	Needs less data for training, Accurate with stationary datasets	Accurate with complex data, needs less data for training, can accept multiple metrics	Highly Accurate for stationary and dynamic data sets, accurate for long-term and short-term predictions, can accept multiple metrics
Costs:	Performs poorly with complex datasets, works with a single metric	Lower accuracy for long sequences of data, trains slower	Needs more data to be trained, more complex training process

The first algorithm known as the ARIMA algorithm consists of two primary components known as known as the Auto Regressive component and Moving Average component [11]. The first component is used to depict the regression of the prediction the model, while the second component is used to model the error of the algorithm with each iteration. The SARIMA algorithm builds upon the ARIMA algorithm with the added support of being compatible with seasonal data, meaning that unlike ARIMA, it can perform detection upon data that is not likely to repeat.

The algorithm chosen for the intrusion detection module is part of a class of neural networks known as recurrent neural networks. These network architectures use a recurrent feedback memory loop to handle time series data. Within this class, the specific algorithm is known as the Long Short-Term Memory (LSTM) Autoencoder. This algorithm was chosen for this research over the other algorithms for a variety of reasons. When compared to the ARIMA model, the LSTM autoencoder has been shown to be more accurate at identifying complex data patterns when analyzing power consumption data [12]. Additionally, for each training iteration the LSTM autoencoder has also been shown to be 85% more accurate [13]. When both algorithms are compared the ARIMA model has much lower accuracy and for the implementation of intrusion detection for voltage behavior at the grid-edge, being able to accurately identify behavior with more complex datasets is of paramount importance as voltage patterns can vary greatly. The SARIMA algorithm could also be used for implementing an IDS at the grid edge, however its added structure for factoring in seasonal data is not relevant for the task at hand. The goal is to detect cyber-attacks, not to merely predict voltage trends, and as a result the added feature of seasonal data adaptation is not of benefit. The LSTM autoencoder does need more data to be trained when compared to these other algorithms, however that added cost is not an issue for training the IDS, because training the dataset will not occur on the ARM process of the inverter, but at a separate device. The LSTM autoencoder algorithm is chosen over the other mentioned algorithms because of its high accurate detection rate, it has the ability to factor in multiple metrics and thus can be expandable for future implementations, and it is successfully able to work on complex data sets.

Neural networks use memory and add the ability to learn from data of a temporal nature and recognize patterns therein. This makes it possible to perform analysis on time series data such as speech, as well as voltage in this case. An autoencoder is a type of neural network that learns a compressed representation of an input [10]. It can be seen in Figure 2.9, that an LSTM Autoencoder consists of an encoder that takes an input and compresses it and then a decoder that attempts to reconstruct the data as an output.

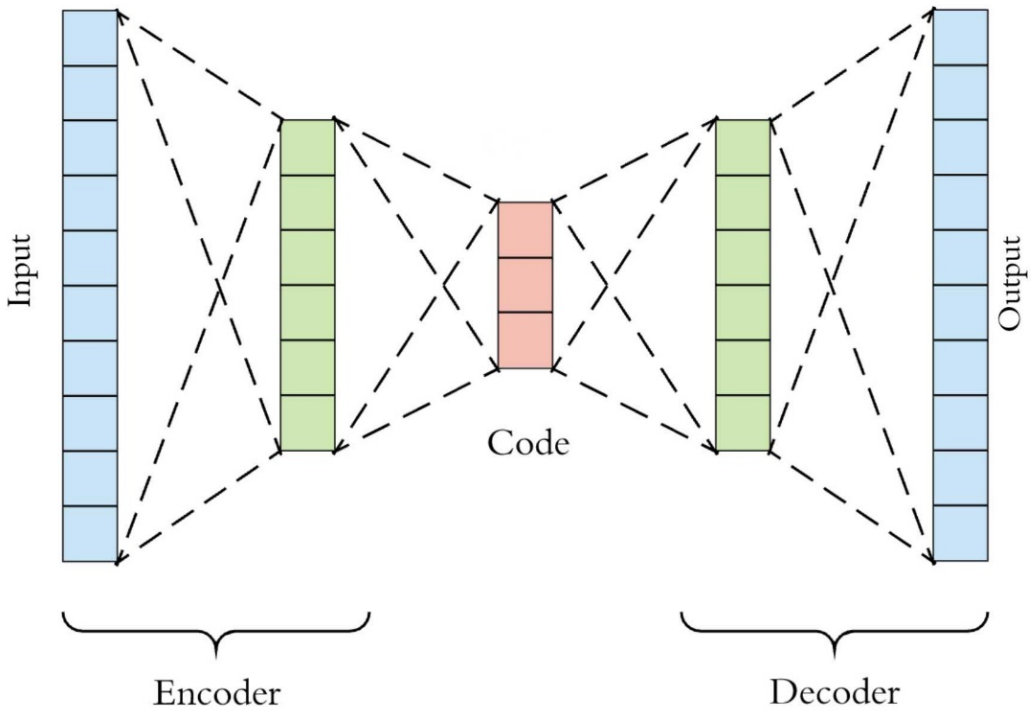


Figure 2.10. LSTM Auto-encoder Model.

The LSTM autoencoder uses two layers for compressing the input data and then two layers for decoding the data and reconstructing it [10]. The LSTM Autoencoder is a useful algorithm for data that is represented as a series. For this intrusion detection system it is particularly useful for detecting malicious behavior in the time series of voltage behavior. Each example for the autoencoder is not merely a voltage point but a voltage series of data points, and as a result the LSTM autoencoder is able to detect more complex patterns than other neural network algorithms that are not able to be trained through time series data. The autoencoder is then used to classify the input behavior as normal or malicious. This is done by characterizing loss. Since the autoencoder is trained to reproduce data with low loss, when trained on normal data one can detect anomalies simply by checking the loss value. If an arbitrary input time series is reconstructed with low loss, one may consider this to be normal behavior as the model, which has seen normal behavior, recognizes it quite well. However, if the loss is much higher than it typically is when processing a normal sequence, it can be concluded that this particular sequence is anomalous. However, it is important to note that the LSTM Autoencoder is a powerful and ubiquitous method that can be used in multiple ways to perform the intrusion detection; the aforementioned approach is simply one of several possible approaches. Users may adapt how the system uses the losses or learned features of the LSTM Autoencoder to classify the data according to their needs.

2.4 References

[1] J. Zhan, J. Huang, L. Niu, X. Peng, D. Deng and S. Cheng, "Study of the key technologies of electric power big data and its application prospects in smart grid," 2014 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC), Hong Kong, 2014, pp. 1-4.

[2] B. Networks, “Barracuda Networks,” *What is an Intrusion Detection System? | Barracuda Networks*. [Online]. Available: https://www.barracuda.com/glossary/intrusion-detection-system. [Accessed: 01-Jul-2021].

[3] H. Mahmood and H. Mahmood, “Application Threat Modeling using DREAD and STRIDE,” *Haider Mahmood Infosec Blog*, 04-Nov-2017. [Online]. Available: https://haiderm.com/application-threat-modeling-using-dread-and-stride/. [Accessed: 02-Jul-2021].

[4] Microsoft, The STRIDE Threat Model, Nov 2009. [Online] Available: https://docs.microsoft.com/en-us/windows/deployment/windows-autopilot/add-devices

[5] C. S. R. C. C. Editor, “authenticity - Glossary,” *CSRC*. [Online]. Available: https://csrc.nist.gov/glossary/term/authenticity. [Accessed: 01-Jul-2021].

[6] C. S. R. C. C. Editor, “integrity - Glossary,” *CSRC*. [Online]. Available: https://csrc.nist.gov/glossary/term/integrity. [Accessed: 01-Jul-2021].

[7] D. Swinhoe, “What is a man-in-the-middle attack? How MitM attacks work and how to prevent them,” *CSO Online*, 13-Feb-2019. [Online]. Available: https://www.csoonline.com/article/3340117/what-is-a-man-in-the-middle-attack-how-mitm-attacks-work-and-how-to-prevent-them.html. [Accessed: 02-Jul-2021].

[8] I. by LucidView, “Why behaviour-based IDS/IPS is more effective than traditional signature-based IDS/IPS,” *ITWeb*, 04-Sep-2019. [Online]. Available: https://www.itweb.co.za/content/kYbe9MXxXomMAWpG. [Accessed: 01-Jul-2021].

[9] By: IBM Cloud Education, “What are Neural Networks?,” *IBM*. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks. [Accessed: 01-Jul-2021].

[10] J. Brownlee, “A Gentle Introduction to LSTM Autoencoders,” *Machine Learning Mastery*, 27-Aug-2020. [Online]. Available: https://machinelearningmastery.com/lstm-autoencoders/. [Accessed: 01-Jul-2021].

[11] S. Prabhakaran, “ARIMA Model - Complete Guide to Time Series Forecasting in Python: ML+,” *Machine Learning Plus*, 14-Jun-2021. [Online]. Available: https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/. [Accessed: 16-Jul-2021].

[12] M. Grogan, “ARIMA vs. LSTM: Forecasting Electricity Consumption,” *Medium*, 16-Dec-2020. [Online]. Available: https://towardsdatascience.com/arima-vs-lstm-forecasting-electricity-consumption-3215b086da77. [Accessed: 16-Jul-2021].

[13] *ARIMA/Sarima and LSTM time series data integration learning - Programmer Sought*. [Online]. Available: https://www.programmersought.com/article/64975561621/. [Accessed: 16-Jul-2021].

CHAPTER 3

TEST SETUP AND EXPERIMENTAL RESULTS

3.1 Introduction

The purpose of this section is to detail the process of training the neural network algorithm to be able to detect malicious behavior and then deploy the algorithm onto hardware to run on real-time data. The procedure and the tools used to collect the data, process it, and feed it into the model will be explained in this section. The implementation of the graphical user interface that is used to monitor the data will also be explained. Additionally, the process of designing and simulating the various datasets to train the neural network algorithm for optimal detection of malicious behavior will be explained in this chapter, along with the training process itself. Lastly, the procedure for saving the model and successfully deploying it onto the hardware such that it is able to run in real time will be explained along with the libraries and dependencies needed for implementing the intrusion detection module.

3.2 Data Collection Process

As mentioned earlier in the previous section, in order to implement an artificial neural network algorithm, adequate data needs to be collected and processed before the data is fed to the model. For the IDS to continue to update and improve, the data collection and preparation process is a crucial piece of the overall pipeline. The communication layer consists of all the necessary components necessary to collect the data because this layer is responsible for managing and processing the data. In particular, the Raspberry Pi serves as the core component of the communication layer and since the data is collected here, this is where the intrusion detection module exists. There are various communication and protocols that can be implemented when designing the communication link. For this research, the Modbus TCP protocol is implemented for the communication from the DSP to the Raspberry Pi. Modbus TCP is a protocol used to assign ways of managing data between various layers. It is important to have a protocol so that the data is transmitted in a consistent manner. The communication path and protocol of the inverter for this reference design can be seen in the illustration in Figure 3.1. The controller at the control layer is flashed with DSP code in C, and it will obtain the values from the power-electronics layer and communicate them back to the communication layer that will then process the data and display it on the GUI. Sending commands from the Raspberry Pi works in a similar manner in that the user would write values to a script running on the computer interface, which values will then be sent to the designated registers in the control layer. Finally, it will update the values at the hardware layer.

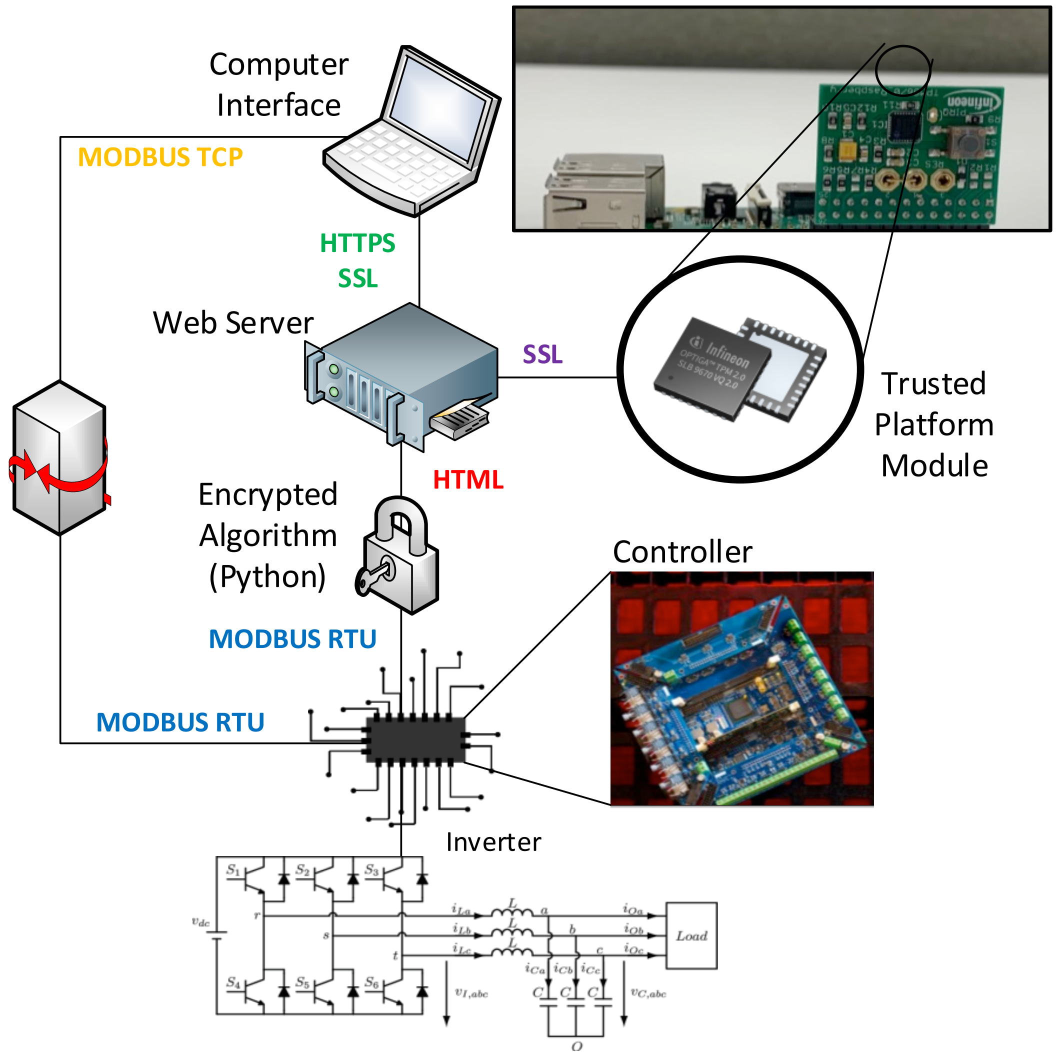
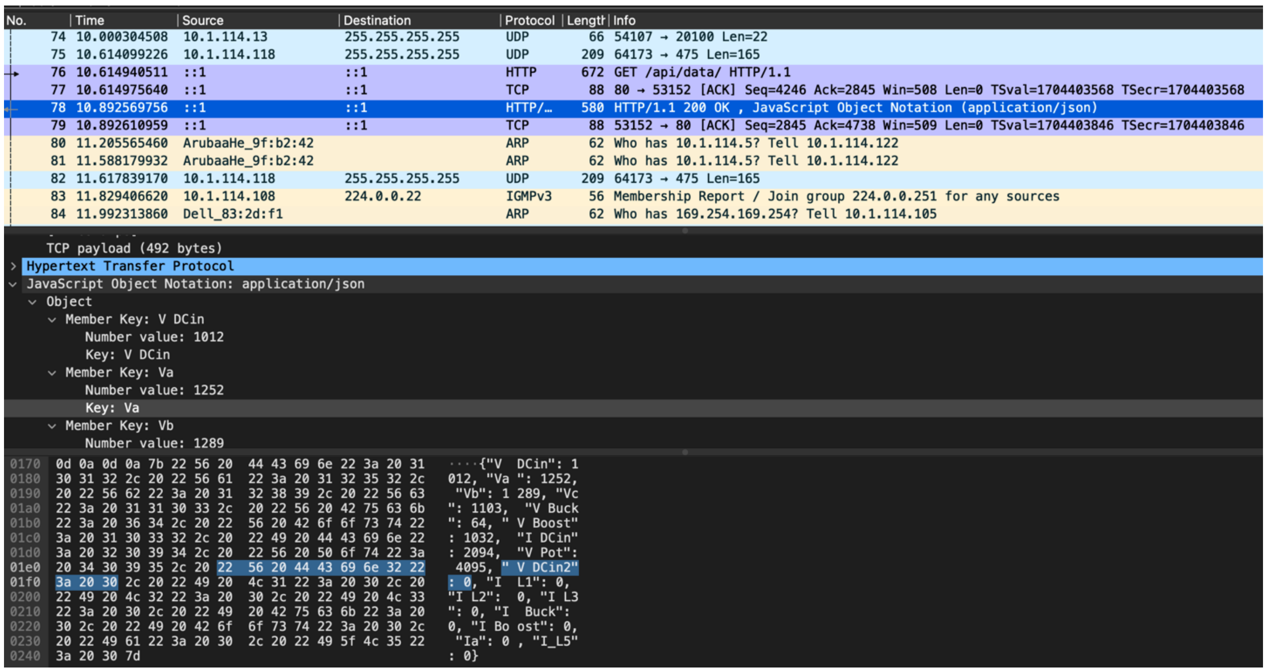


Figure 3.1. Communication path diagram.

There are a variety of packet sniffing tools that can be used to collect data. For this research, Wireshark was used to collect the data because it is open source, well documented, and widely known in the world of information technology. A sample of the data being collected can be seen in Figure 3.2 of the Wireshark packets.



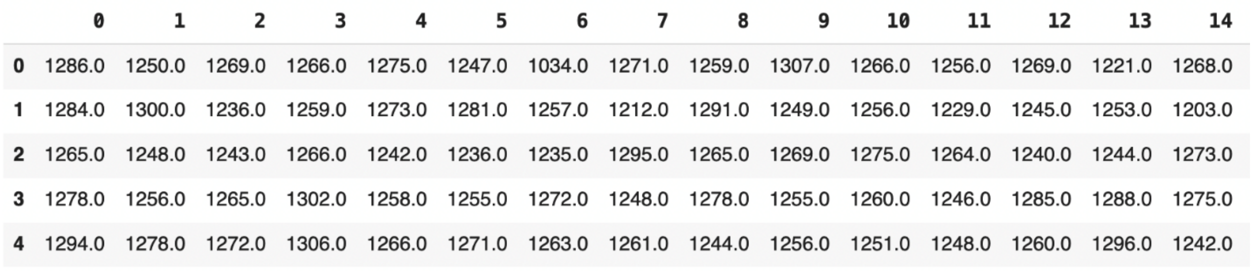
```
No.  Time          Source             Destination      Protocol Length Info
  74 10.000304508  10.1.114.13        255.255.255.255  UDP        66  54107 → 20100 Len=22
  75 10.614099226  10.1.114.118       255.255.255.255  UDP       209  64173 → 475 Len=165
  76 10.614940511  ::1                ::1              HTTP      672  GET /api/data/ HTTP/1.1
  77 10.614975640  ::1                ::1              TCP        88  80 → 53152 [ACK] Seq=4246 Ack=2845 Win=508 Len=0 TSval=1704403568 TSecr=1704403568
  78 10.892569756  ::1                ::1              HTTP/…    580  HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
  79 10.892610959  ::1                ::1              TCP        88  53152 → 80 [ACK] Seq=2845 Ack=4738 Win=509 Len=0 TSval=1704403846 TSecr=1704403846
  80 11.205565460  ArubaaHe_9f:b2:42                   ARP        62  Who has 10.1.114.5? Tell 10.1.114.122
  81 11.588179932  ArubaaHe_9f:b2:42                   ARP        62  Who has 10.1.114.5? Tell 10.1.114.122
  82 11.617839170  10.1.114.118       255.255.255.255  UDP       209  64173 → 475 Len=165
  83 11.829406620  10.1.114.108       224.0.0.22       IGMPv3     56  Membership Report / Join group 224.0.0.251 for any sources
  84 11.992313860  Dell_83:2d:f1                       ARP        62  Who has 169.254.169.254? Tell 10.1.114.105

    TCP payload (492 bytes)
> Hypertext Transfer Protocol
v JavaScript Object Notation: application/json
  v Object
    v Member Key: V DCin
        Number value: 1012
        Key: V DCin
    v Member Key: Va
        Number value: 1252
        Key: Va
    v Member Key: Vb
        Number value: 1289

0170  0d 0a 0d 0a 7b 22 56 20  44 43 69 6e 22 3a 20 31   ····{"V  DCin": 1
0180  30 31 32 2c 20 22 56 61  22 3a 20 31 32 35 32 2c   012, "Va ": 1252,
0190  20 22 56 62 22 3a 20 31  32 38 39 2c 20 22 56 63    "Vb": 1 289, "Vc
01a0  22 3a 20 31 31 30 33 2c  20 22 56 20 42 75 63 6b   ": 1103,  "V Buck
01b0  22 3a 20 36 34 2c 20 22  56 20 42 6f 6f 73 74 22   ": 64, " V Boost"
01c0  3a 20 31 30 33 32 2c 20  22 49 20 44 43 69 6e 22   : 1032,  "I DCin"
01d0  3a 20 32 30 39 34 2c 20  22 56 20 50 6f 74 22 3a   : 2094,  "V Pot":
01e0  20 34 30 39 35 2c 20 22  56 20 44 43 69 6e 32 22    4095, " V DCin2"
01f0  3a 20 30 2c 20 22 49 20  4c 31 22 3a 20 30 2c 20   : 0, "I  L1": 0,
0200  22 49 20 4c 32 22 3a 20  30 2c 20 22 49 20 4c 33   "I L2":  0, "I L3
0210  22 3a 20 30 2c 20 22 49  20 42 75 63 6b 22 3a 20   ": 0, "I  Buck":
0220  30 2c 20 22 49 20 42 6f  6f 73 74 22 3a 20 30 2c   0, "I Bo ost": 0,
0230  20 22 49 61 22 3a 20 30  2c 20 22 49 5f 4c 35 22    "Ia": 0 , "I_L5"
0240  3a 20 30 7d                                        : 0}
```

Figure 3.2. Wireshark capture.

The timestamp, protocol, packet length and information can be seen in the screenshot. Once the packets arrive into the Raspberry Pi, they are written into variables and stored in a Python script. Thereafter, they are published to the webserver through JavaScript. JavaScript is a language used for frontend web development and for formatting the data appropriately. Hence in the figure above JavaScript can be seen above the variables that are being read from the inverter, such as VDCin, Va, Vb, and other variables that are being read from the power electronics. Once the data is collected here, it is then exported into a text file. This text file will not be used to feed the model but rather it is to ensure that all the packets collected are stored in one location. Following this step, another Python script will be used to filter through the data looking for a key metric, such as Voltage A, and then store the values into a comma separated values file. This step is crucial because the neural network cannot simply be fed a text file with various packets and different types of data. The data needs to be formatted in a specific manner in order for it to be comprehensible to the machine learning model [1]. Different algorithms will require various methods of preprocessing the data and for the LSTM autoencoder used in this research, the data will be in the format of a voltage time series of a given sequence length. This means that each row of the data table will be one trend of a voltage time series, with the columns corresponding to different time steps in this series. For the purposes of how the IDS model considers data, each of these rows corresponds to one "example" of either normal or anomalous behavior to the model. It is important to note that when a single metric is being used as the input, one data point is not an example, but rather a list of data points is considered to be one example as can be seen below in Figure 3.3. If multiple metrics were being used, such as current or an additional voltage, each extra metric would increment the dimensionality of this input data. For example, for two metrics each example would be a two-dimensional data structure, and for three metrics each example would be a three-dimensional data structure, etc.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1286.0	1250.0	1269.0	1266.0	1275.0	1247.0	1034.0	1271.0	1259.0	1307.0	1266.0	1256.0	1269.0	1221.0	1268.0
1	1284.0	1300.0	1236.0	1259.0	1273.0	1281.0	1257.0	1212.0	1291.0	1249.0	1256.0	1229.0	1245.0	1253.0	1203.0
2	1265.0	1248.0	1243.0	1266.0	1242.0	1236.0	1235.0	1295.0	1265.0	1269.0	1275.0	1264.0	1240.0	1244.0	1273.0
3	1278.0	1256.0	1265.0	1302.0	1258.0	1255.0	1272.0	1248.0	1278.0	1255.0	1260.0	1246.0	1285.0	1288.0	1275.0
4	1294.0	1278.0	1272.0	1306.0	1266.0	1271.0	1263.0	1261.0	1244.0	1256.0	1251.0	1248.0	1260.0	1296.0	1242.0

Figure 3.3. Example of a dataset.

Although only 14 points are being shown in the figure, each row in total has 150 data points and that individual row is one example of what constitutes normal voltage behavior, meaning that these data points were collected when the power electronics device was running under normal operating conditions with an acceptable frequency of 50 Hz. Collecting the data, filtering it from the packets, and then storing it into a comma separated file in the correct format are the main steps involved before the data can be fed into the model. Furthermore, it is important to store the data into what is known as a Pandas DataFrame when it is read from the csv file because this is the format that the neural network system is configured to understand in this implementation [1]. Also, the larger the dataset the more accurate the model's detection results are after training. This dataset collected 94 examples over 12 hours which is still a relatively small sample size when it comes to machine learning. Ideally, there should be thousands of examples when training the model for robust results. To the end of increased performance, such data can be obtained by creating simulations based on transformations of the real data that was collected. Simulating the data is an important part of the machine learning pipeline in order to achieve the best possible overall performance. While the model was initially trained on these 94 examples, more data examples were needed for better results and also for thoroughness and robustness of the model. When simulating the data, it must be ensured that the data matches closely with the real world collected data. As a result, the data simulations were constructed based on the real world collected data. The normal operating conditions chosen for this data set range from 1100 to 1500 millivolts as can be seen in the plot below in Figure 3.4. This is real data collected from the hardware, visualized to help inform the simulation methods. A crucial component to understand is that the voltage here is scaled down as it is being tested on a power-electronics board and not a full-scale inverter in order to make testing and prototyping easier. Additionally, when deploying onto the inverter in the field, the only change would be scaling.

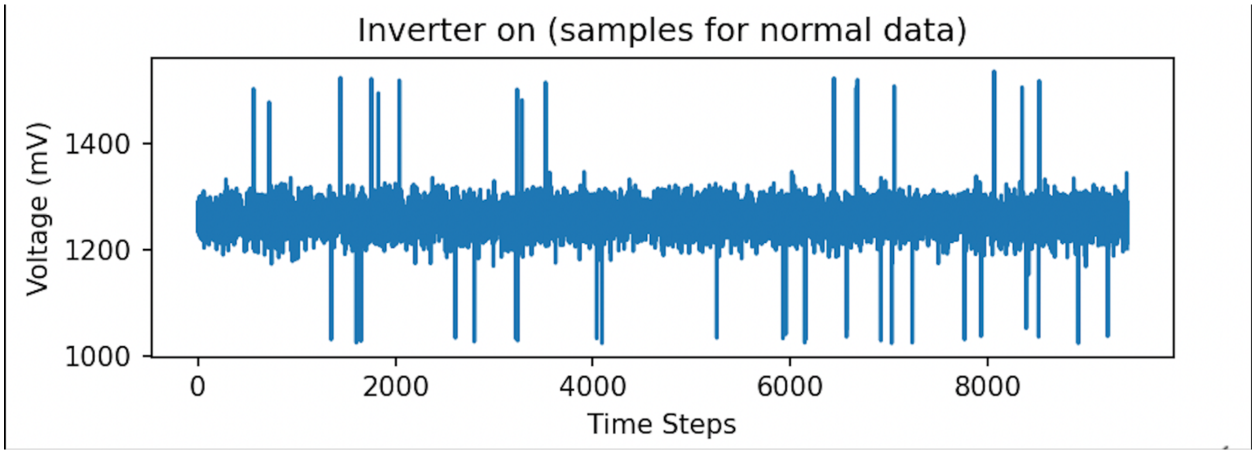


Figure 3.4. Normal collected data plot.

Furthermore, it is important to understand that whatever data is chosen to feed the inverter, that will be the data that the inverter will interpret as normal and as result, even if there are minor oscillations in the data from 1100 to 1500 mV, the neural network will be able to understand that such oscillations do not constitute malicious behavior as the data for the anomalous dataset will be different in comparison to normal data. Additionally, it can be seen that the collected data is noisy, however since the algorithm trains on the data that is collected, the noise will not negatively affect the detection. The LSTM Autoencoder algorithm implemented in this work is able to distinguish variances in data that the human eye cannot, a fact that can be leveraged by providing access to large amounts of data. Once the initial dataset has been collected it can be used to train the model, or more data can be simulated and the model can be trained with a larger dataset. Along with the normal data set, a malicious dataset was also labelled and data for it was collected as well. There are multiple reasons to collect and identify an anomalous data set in the development process. The first reason is to be able to train the model and verify that it can distinguish between what is considered normal and malicious. The second reason is to be able to run predictions on input data – known as inference -- using the model once the training is complete and use that to configure its process of using the loss values to distinguish between normal and malicious behavior [2]. Finally, these can be used to quantify the performance of the model, tracking how often it makes correct predictions. Training and testing the model will be detailed in later sections of the model; for now it is sufficient to understand that the anomalous dataset is used throughout the development process in training the model and validating its performance before deploying it onto hardware to operate on real-time data. A sample of what is considered unacceptable voltage behavior can be seen below in Figure 3.5, where data was collected from the inverter with values from the DSP which the power electronics board shut off.

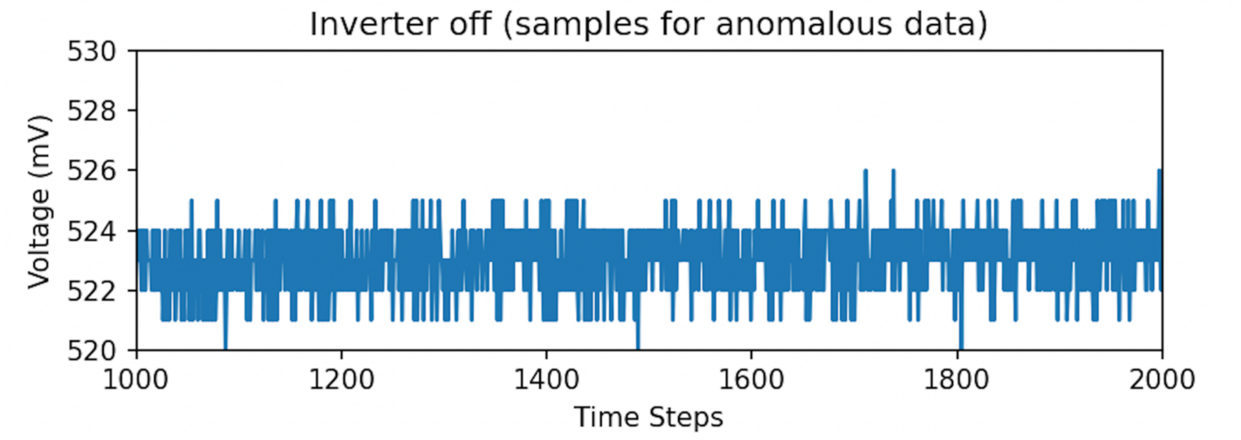


Figure 3.5. Malicious collected data plot.

This dataset is labelled as anomalous and the neural network algorithm will classify any data that matches this data to be identified as malicious. The goal here is for the algorithm to understand that the given data would not be considered as normal. An attacker may for instance, suddenly drop the voltage values below the threshold to interfere with the application of the power electronics device. This collected data is an example of one of the various examples that was used in the anomalous dataset when training and testing the model. The data simulation sets will be elaborated upon later in the following section.

3.2.1 Normal and Malicious Data Simulation

Having a large dataset is a key factor in training machine learning algorithms for deployment [3]. After collecting normal and anomalous data, the next step is to simulate data and generate larger and higher quality datasets using the collected samples. The example of the normal dataset simulation can be seen below in the plot of the data in Figure 3.6. It stays within the range of the collected normal data set that was shown in the previous section.

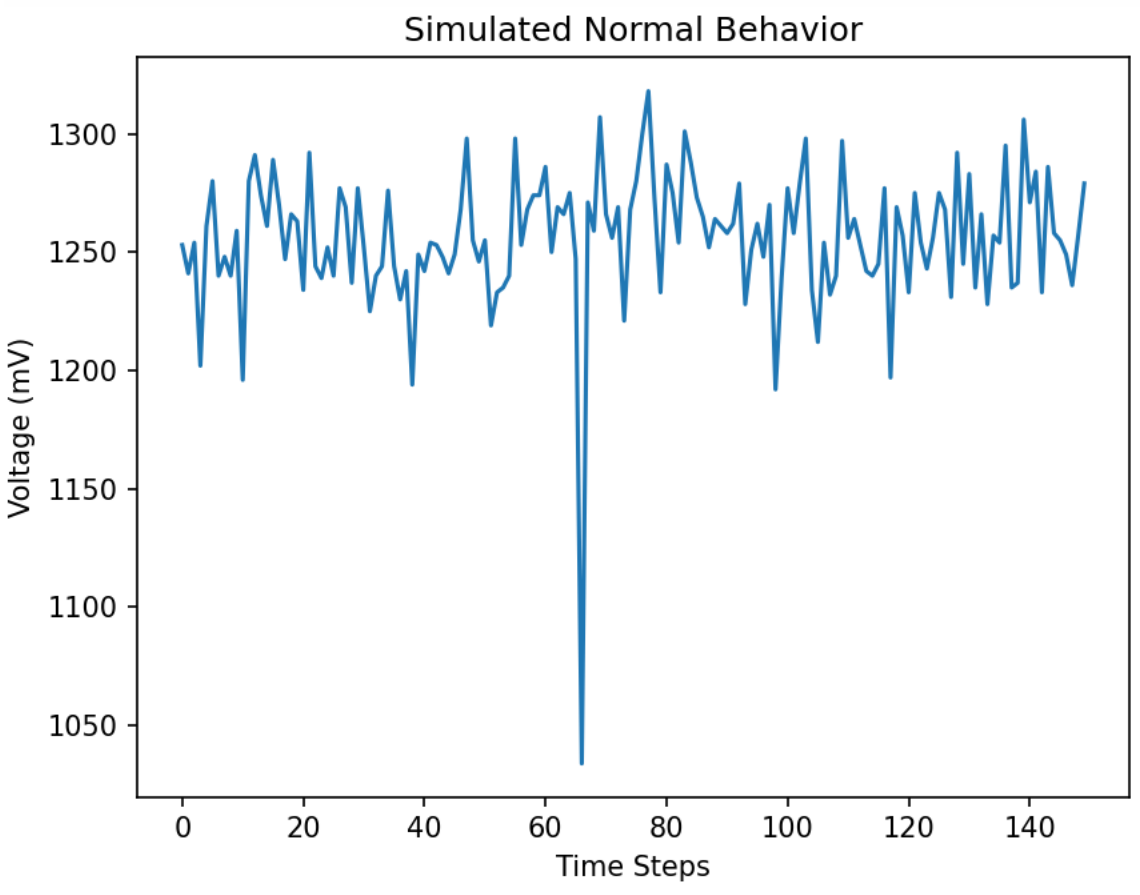


Figure 3.6. Simulated normal behavior.

This data was generated via a simulation that extrapolated from the collected data with the goal of adding a larger quantity of more diverse examples for the model to learn from. It can be compared to the collected normal data plot in Figure 3.7, and it can be seen qualitatively that the simulated dataset matches closely to the collected normal dataset.

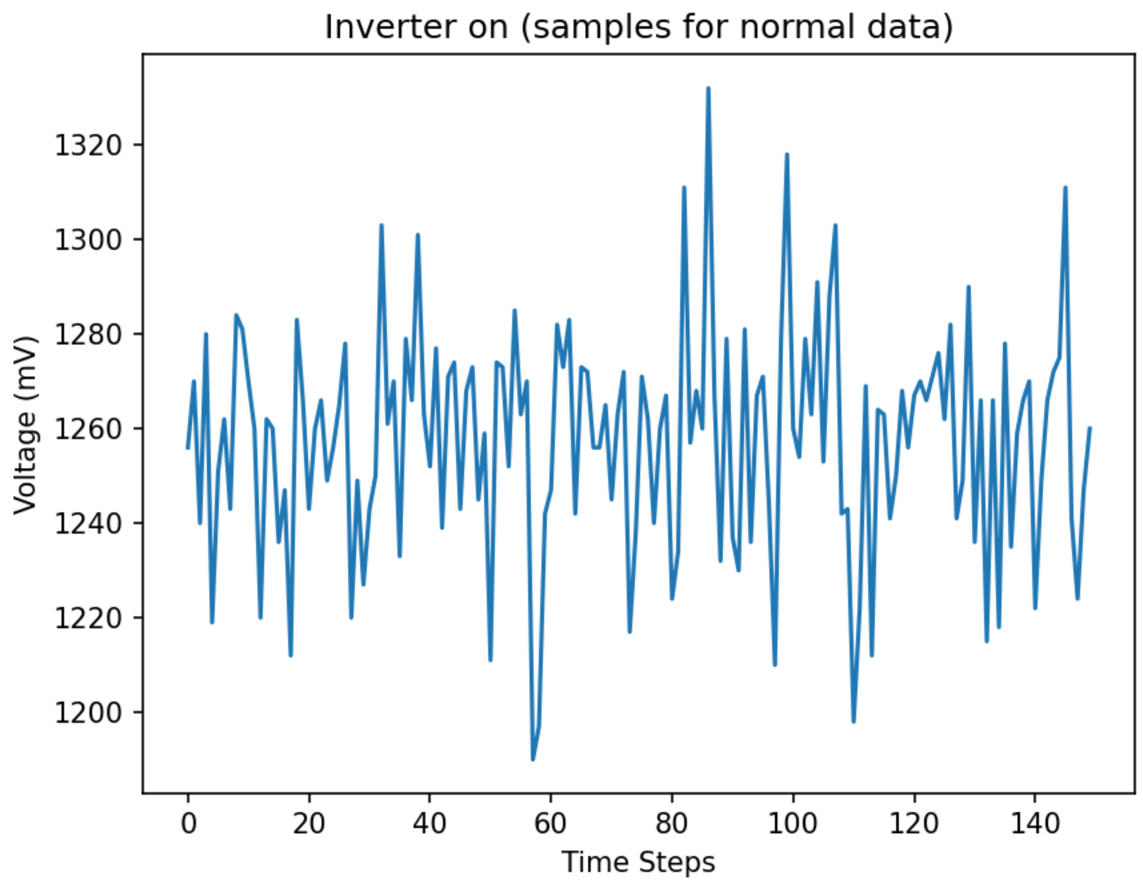


Figure 3.7. Normal collected data.

After simulating the normal datasets, multiple examples were simulated for the malicious dataset. The attack vectors considered when simulating the malicious data were mentioned in Chapter 2 and they included a denial-of-service attack, a faulty data injection attack, and a man in the middle attack.

The simulated behavior of the denial-of-service attack can be seen in Figure 3.8 below. In this scenario the voltage drops to an extremely low value that would not allow the inverter to function correctly. We simulate the behavior of the inverter if it were to suddenly turn off, dropping from its normal operation mode. In order to train the model to be able to recognize this type of attack in a diverse range of scenarios, we generate a variety of different examples in which this may occur. These examples start with normal behavior and then turn off at a random point. Moreover, the "off" behavior is a randomly sampled sequence of points from the above "off" dataset. This ensures diversity in the dataset such that the model is able to correctly recognize the behavior in all cases.

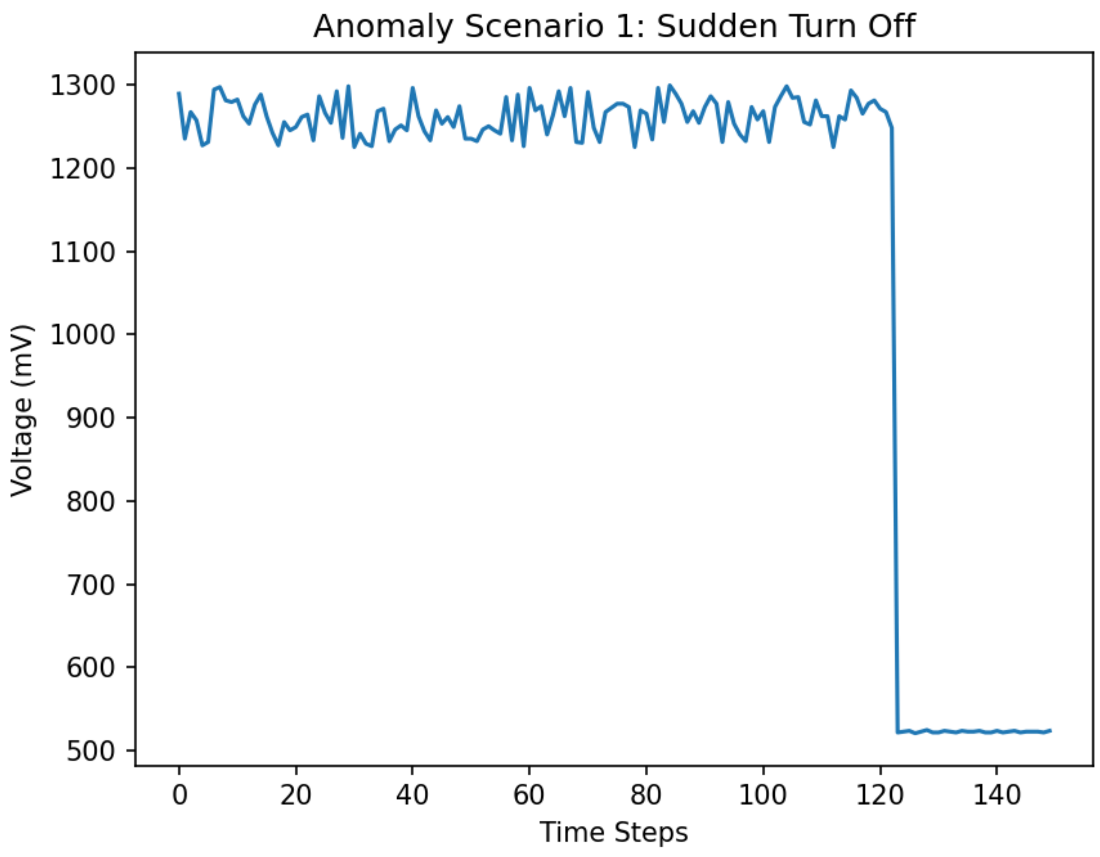


Figure 3.8. Denial of Service Attack data.

The next example for the malicious dataset considers the faulty injection data attack by writing voltage values that oscillate to high voltage values beyond the acceptable range, in this case up to 2000 mV, then such behavior would be considered malicious and can be seen in Figure 3.9. It is important to understand that both these types of attacks could result from a man in the middle data spoof attack as well, therefore such malicious behavior could result from multiple attack vectors. The examples chosen for this intrusion detection algorithm are to demonstrate the machine learning capabilities and the advantages of using behavioral based intrusion detection.

As can be seen, these types of attacks do not alter the signature of the packets, but alter the values of the data being read, hence a behavior-based detection tool is crucial [3]. This example demonstrates two different attack vectors being identified via the same method. Since this attack can happen in a variety of ways, we create a diverse dataset of scenarios by randomizing the number of spikes/oscillations, the amplitude of each one, its duration, and where they occur.

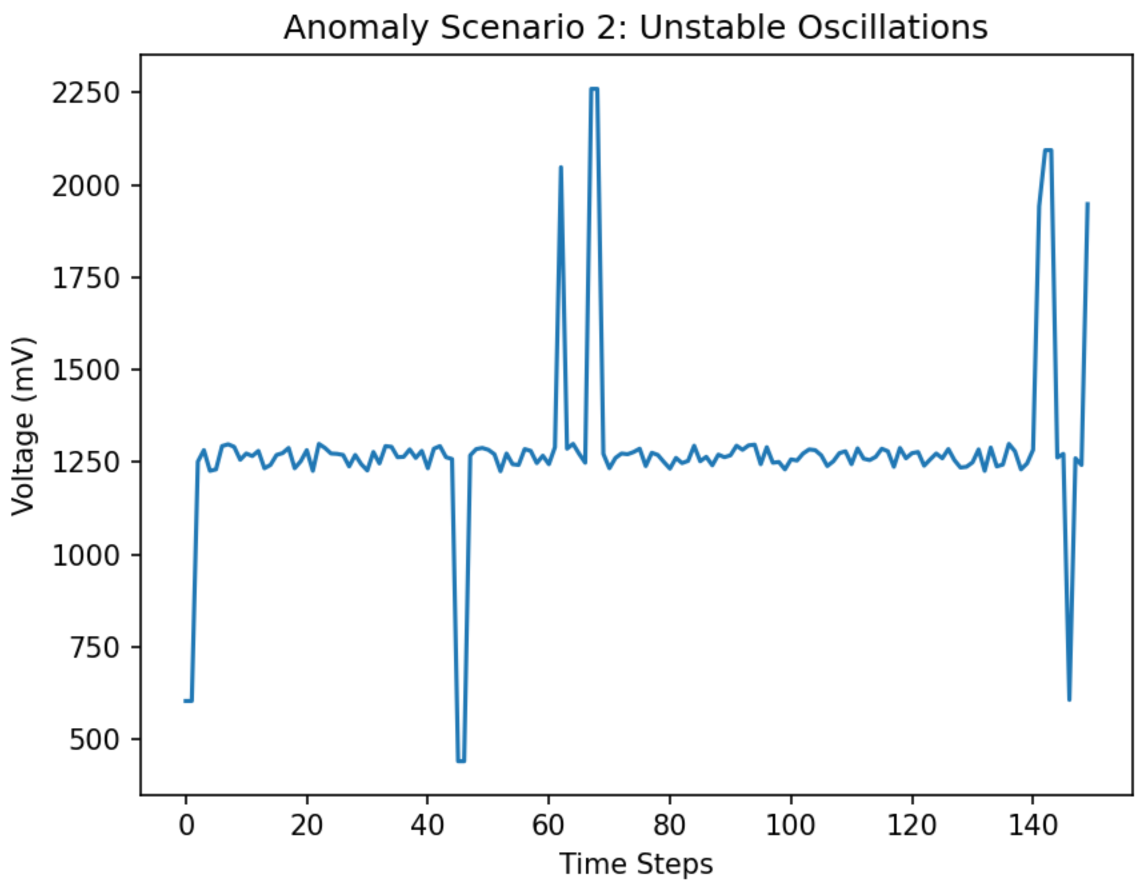


Figure 3.9. Faulty Injection Data Attack.

A faulty data injection attack could take myriad other forms. However, this form of altering the voltage to beyond the acceptable threshold demonstrates the neural network can distinguish between normal behavior and different types of malicious behavior.

The third example that was simulated considered the man in the middle data spoof attack, in which the attacker shoots the voltage far above the threshold to damage the hardware as shown in Figure 3.10. The timing and intensity of the spikes are randomized across examples in the simulated data.

Each of these attacks contribute to the number of examples in the anomalous dataset that number approximately 9000 in the largest data set and 2500 in the smaller dataset. There are other attack vectors that can be simulated as well, however these three scenarios are sufficient to demonstrate the efficacy of the artificial neural network as these examples differ from one another and can portray the algorithm's ability to be able to identify varying complex patterns.

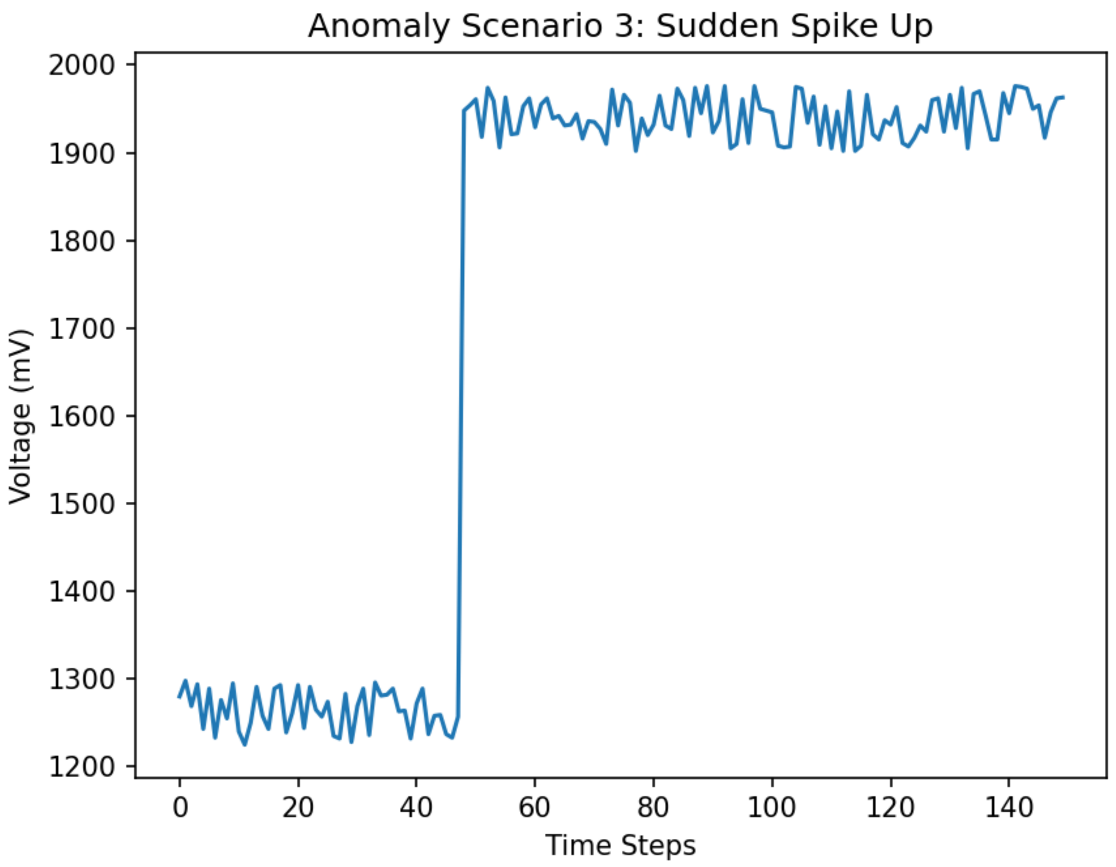
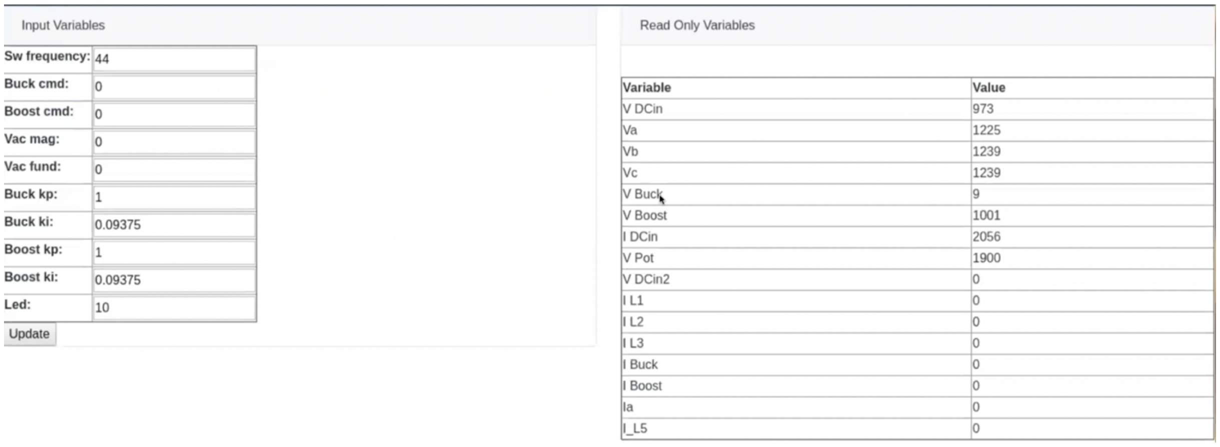


Figure 3.10. Sudden Voltage Spike Attack data.

After simulating all the various datasets, they are written into a comma separated file in order to be able to feed into the artificial neural network. There were a total of 9000 normal data examples, and 9000 anomalous data examples in the largest dataset which sums to 18000 examples. The next section will briefly explain the webserver implementation used to communicate with the inverter, as this was the primary means of monitoring and collecting data for training the algorithm. After that the process of training and testing the neural network will be explained.

3.2.2 Webserver Implementation and Overview

As mentioned in the previous section, the web server within the communication layer is a graphical user interface in which the control layer can communicate information to the computer interface. It is also a vital component of the inverter as it is used to collect the data. Running the webserver on the Raspberry Pi 4 was a choice made because of the documentation that is available as well as the compatibility of the device with Modbus TCP and Python 3. The Webserver is implemented using the Apache HTTP server that hosts the back end, with the Django web framework, which is in charge of the higher-level web design that interfaces with html and JavaScript [4]. Users can view the webserver through any modern web browsers such as Firefox, Google Chrome, and Safari. A screenshot of the web server user interface can be seen in 3.11 below.



Input Variables

Sw frequency:	44
Buck cmd:	0
Boost cmd:	0
Vac mag:	0
Vac fund:	0
Buck kp:	1
Buck ki:	0.09375
Boost kp:	1
Boost ki:	0.09375
Led:	10

Update

Read Only Variables

Variable	Value
V DCin	973
Va	1225
Vb	1239
Vc	1239
V Buck	9
V Boost	1001
I DCin	2056
V Pot	1900
V DCin2	0
I L1	0
I L2	0
I L3	0
I Buck	0
I Boost	0
Ia	0
I_L5	0

Figure 3.11. Webserver GUI.

The data that is displayed is split into two different types of variables known as read and write variables. The read only variables consist of sensory data that is not modified manually by the operator but results in a modification from the manipulation of the write variables to manage the inverter in real time. These variables consist of the switching frequency, PI gain parameters, ac frequency and amplitude, and other parameters. Additionally, the read only variables are being updated every second and this is important as the data that is collected for training the neural network is updated with that same metric. Furthermore, the method used to collect the data is adaptable to a designer who wants to collect different metrics or even additional metrics. The versatility of Django allows the web interface to be flexible to the developer as they can display data in various ways while also allowing for additional features to be added.

The two core components, Apache and Django, are critical in the implementation in order to communicate and display data to the user. Apache is an HTTP server host for modern operating systems that meets current HTTP standards [5]. There are numerous standard configuration parameters that were listed before that allow the user to interface with the inverter. Additionally, the Apache server also has an HTTPS configuration that is utilized in this system. The HTTPS configuration encrypts the communication in order to protect the user and the server communication link. The encryption algorithm used is a TLS v1.3 encryption. Although the communication is encrypted, an intrusion detection module is implemented because a threat could always occur if something malfunctions or the attacker gains access to the link through other means. As a result, a conservative approach when implementing security is preferred as opposed to a more lax one. The webserver also contains a password authentication and other features that are extraneous and beyond the scope of this research. It is sufficient to know that the webserver takes the data from Python and renders it into a format that is easily understandable through JavaScript, and those JavaScript packets are what appear when collecting data for training the Intrusion Detection Module. The data can be collected through other means such as LabView, however, such software is not necessarily compatible with the Raspberry Pi, and hence this webserver is implemented to allow for the designer to be able to collect data on the same device that the neural network will be deployed on. The next section focuses on training and testing the neural network with the collected and simulated datasets.

3.3 Implementing the Neural Network Model

Implementing the model for the neural network consists of three primary functions: the encoder function, the decoder function, and the autoencoder function. After the data has been processed it is then converted into tensors, which are multi-dimensional arrays that represent the data. These tensors are then fed into the LSTM autoencoder. The first function is the encoder function and its job is to compress the data. The arguments for the encoder function consist of a sequence length and the number of input features. In the case of this implementation the number of input features is 1, as the analysis is being performed on one metric. The sequence length is 150, as 150 voltage points were collected for one example of a voltage time series. The job of the encoder function is to compress the data and so the tensor of the data is taken and reshaped to a compressed data representation. The decoder has similar arguments in that it takes in a sequence length and a number of input features. The decoder has an additional layer that is used to decompress the data, and this is done by altering the shape of the tensor, and reconstructing the compressed data based on the input size and sequence length. The autoencoder function calls the encoder and decoder functions and passes the data through those two functions.

3.4 Training and Testing the Neural Network

Once all the data has been collected, simulated, processed into the correct format, the machine learning model now needs to be trained and tested. The dataset is split into two sections: a train dataset and test dataset, and this is done to ensure the model detects accurately. For the normal dataset, the test size was 15% of the data, which is 1350 examples, while 85% was used for training. The anomalous dataset had a test size of 33% of its data, which is 2970 examples. Typically the train dataset is larger because it requires more iterations to train the algorithm than it does to test it [6].

The next key component is the implementation of the function that trains the model, which takes in the train and test datasets and feeds it into the model to train for a certain number of epochs. Depending on the specific task, dataset, and network architecture, the requisite number of training epochs can vary. It is only by training the model multiple times that the correct epoch amount can be determined. An analysis of the various datasets and epochs will be explained in Chapter 4. For this dataset, the model was first trained with 50 epochs. For each epoch, the training function feeds the model with all the training examples in the dataset, and it then evaluates the model on the test dataset. For each individual epoch an optimizer function is used. This optimizer function is used to adjust the weights of the neural network in order to decrease the losses that incur from the model [7]. The loss function is used to calculate the prediction error that is yielded from the neural network. In this implementation, the loss function was the L1loss from PyTorch's neural network module, which computes the mean absolute error between each element in the input and target. This is well suited to a time series task that needs to extract out a relatively small set of relevant features. The optimizer used is the adaptive moment estimation optimizer, also known as Adam. This is one of the most reliable optimizers used in recent works in deep learning. The train function then stores the losses for the train and test datasets and calculates the mean of those datasets. Those values are then compared to determine which dataset yields the lowest loss result. Then the loss values of the train and test data sets are printed in order to determine how many epochs it takes to incur a lower loss value for the datasets. From Chapter 2, it is known that the goal is for the losses to decrease with each successive epoch. Once the model has completed training for the designated number of epochs it needs to be saved and it is then ready for deployment.

3.5 Deploying the Neural Network Onto the Hardware

The leading focus of this section is to detail the process of deploying the neural network onto hardware and explain the process of saving and loading different models. Not all machine learning models are compatible with every hardware device and this was one of the reasons that the Raspberry Pi was chosen for implementing the machine learning intrusion detection module. There are various libraries that are used to implement neural networks such as Pytorch or Tensor Flow. However, Pytorch was chosen because it is a widely used library that has considerable amount of documentation, it includes efficient native implementations of fundamental neural network functions that are helpful in inference, and it is useful for rapid prototyping and deployment. In order to deploy the model successfully it needs to be saved in a specific state. PyTorch has two primary functions for the saving models, one function saves the model without saving the weights and biases of the module, and another method that saves it as a state dictionary. The second method is preferred as the first version requires the same version of PyTorch to be operating on the hardware used for deployment. The state dictionary is a Python dictionary object that maps each layer to its parameter tensor [8]. Tensors are the data structure that the values of the model's parameters are converted to, in order for the model to understand the parameters. The state dictionary offers modularity as this method allows for the model to saved, updated, and altered without having to retrain a neural network on the device that it is being deployed on. This is extremely beneficial and can save time when training a model as engineers could train the model on more powerful devices and can then deploy them onto the communication layer of the inverter for quicker deployment because training models is taxing on the CPU and is time dependent. For larger datasets, training the model for each epoch takes significantly longer. Using the PyTorch save state approach allows for rapid prototype deployment of neural network algorithms. Once this model is saved, it is then loaded onto the Raspberry Pi with the load state dictionary function. After the model is loaded, it is then time to feed the model data in real time and run the predict function on it. This is the core implementation of the intrusion detection module. The data is read from the register and is then appended to a list, reshaped for use in the model, and then fed in sequentially in sequences of 150 points each, the length of each example in training. It is then fed into the predict function. The predict function will calculate the losses incurred by the model for each entry of the data. A threshold value is chosen to determine if the behavior is anomalous or malicious. The threshold value is chosen based on the losses incurred by the model and it will vary depending on the exact dataset used to train the model. The correct threshold value is determined by the amount where the losses begin to decrease rapidly. If the loss is less than the threshold then the data is considered normal, otherwise it is identified as malicious and the script will flag the behavior as such. When the neural network is running in real time, the normal data trends will correspond to a certain loss value and the malicious data trends will correspond to a particular loss value, and the threshold value distinguishes between the two. At this point, the intrusion detection algorithm is running on hardware and is monitoring the data and using this configuration to distinguish anomalous behavior from normal behavior.

3.5 References

[1] P. Mahto, "PANDAS for Machine Learning," *Medium*, 16-Sep-2020. [Online]. Available: https://medium.com/mlpoint/pandas-for-machine-learning-53846bc9a98b. [Accessed: 01-Jul-2021].

[2] "What is Machine Learning Inference?," *Hazelcast*, 05-Sep-2020. [Online]. Available: https://hazelcast.com/glossary/machine-learning-inference/. [Accessed: 01-Jul-2021].

[3] I. J. Sagina, "Why go large with Data for Deep Learning?," *Medium*, 24-Apr-2018. [Online]. Available: https://towardsdatascience.com/why-go-large-with-data-for-deep-learning-12eee16f708. [Accessed: 01-Jul-2021].

[4] Django Software Foundation, Django Project, July 2005. Accessed on: July 2020. [Online] Available: https://www.djangoproject.com/

[5] The Apache Software Foundation, APACHE HTTPS SERVER PROJECT, April 1995. [Online] Available: https://httpd.apache.org/

[6] G. Malato, "Why training set should always be smaller than test set," *Medium*, 06-May-2020. [Online]. Available: https://towardsdatascience.com/why-training-set-should-always-be-smaller-than-test-set-61f087ed203c. [Accessed: 01-Jul-2021].

[7] S. Kumar, "Overview of various Optimizers in Neural Networks," *Medium*, 09-Jun-2020. [Online]. Available: https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5. [Accessed: 01-Jul-2021].

[8] "Saving and Loading Models¶," *Saving and Loading Models - PyTorch Tutorials 1.9.0+cu102 documentation*. [Online]. Available: https://pytorch.org/tutorials/beginner/saving_loading_models.html#what-is-a-state-dict. [Accessed: 01-Jul-2021].

CHAPTER 4

EVALUATION OF THE NEURAL NETWORK BASED IDS

4.1 Introduction

This research focuses on implementing a behavior-based intrusion detection module using an artificial neural network by training it with various datasets and deploying it onto hardware. This design consists of an intrusion detection algorithm being implemented at the top layer of a power electronics inverter topology. This reference design is focused on the core concepts of training a neural network to detect malicious behavior and then deploying that intrusion detection algorithm onto an ARM processor. This section will explain the results of the experimental setup that examines the effectiveness of the cybersecurity approach implemented in this research. The numerous datasets with varying attack vectors that were used to train the neural network will be examined, followed by comparing and contrasting the accuracy of the model after being trained with various datasets. Also, the number of epochs used to train the model that yielded varying detection rates with different datasets will also be explored and explained. Lastly, the CPU performance costs induced by the intrusion detection algorithm will be analyzed. This design focuses on training a base intrusion detection algorithm to identify malicious behavior from varying attack vectors and deploying it onto hardware while allowing for the code to be updated without redesigning the base algorithm. Hence, the implementation is not dependent on the lower levels of the hardware and is thus a flexible overall approach.

4.2 Evaluation of the Train and Test Datasets

The primary concern once the model is trained is the accuracy of the detection algorithm. This will be used to determine whether the model needs to be trained more or it is detecting at an accurate rate, and it can be deployed. In Chapter 3 the process of training and testing the model was explained. In this section, the detection accuracy of the train and test datasets, the number of epochs it took to train them, and the value of losses that occurred for each training. The intrusion detection algorithm was trained multiple times with different datasets. For the initial training a dataset of 4500 examples was used. In this dataset 2000 examples were used to represent normal behavior while 2500 examples were used to represent the malicious behavior. This was the first dataset used to train the model and it is not the dataset that was used when deploying the model onto the hardware. This will become clear when the detection rate is discussed briefly. This dataset was trained on 100 epochs and the loss function is shown below in Figure 4.1.

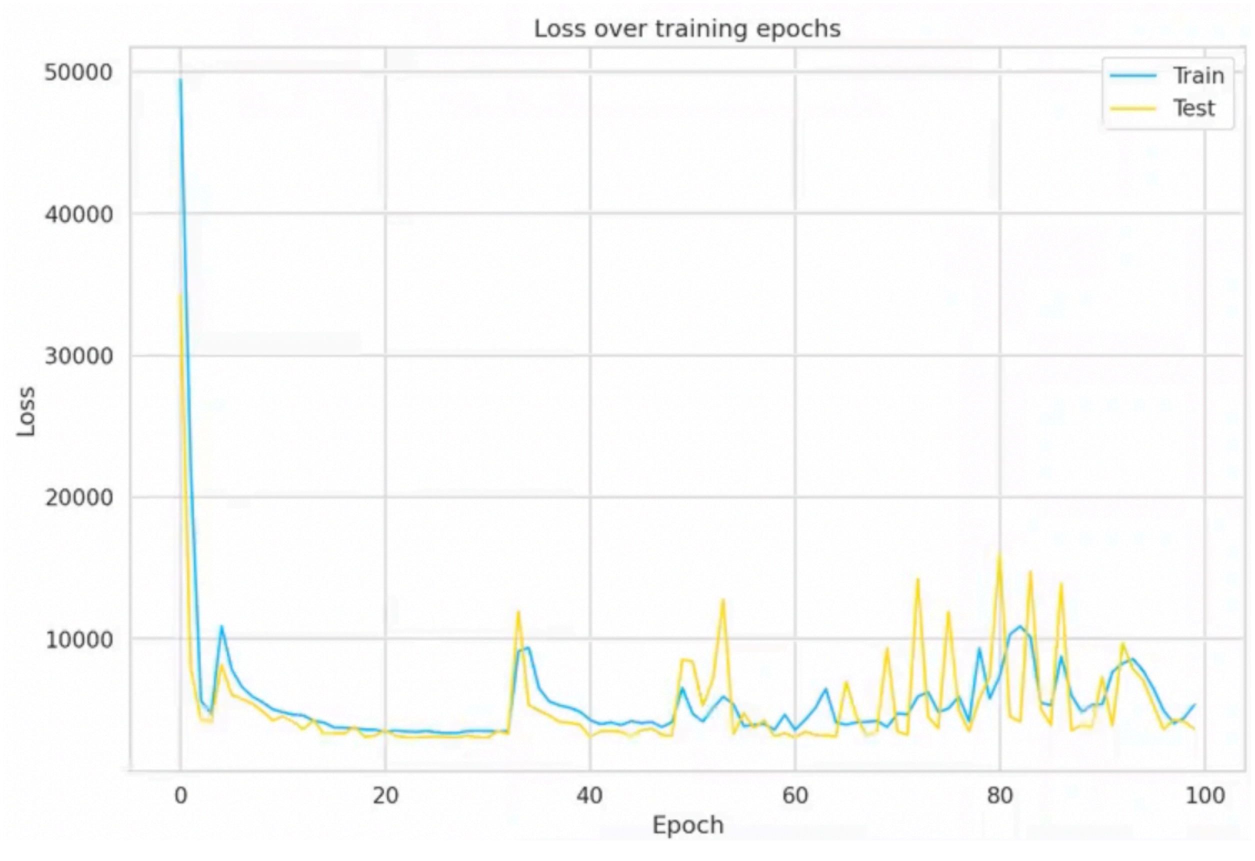


Figure 4.1. Losses over each epoch.

As it can be seen in the first few epochs of the model the losses are high initially but quickly drop throughout the training and remain relatively stable. Around the 80 epoch mark we see some signs of overfitting. Overall, however, the network converges quite well, despite the fact that this first dataset was somewhat flawed in its representation of inverter behavior. The train and test datasets both did similarly in that the losses were high in the beginning but they tapered down in towards the 100 mark.

The neural network model successfully identified the normal examples at 55% while identifying the anomalous examples at a 96% accuracy, meaning that of the total examples that were fed to the intrusion detection algorithm, 55% of them were successfully identified as normal and 96% of them were successfully identified as malicious [4]. This indicates that the model is somewhat biased towards yielding false positives in flagging attacks. While the ideal case is high detection in both, for this use case erring on the side of flagging too many things as attacks is preferable to missing actual attacks. This is because in security, a more conservative approach for additional safety is preferred. Additionally, the malicious dataset identification rate is much higher and this is because there were multiple types of examples chosen for the malicious dataset, and also the behavior was drastically different when compared to the normal dataset. The accuracy in correctly identifying normal behavior was low because this dataset did not effectively capture true normal behavior and did not have the correct, diverse data profile needed to do so. This example also illustrates the importance of using multiple metrics to evaluate the performance of the model. Seeing a high anomaly detection rate, one might be tempted to observe that the model is performing excellently at its task. However, this is misleading, as further measurement via checking the accuracy of normal predictions shows that this model is achieving that score by calling most examples anomalies instead of truly generalizing well. The next dataset and training iteration aimed to correct this issue.

The next dataset we will look at is a dataset that uses only real collected data for the normal behavior case and a more representative simulation protocol for the anomalous case. As can be seen in Figure 4.2, this dataset was trained for about 200 epochs. Although the model takes more epochs to reach a low level of loss, performance on both the train and test datasets converge to low loss quite smoothly. The loss decreased at a much steadier rate, indicating that the patterns took more training to learn, which is indicative of a more representative dataset of normal behavior. Moreover, training and test loss align very closely, indicating that the model performs almost exactly the same on data it has never seen before as on the data it has seen. This means that the model has converged to a very robust general understanding of the task, but also implies that the model can handle more complex tasks than the one it was given as well, since it was able to perform this one relatively easily with the given data. Additionally, for this dataset the model identified normal behavior at an accuracy of 98% and it identified malicious behavior at an accuracy of 98%, meaning of all the examples provided of normal behavior and malicious behavior, it identified 98% of the examples successfully, which is an accuracy that is much higher than what was yielded by the previous dataset. Furthermore, the previous dataset was largely simulated and was not based on a large amount of real time collected data. The collected data in this dataset, on the other hand, was obtained over-night, whereas the previous dataset was simulated based on a smaller number of datapoints obtained from a different device. As a result, the importance of collecting real time data and simulating data based on that to then train the model is a key factor in yielding accurate detection. This shows the stark effect of data on the performance of the algorithm, since this drastic improvement was achieved simply by more carefully having the data represent the desired task.

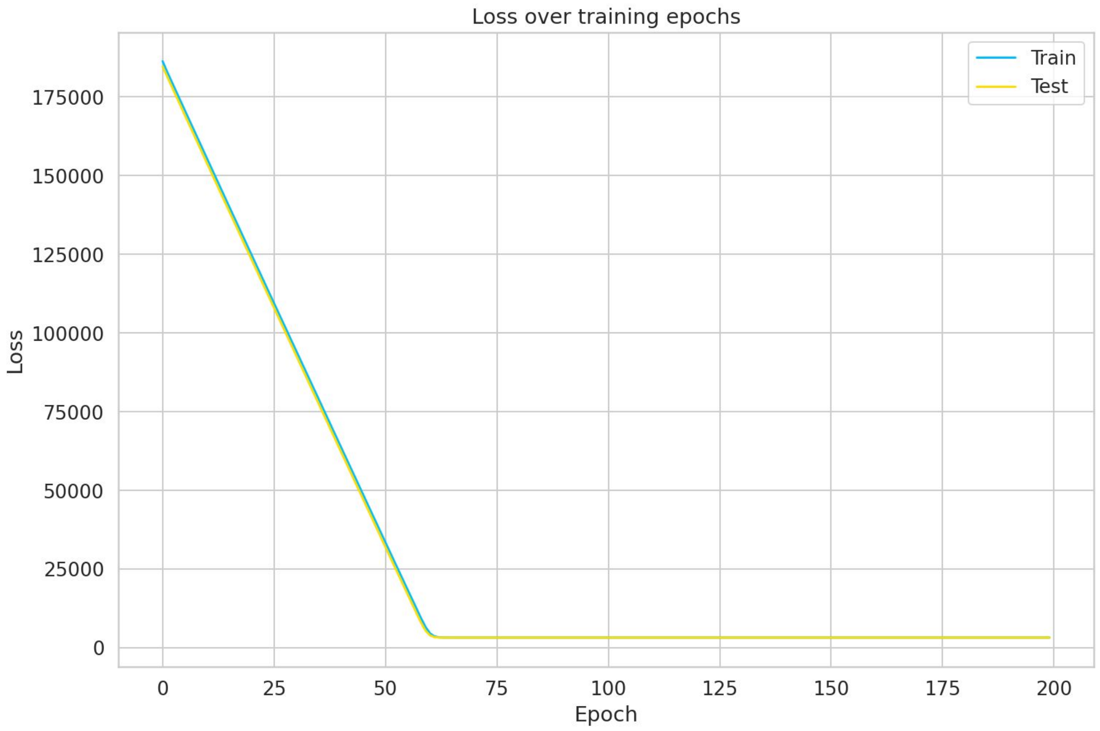


Figure 4.2. Losses over each epoch for improved dataset.

To further test, a more difficult dataset was created in which the simulated behavior for both the normal and attack scenarios were more complex. In this dataset, the simulated normal behavior was more varied and included positive and negative amplitude offsets of differing amounts to represent normal variations in a photovoltaic inverter, perhaps due to weather or other conditions that could elicit such effects.

This additional difficulty in the dataset was offset by greatly increasing the amount of training data from hundreds of examples to thousands. 4000 samples of each case were included in this dataset. The resulting loss-epoch curve is shown below in Figure 4.3.

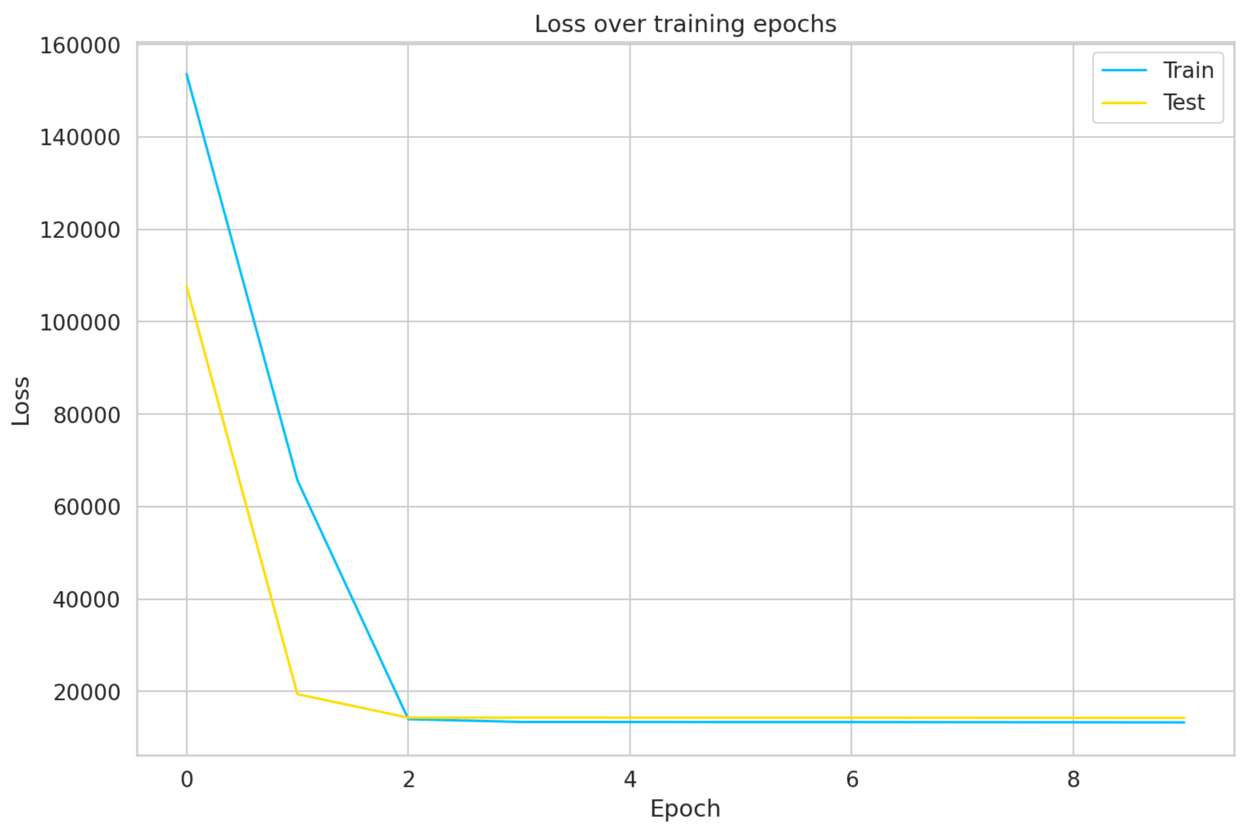


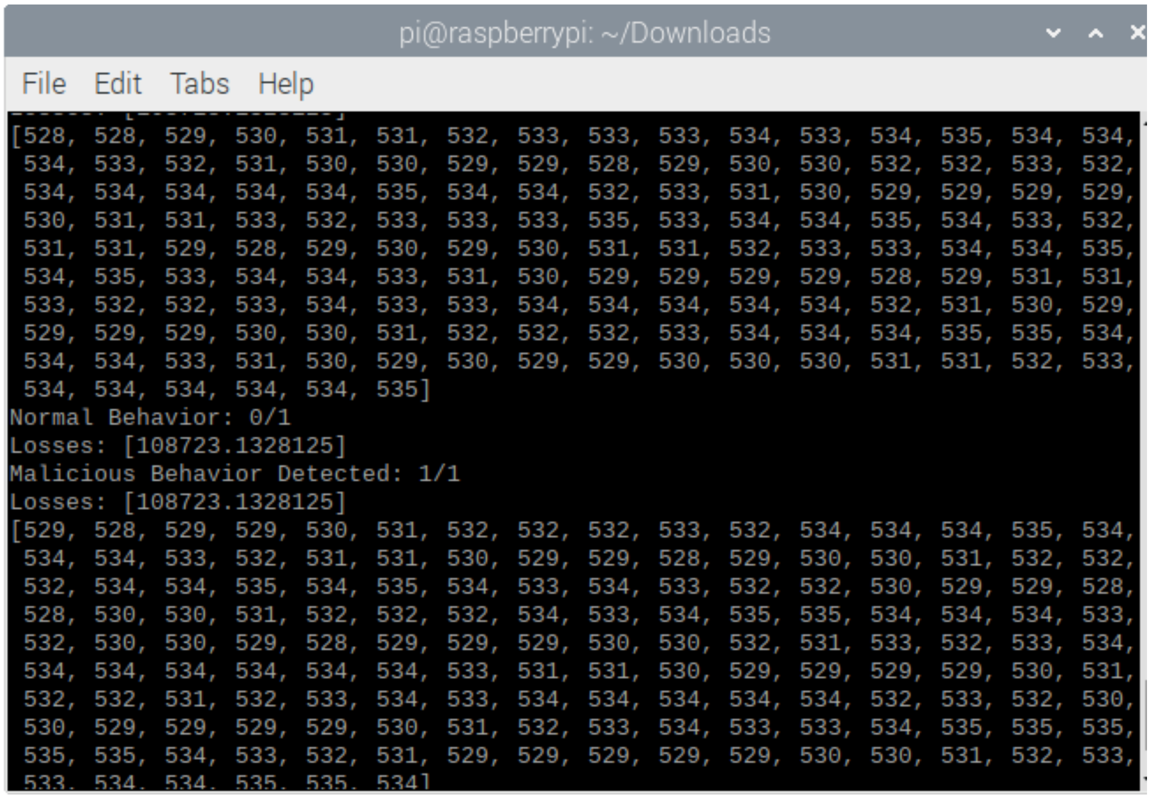
Figure 4.3. Losses over each epoch for first dataset.

With this drastic increase in the dataset size, the model was able to handle even the more complex behavior relatively easily, again reaching 98% accuracy as it did on the easy dataset. This shows that the model is able to learn more complex features and compensate for more difficult tasks simply by simulating more data, provided this simulated data is based on and closely matches the real data. Another point to note is that even in this case, it converges within only a few epochs of training.

This again hint at the algorithm's potential to identify more complex behavior. Some of the reasons this dataset performed better is that the dataset is much larger, so the model has more examples to learn from. Additionally, the normal examples numbered 4000 and the malicious examples numbered 4000, hence both examples are symmetrical, so this may have contributed to a similar detection rate by eliminating bias from statistical factors relating to sample size. Lastly, the epochs required for convergence were less, although each epoch took longer to run. This is expected as the model needs less cycles to decrease losses because the dataset contains many more examples. These results indicate the power of the model to learn more sophisticated behavior and features in identifying diverse attacks, making it encouraging for further development, as will be discussed in Chapter 5.

4.3 Results of the Attack Simulated On the Deployed Model

Once the neural network was deployed onto the hardware, an attack was simulated in order to verify the validity of the detection model. While the model was already validated extensively before deployment based on malicious datasets, ensuring that neural networks maintain their same performance when deployed on hardware for real-time inference is in general a non-trivial task. This attack was executed in real time to portray the model successfully identifying the malicious behavior. A man in the middle data spoof attack was simulated where the voltage was dropped below the acceptable range for normal operating conditions. As can be seen in the Figure 4.4. below the algorithm successfully identified that the voltage behaved in a manner fulfilling one of the anomalous cases it was trained to identify, i.e. a sudden voltage drop below the acceptable threshold to the off state and successfully flagged it as malicious. This indicates that its performance as evaluated rigorously on the anomalous datasets in the training environment transferred over to its real time performance.



```
[528, 528, 529, 530, 531, 531, 532, 533, 533, 533, 534, 533, 534, 535, 534, 534,
 534, 533, 532, 531, 530, 530, 529, 529, 528, 529, 530, 530, 532, 532, 533, 532,
 534, 534, 534, 534, 534, 535, 534, 534, 532, 533, 531, 530, 529, 529, 529, 529,
 530, 531, 531, 533, 532, 533, 533, 533, 535, 533, 534, 534, 535, 534, 533, 532,
 531, 531, 529, 528, 529, 530, 529, 530, 531, 531, 532, 533, 533, 534, 534, 535,
 534, 535, 533, 534, 534, 533, 531, 530, 529, 529, 529, 529, 528, 529, 531, 531,
 533, 532, 532, 533, 534, 533, 533, 534, 534, 534, 534, 534, 532, 531, 530, 529,
 529, 529, 529, 530, 530, 531, 532, 532, 532, 533, 534, 534, 534, 535, 535, 534,
 534, 534, 533, 531, 530, 529, 530, 529, 529, 530, 530, 530, 531, 531, 532, 533,
 534, 534, 534, 534, 534, 535]
Normal Behavior: 0/1
Losses: [108723.1328125]
Malicious Behavior Detected: 1/1
Losses: [108723.1328125]
[529, 528, 529, 529, 530, 531, 532, 532, 532, 533, 532, 534, 534, 534, 535, 534,
 534, 534, 533, 532, 531, 531, 530, 529, 529, 528, 529, 530, 530, 531, 532, 532,
 532, 534, 534, 535, 534, 535, 534, 533, 534, 533, 532, 532, 530, 529, 529, 528,
 528, 530, 530, 531, 532, 532, 532, 534, 533, 534, 535, 535, 534, 534, 534, 533,
 532, 530, 530, 529, 528, 529, 529, 529, 530, 530, 532, 531, 533, 532, 533, 534,
 534, 534, 534, 534, 534, 534, 533, 531, 531, 530, 529, 529, 529, 529, 530, 531,
 532, 532, 531, 532, 533, 534, 533, 534, 534, 534, 534, 534, 532, 533, 532, 530,
 530, 529, 529, 529, 529, 530, 531, 532, 533, 534, 533, 533, 534, 535, 535, 535,
 535, 535, 534, 533, 532, 531, 529, 529, 529, 529, 529, 530, 530, 531, 532, 533,
 533, 534, 534, 535, 535, 534]
```

Figure 4.4. IDS detection screenshot.

Furthermore, as explained in Chapter 3 the algorithm is not merely looking at one data point but a trend of the data. Once the behavior was altered, it flagged the behavior as malicious. Another point to note is that the losses are high, over 100,000. This means that the behavior detected does not fall within the loss value that corresponds to the normal behavior. The threshold for normal behavior was around 8000, and the loss value shown here is much greater than that and hence an intrusion had occurred. An additional point to note, is that the malicious behavior conducted was the result of a man in the middle data spoof attack, however this behavior could also result from a faulty data injection attack. The benefit of using a behavior-based artificial neural network for intrusion detection, is that it can identify newer types of malicious behavior that could result from varying attacks because such attacks would not match normal behavior [1].

4.4 Hardware Usage of the Artificial Neural Network Algorithm

Like any algorithm that uses software there will be trade-offs to consider when implementing the algorithm on real-time data on the selected ARM processor. In the field of cybersecurity it is also important to consider the impact of the specific solution on the system. It is important that the algorithm chosen does not hinder the other tasks necessary for the module where neural network is implemented. There are different libraries that support machine learning algorithms and different types of ARM processors that are compatible with those algorithms [2]. The top layer of the inverter consists of the Raspberry Pi 4 that has 8 GB of RAM, that runs a webserver with apache as its host, Django as the implementation of the GUI, and it also runs the machine learning intrusion detection module using Python. The CPU usage of the incurred by the neural network ranged from 60%-79.5% with a memory usage of 1.8%. The intrusion detection script itself was running and it used 0.14 GB of RAM. Machine learning algorithms are much more CPU intensive than conventional code and this is because of the analysis that is run on the data and analytics the algorithm performs in order to make accurate predictions [3]. However, the added benefit is that such algorithms can detect malicious behavior that does not alter the signature of the packets. With the current implementation, the Raspberry Pi is not taxed heavily as only two primary scripts are running and the algorithm only uses 0.14 GB of RAM, and thus this algorithm can be implemented without extensively taxing the RAM of the ARM processor.

4.5 References

[1] I. by LucidView, “Why behaviour-based IDS/IPS is more effective than traditional signature-based IDS/IPS,” *ITWeb*, 04-Sep-2019. [Online]. Available: https://www.itweb.co.za/content/kYbe9MXxXomMAWpG. [Accessed: 01-Jul-2021].

[2] “Best Python libraries for Machine Learning,” *GeeksforGeeks*, 23-Aug-2019. [Online]. Available: https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/. [Accessed: 01-Jul-2021].

[3] *Hardware Requirements for Machine Learning*, 11-Dec-2019. [Online]. Available: https://www.einfochips.com/blog/everything-you-need-to-know-about-hardware-requirements-for-machine-learning/. [Accessed: 01-Jul-2021].

[4] Yatish Dubasi, Ammar Khan, Qinghua Li, and Alan Mantooth, "Security Vulnerability and Mitigation in Photovoltaic Systems." In 2021 IEEE 12th International Symposium on Power Electronics for Distributed Generation Systems (PEDG). PEDG, 2021.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This intrusion detection system using neural networks design is developed to assist power electronics engineers and computer engineers in developing a ubiquitous intrusion detection system that uses machine learning to detect malicious behavior of several kinds at the inverter level. Detecting cyberattacks is of paramount importance for being able to develop sound mitigation strategies for threats. With the advancement of technology, it is important to be able to employ intrusion detection methods that are efficient at detecting malicious attacks resulting from different types of threats as well as addressing the need of being able to detect newer threats that could be discovered in the future. This design proposes a method to detect cyberattacks effectively by training a recurrent neural network to detect attacks based on various datasets that encapsulate normal and malicious behavior, deploying the neural network algorithm onto hardware, and then being able to update the algorithm to allow for future features to be integrated. The LSTM-Autoencoder algorithm was chosen over other mentioned algorithms due to its highly accurate detection rate, ability to be expanded as it can take additional features, and it is an algorithm that does not need to be retrained for newer attacks. Although the LSTM autoencoder requires larger datasets for training in comparison to other algorithms, its higher accurate detection rate makes it superior as accuracy is crucial in the field of cybersecurity. Numerous attack scenarios were considered when training the model to detect malicious behavior such as man-in-the-middle data spoof attacks, denial of service attacks, and malicious data injection attacks. Once the model was trained to detect malicious voltage behavior at a high degree of accuracy, it was then deployed onto the hardware to collect data real time. The various malicious datasets used to train the model are evaluated, along with the algorithm’s detection rate, to verify that the model identified malicious examples successfully. Results showed that the model is able to scale well with identifying complex patterns, achieving up to 98% detection accuracy for normal and anomalous behavior and demonstrating the potential for being trained to analyze more and more sophisticated behavior.

In conclusion, the IDS developed has a high degree of accuracy, is adaptable to varying types of hardware, and is robust to new types of attacks. This is an important contribution because this technology can be leveraged in the field of cybersecurity with solar inverters, because the algorithm deployed does not require re-implementation for adding additional features and it is flexible to different devices. Additionally, the model is robust to new types of attacks because it learns based on normal data, and as a result, it would not need to be re-trained in the field for identifying newer attacks. This is another key benefit as a newer implementation will not be required for newer attacks, and patching the algorithm would only be done for edge cases, adding additional features, and optimization. This work sets up a pipeline for future researchers to forego concerning themselves with the hardware, but rather focus on building more sophisticated intrusion detection without having to deconstruct the base implementation and exploring additional possibilities for improving the detection module.

5.2 Recommendations and Future Work

This reference design was developed as a base intrusion detection module for integrating a machine learning algorithm with a grid connected device. Potential recommendations can be implemented with this base algorithm for future work. Currently, the neural network algorithm takes in one metric and detects malicious behavior based on that metric. Additional metrics such as different voltages or current could be collected and added to the module to expand upon the intrusion detection capabilities without having to change the base algorithm. With the modular design, this would be relatively straightforward to implement. The model would simply be initialized with a higher number of input features, which comes down to changing a single variable in the code. One would then collect the data for the additional metrics, shape it into tensor form, and feed it into the model. The training process would be the same as described in Chapter 3. Finally, after completing training, inference could be performed by using the losses to classify anomalous and normal behavior just as described in this work. The main challenge would be to collect enough diverse data to encapsulate all of the different ranges of normal behavior for the additional metrics.

Additionally, more data could be collected and simulated to both better cover normal modes of operation as well as account for newer attack scenarios in the field to continually improve as cyberattacks evolve to surmount existing security measures. This could be beneficial as the intrusion detection algorithm could be expanded for the grid connected device without having to make modifications for the current topology of the inverter. Collecting data in a variety of different conditions that would affect the photovoltaic inverter would also make performance more robust in a real-world scenario. For example, if data from somewhat abrupt voltage drops -- such as from a cloud passing over the solar inverter, for example – is incorporated into the normal dataset, the model could learn to distinguish these voltage drops from malicious attacks using its powerful pattern recognition and feature extraction abilities. Another approach to this method is to consider using a seasonal model for the data collection, where the normal data consists of varying examples for each season. The same algorithm could be used, but the only difference would be the normal dataset would have examples from each season. Alternatively, the SARIMA model could be used but this may drop accuracy however the trade-off would be that the algorithm itself has inputs for seasonal data. This approach could be explored and compared to the current approach to see which process is more optimal. The LSTM has high accuracy and would only need to be retrained with data from different seasons, whereas the SARIMA might have lower accuracy but will have the inputs for seasonal data.

Another recommendation for future work would be to expand the current training and deployment pipeline to automatically update the model. Training could be hosted on an online server, which the device could communicate with. As the model runs inference, it could collect the data and periodically send it to this server, where the model could be trained on the cloud with cloud computing resources. Upon completion of training, the Raspberry Pi could pull the new parameters from the updated model and use this better-trained model for inference as well. In this way, the IDS could iteratively improve in an automated manner. Another possibility is that this could exist in the digital twin. Such a feat has not been done before, but it can be explored and researched. However, the benefit from the cloud is different from the benefit that a digital twin could offer. The cloud is used for updating the model and it has more resources for training the model and improving it without interfering with the rest of the system. The digital twin is not concerned with utilizing resources to train the model, but to serve as a digital representation of the system, and hence may not have the functionality necessary to train the model.

Furthermore, this reference design could be applied to other projects with different metrics as the process for collecting, simulating, and packaging the data to train and test the model with a current metric has been explained and implemented. This reference design implements the intrusion detection algorithm at the communication layer of the inverter, which is at the Raspberry Pi, and obtains the metrics from the lower layers through Modbus TCP packets.

If another protocol is used, training the algorithm or simulating the data would not change; only a portion of the data collection process would need to be updated. As a result, this reference is design is transferable for use with other communication protocols.