Spring 5-14-2022

APPLICATION OF MACHINE LEARNING FOR PREDICTING IEMI UPSET IN MULTI-ARCHITECTURE MICROCONTROLLERS

Daniel S. Guillette

Daniel S. Guillette
Candidate

Electrical and Computer Engineering
Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Christos Christodoulou, Chairperson

Edl Schamiloglu

Nathan Jackson

Sameer Hemmady

Timothy Clarke

F. Mark Lehr

APPLICATION OF MACHINE LEARNING FOR PREDICTING IEMI UPSET IN MULTI-ARCHITECTURE MICROCONTROLLERS

By

DANIEL GUILLETTE

B.S., Electrical Engineering, New Mexico Institute of Mining and Technology, Socorro, NM, 2012

M.S., Electrical Engineering, New Mexico Institute of Mining and Technology, Socorro, NM, 2014

DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico

Albuquerque, New Mexico

May 2022

DEDICATION

To my loved ones.

"We keep moving forward, opening new doors,

and doing new things because we're curious

and curiosity keeps leading us down new paths."

– Walt Disney

ACKNOWLEDGEMENTS

I would like to sincerely thank my academic advisor and committee chair, Dean Christos Christodoulou, for his support, advice, and encouragement in completing this dissertation. This pursuit would not be possible without his flexibility and openness to pursuing an externally motivated topic while a part-time student. For that, and his wisdom to ward me away from research pitfalls and likely other things I cannot fathom, I am deeply grateful.

Next, I would like to thank and acknowledge the support of my committee members: Dr. Timothy Clarke, Prof. Sameer Hemmady, Prof. Nathan Jackson, Dr. Mark Lehr, and Prof. Edl Schamiloglu, who were instrumental in bridging the AFRL to UNM chasm. Their vital feedback and encouragement enabled me to pursue this topic confidently and complete meaningful research that serves academic and national security interests. Furthermore, Dr. Timothy Clarke deserves additional special thanks for his efforts to secure funding for this work over the last five years.

I am incredibly grateful to Dr. Julie Lawrence, who encouraged and initiated my entering into UNM's ECE Ph.D. program. I can honestly say that I would not have considered entering the program without her encouragement and direction. Furthermore, I am indebted to Mr. Matthew Landavazo and Dr. Rusmir Bilalic, who accompanied me in conquering the qualification exam, coursework, and general malaise. Their friendship and support cannot be overstated.

This work was locally peer-reviewed by Dr. Michael Lambrecht, Dr. Nicholas Erickson, Mr. Jeremy McConaha, Mr. Alexander Thayer, Dr. Joseph Connelly, and Dr.

Peter Mardahl. To them, I extend a massive thank you. Not only did they read multiple versions of this dissertation, but they also acted as a sounding board to a million different thought experiments and provided meaningful suggestions and advice.

Additionally, I gratefully acknowledge my current and past Air Force Research Laboratory leadership: Mr. Jose Chacon, Dr. Andrew Greenwood, Ms. Sabrina Maestas, Capt. Seth Nickolas, and Ms. Mary Lou Robinson. Working full-time while also in school is a challenge, but they each did all they could to make it painless for me, and I am genuinely grateful for that.

Next, I want to extend an enormous thank you to my parents, Mr. Stephen Guillette and Mrs. Sherry Guillette, who gave my sister and me so many opportunities that they never had. I specifically want to thank them for writing and holding up the "road signs of life" that pointed me in the right direction during challenging times of my life. I know for a fact that without their love and guidance, I would not have had the confidence or means to go to college, pursue an engineering degree, or continue into graduate school…twice.

I am deeply grateful to my partner, Ms. Ashley Guild, who helps me weather the good, bad, and ugly days that occur in life. The significance of her love, support, and willingness to stand by me is hard to quantify, and I am confident I would not be as whole – or sane - as I am now at the end of this venture without all she has done.

Lastly, I am fortunate to have many family, friends, and cohorts; I can confidently say their kindness has made a positive difference in my world.

APPLICATION OF MACHINE LEARNING FOR PREDICTING IEMI UPSET IN MULTI-ARCHITECTURE MICROCONTROLLERS

By

DANIEL GUILLETTE

BS Electrical Engineering, New Mexico Institute of Mining and Technology, Socorro, NM, 2012

MS Electrical Engineering, New Mexico Institute of Mining and Technology, Socorro, NM, 2014

Ph.D. Engineering, University of New Mexico Albuquerque, NM, 2022

ABSTRACT

Four microcontrollers were programmed to execute a simple counting program. Pulsed RF signals – also known as Intentional ElectroMagnetic Interference (IEMI) – were injected into the clock input of the microcontrollers. At the same time, the output lines were monitored to determine whether the IEMI signal altered the output of the counting program – referred to as an upset. A state-of-the-art automated testing apparatus was used to collect and process 120,960 samples of IEMI upset data. The data was used to perform a traditional upset trends study and train a series of machine learning (ML) techniques – k-Nearest Neighbors, Support Vector Machines, and Decision Trees – to predict IEMI upset using information about the IEMI waveform and injection time. It was determined through comparisons of the traditional and classifier-based trends that same-architecture devices shared remarkably similar trends, and the different architecture device had trends that were similar, but were offset to suggest higher resistance to IEMI upset. The Weighted k-Nearest

Neighbors (k-NN) technique was identified as the best overall method, having the highest prediction accuracy and second-lowest training time compared to multiple variations of Support Vector Machines, Decision Trees, and k-NN algorithms. Ten features from the IEMI waveform characteristics, such as frequency, power, and pulse width, were used to train a Weighted k-Nearest Neighbors Machine Learning classifier. MATLAB provided the means to train, validate, and export 1023 different classifiers using the ten features in all possible combinations, such that the relative importance of each feature and the best feature combinations could be determined.

Key results include: 1) Classifiers trained with data from a single microcontroller can make reasonably accurate ($P > 85\%$) predictions when validated against the other devices' datasets, even when the microcontrollers have different architectures. 2) The optimal training set used data from all four devices to result in an average accuracy of $P = 91.58\%$. 3) Using data from only MCU1 and MCU2 – which are different instances of the same device - resulted in a median accuracy of 91.27% across all four devices. 4) There was a ~2% decrease in prediction accuracy when only 10% of the entire dataset (randomly chosen) was used as training input and a ~10% decrease when using only 1% (randomly chosen) of the data. These results suggest meaningful upset predictions across multiple architectures can be made using k-NN classifiers, even with sparse datasets.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

The world is rapidly integrating with digital electronics to the point where nearly every facet of modern civilization relies on computers. [1] Utilities, Entertainment, Transportation, Food Production, Commerce, and more all rely in some way on digital electronics to complete their task. [2] As established in the Caselli paper, computers increase efficiency for a wide variety of tasks and jobs, which results in a more prosperous civilization. As technology evolves, so too does its connectivity. The advent of the internet has pushed electronic devices away from standalone usage in favor of an always-connected internet-of-things (IoT) based ecosystem. [2] To be part of the IoT, electronic devices must have the ability to transmit and receive radio frequency (RF) signals. These RF signals vary in frequency, strength, duration, and communication protocol.

Electronics manufacturers, therefore, must contend with a crowded RF signal environment to ensure that their device sends and receives the signals it is designed to while not being interrupted by other RF signals nearby. Experts of the Electro-Magnetic Interference and Compatibility (EMI/EMC) disciplines and government bodies, such as the US Federal Communication Commission (FCC), work hard to establish standards that define how devices should operate. These standards ensure that commercial products do not unintentionally interfere with radiofrequency spectrums that are not approved for the device class or are reserved for other functions such as emergency or military communications. [3]

Additionally, the United States, Russia, China, England, Germany, and other countries continue to work on electro-magnetic-based weapons. Their research likely follows on from discovering an Electro-Magnetic Pulse (EMP) results from the detonation of a nuclear weapon. [4] When an EMP occurs, it disrupts nearly all powered-on electronic devices within its area of influence. High Power Electro-Magnetic (HPEM) weapons programs were founded and began focusing on building technologies that could produce EMP-like signals without the kinetically destructive forces associated with nuclear weapons. While commercial electronic devices are designed to meet EMI/EMC and FCC standards, they often are not hardened against EMP/HPEM generated signals. Only devices which operate in extreme radio frequency space or military environments are designed to meet EMP-type standards. Unfortunately, this means that much of the world's infrastructure uses electronics vulnerable to EMP/HPEM. [5] The exact scope of infrastructure damage resulting from an EMP/HPEM attack is unclear. However, recent cyber-attack events on infrastructure, such as the Ukraine power grid hack in 2015, the 2021 Colonial Pipeline Co. gas line ransom, and the 2021 Texas Power Crisis, prove that utility systems have little redundancy or margin for error when disrupted. [6] [7] [8] These concerns are further discussed in the 2010 report by Radasky and Savage which details the methods and means by which malicious electro-magnetic signal attacks could impact the US power grid. [9]

When electronic devices encounter EMP/HPEM signals, they can produce a wide range of adverse responses; from low-impact outcomes like computer screen flickers to physical damage outcomes on circuit board components. [10] [11] These responses are

formalized by Schamiloglu et al. to define an overarching, generic set of categories and consequences for electromagnetically induced effects – see Table 1 . [5]

Table 1: Schamiloglu's categories and consequences of electronic effects

Failure Mode	Power Required	Wave Shape	Recovery Process	Recovery Time
Interference/ disturbance	Low	Repetitive Pulse or Continuous	Self-recovery	Second
Digital Upset	Medium	Short pulse, single or Repetitive	Operator intervention	Minutes
Damage	High	UWB or narrow band	Maintenance	Days

Unfortunately, thousands of new electronic devices are manufactured each year, making it difficult for interested parties to safeguard every device against an increasingly crowded EM signals space. Until 2001 EMP-like signal attacks were referred to as EM terrorism to differentiate between accidental or passive EM induced disruption, and intentionally caused EM disruption. [12] [13] The term EM terrorism was replaced by the EMI/EMC community with Intentional Electro-magnetic Interference (IEMI). [14] Although much of the EMP work influenced IEMI work, Radasky et al. clarify that IEMI is not to be confused with High-altitude Electro-Magnetic Pulse (HEMP) effects. The primary reason being that HEPM signals often contain additional phenomina such as gamma and x-rays in addition to ultra-wideband electro-magnetic fields. Radasky gives

the following definition for IEMI, *"In general, we are speaking of the intense electro-magnetic fields."* [9] Further clarification suggests that IEMI signals need to be designated by their bandwidth and amplitude, as not every electro-magnetic field is the same. Given the ever-growing number of devices, it is essential to begin modeling the conditions in which electronic devices may be disrupted. If IEMI predictive models are not developed, researchers and manufacturers cannot suggest or implement IEMI countermeasures, resulting in damage to global economies or loss of life.

1. Motivation

In the discipline of Electro-Magnetics (EM), one of the most challenging problems is predicting the upset of digital electronics due to the sheer complexity and individual variation of each device and any contained components [15]. Furthermore, RF circuit design best practices suggest that the length, width, and location of circuit elements, such as circuit board traces, resistors, microcontrollers, and more, can significantly impact RF coupling and compatibility [16]. Consequently, trying to simulate or model upset on any given electronic device accurately would require perfect knowledge of the circuit board layout, the components, as well as their relative placement to the impinging EM wave. According to some experts, the problem's scope suggests that properly building a predictive model would be an untenable problem akin to modeling reality.

Unfortunately, manufactures are often unwilling to reveal the specifics necessary to build a predictive model of the device because it likely would reveal intellectual property. Consequently, best efforts to address EM upset modeling require expensive and time-consuming empirical effects testing, which, if not done carefully, can provide

skewed and counterproductive results. Given that it is impractical and unrealistic to test and collect every permutation of a real-world EM encounter (and on every single device), it is desirable to have a predictive model. Since imperfect information is expected, the goal is to make the best model possible using the available information and tools. One such tool which is well suited to the task is machine learning. Consequently, this research intends to determine the extent to which select machine learning methods can predict upset in microcontrollers during intentional electro-magnetic interference.

Microcontrollers, sometimes referred to as microcomputers or microcontroller units (MCU), are simple computers that can be programmed to execute a set of instructions on a precise time scale, reliably if the device is powered. [17] [18] As such, MCUs are well known to be the backbone of complex IT systems. In this role, MCUs handle simple hardware-based tasks such as switching on and off power supplies, interpreting keyboard stokes, and more. For these reasons, microcontrollers are an ideal test article for research and establishing a solid IEMI predictive understanding.

Although microcontrollers are relatively simple devices, they are still a challenge to model because IEMI events most commonly occur via free-field coupling. Free-field coupling generically is described as one or more RF signals radiating from a source that impinges upon an electronic device (victim). The victim target is designated as such because it is within the beam path of the radiating source. [5] Famous fictional examples of IEMI can be found in the Hollywood films Ocean's Eleven and Ocean's Thirteen. In these movies, EM signals, produced by devices referred to as a 'pinch' and a "magnetron," are used to disrupt infrastructure (knock out the power to the city) and IT systems (shut down a computer server-farm) on the Las Vegas Strip, so that thieves can

achieve a heist in each film. [19] [20] The EM attacks observed in these movies are representative of real-world IEMI upset. However, in standard Hollywood fashion, technical aspects are distorted for the sake of the plot. For example, in Ocean's Thirteen, the magnetron is housed inside a fully functional cell phone, when in reality, it should be roughly the size of a large filing cabinet.

Consider the cartoon in Figure 1.

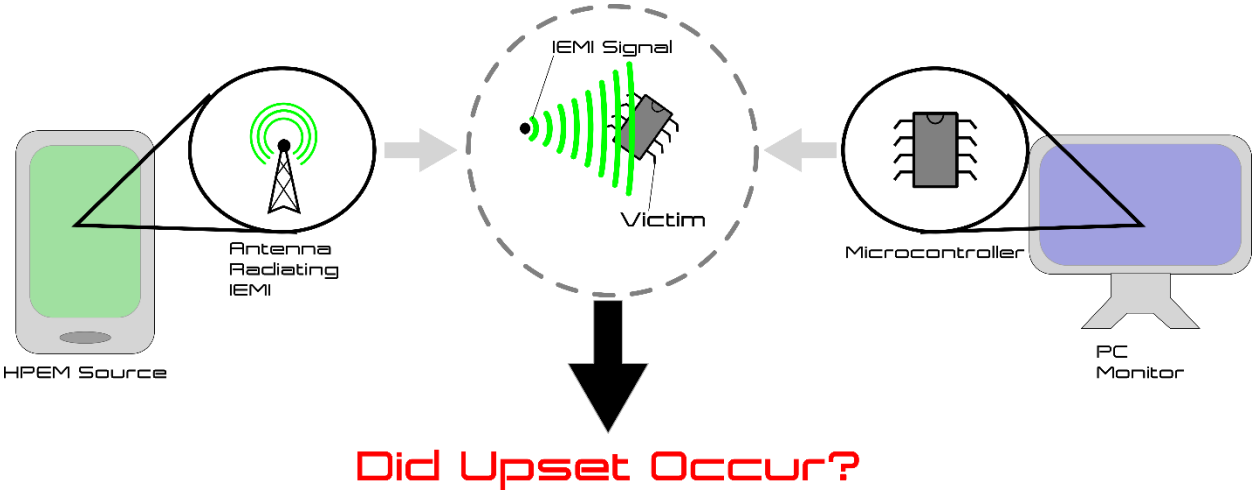

Figure 1: IEMI Encounter Example

In this cartoon scenario, an HPEM source encased in a cell phone passes near a PC monitor. A microchip in the monitor, which turns the monitor's screen on and off, experiences the phone's radiating IEMI signal, and the monitor's screen flickers for a second. Was it the phone that caused that flicker? A power grid fluctuation? Or perhaps a software glitch that caused the display to flicker? The problem with this scenario is that more detail is needed to determine whether an upset occurred and whether the IEMI signal was responsible.

Most electronic devices are part of complicated systems-of-systems which rely on one another to execute an overarching set of tasks. As a result, a cascading set of device

failures may occur if one device is affected. Therefore, it is important to carefully determine if the lynch-pin device is being affected by the IEMI signal.

From a modeling standpoint, details such as the actual relative locations of the radiating source and victim device, the angle of incidence, frequency, signal power, antenna gain, and pulse width for the IEMI signal will factor in. Next, the material composition of the device's external chassis and environmental factors such as buildings and terrain within the beam path may distort the IEMI signal before it reaches a circuit element. These effects describe RF scattering, which details how an EM signal changes through materials.

Importantly, even with all that information, it would still be unclear if the IEMI coupled to one microchip or multiple circuit components within the monitor. And if it did couple, was it the combination of the signal on those pins/components that caused the upset, or was it caused by a single pin/component?

Ultimately, there are too many "what-ifs" to answer with so little understanding of the actual phenomena taking place. Therefore, the problem was rescoped to ensure that the IEMI waveform characteristics and coupling path were consistent and well tied to a measurable upset outcome.

2. Research Scope

Two simplifications are made to this research to reduce the problem's scope and ease into the broader discipline. 1) a simple 8-bit microcontroller is used as the Device Under Test (DUT) instead of a complicated device, and 2) Direct Power Injection (DPI) is used to send an IEMI signal to the MCU instead of free field exposure via antenna.

Direct Power Injection is a technique that uses instrumentation such as a Radio Frequency (RF) amplifier and an RF switch connected with RF cabling to directly inject an IEMI event via physical connection into a DUT. [21] Direct Power Injection is used in this research because it ensures that only the connected location experiences the IEMI event. Figure 2 shows a diagram example of a direct injection experiment on an integrated circuit created by Baric and Ceperic. [22] This diagram shows that a computer controls an RF signal generator (RF source), RF amplifier, and oscilloscope to send an IEMI signal into an integrated circuit and capture the result.

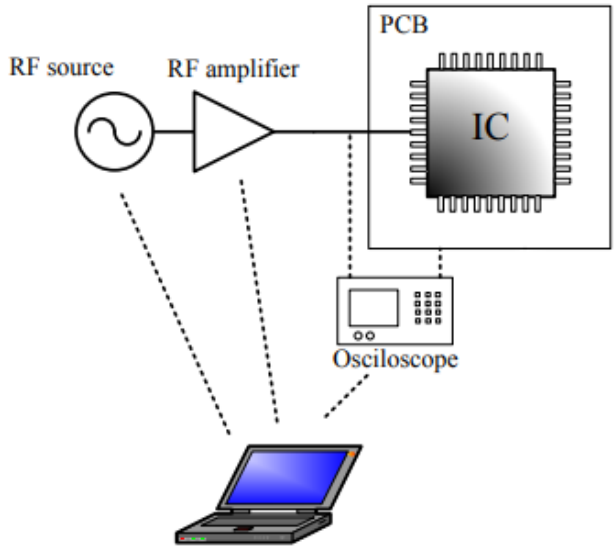

Figure 2: Example of DPI Experiment Setup from Baric and Ceperic

Often, in a DPI experiment, IEMI will be injected on a single input pin of a DUT, meaning the IEMI content can only couple to that pin. Alternatively, in free-field experiments, the IEMI signal could couple to multiple pins, or none, as the RF is projected by an antenna based on its radiation pattern. Therefore, it is crucial to control

and understand where the IEMI couples to so that any adverse response (upset) the device experiences can be appropriately attributed to the IEMI event.

3. Prior Work

It is hard to pinpoint an exact starting point for electro-magnetic disruption research due to the secrecy associated with the Manhattan Project and HEMP testing, but the seminal work done by Richardson et al. in the 1970s is an excellent place to start. [23] Their work focused primarily on how external RF signals, which couple to a circuit out-of-band, are rectified into a DC voltage signal that can adversely affect the operation of a Bipolar Junction Transistor (BJT) and circuitry connected to it. This work also acts as an anchor for two other key concepts, electro-magnetic interference (also known as upset) & out-of-band circuit coupling. Moving forward, other work, such as that done by Wunsch and Bell, established thresholds for junction failure due to thermal change as caused by RF heating. [11] However, Vick and Habiger in 1997 state that upset on microcontrollers vary due to instruction execution, clock state, and IEMI waveform characteristics. [24] This particular work inspired two particular studies performed before this dissertation by Guillette et al. (in 2015, but not published until 2018 & 2019) to evaluate the change in probability of upset for different powers, instructions, pulse widths, and separate instances of the same model chip. [25] [26] This work will be discussed in more detail in Chapter 2.

In 2004 a special issue of IEEE Transactions on Electromagnetic Compatibility was published by Radasky et al. and featured 15 papers which focused on defining standards, trends, and fundamental concepts associated with IEMI and High Power

Electromagnetics (HPEM) effects. [27] Many of the papers are not relevant to this specific dissertation but, as a whole, are a significant milestone in the creation/development of the IEMI effects discipline. For this reason, it is necessary to acknowledge these papers but not to delve into them.

The most recent, noteworthy, and relevant report is the dissertation by Bilalic in 2017, which established that machine learning was well suited to predicting upset on microcontrollers. [28] His work focused on three machine learning methods: Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Gaussian Processes for Machine Learning (GPML), and ultimately determined that all three were successful but that there were tradeoffs between the methods related to data size, complexity, and time to train – See Chapter 2 for additional detail.

This dissertation aims to expand upon Bilalic's work by evaluating fast-to-train machine learning methods, such as k-Nearest Neighbors and Decision Trees. Bilalic cites multiple papers which have looked at the suitability of SVM, GPML, and ANN for electro-magnetics type problems. However, to date, no papers look at the application of fast-to-train methods, such as k-Nearest Neighbors, or Decision Tress, to predict IEMI upset.

For a method to be fast-to-train, it typically needs a basic mathematical backbone – sometimes referred to as a kernel - for quickly classifying the data, even with large datasets. [29] Certain mathematical operations are complex and expensive and can thus be prohibitive for machine learning applications.

A primary challenge of Machine Learning, especially in commonly used methods like Support Vector Machines, is interpreting the result. Multi-dimensional contour planes – which separate data into two or more classifier labels – are often too complicated for humans to interpret. This results in the problem where many machine learning classifiers are treated as black boxes which must be implicitly trusted. Although the classifier may be useful in predicting phenomena occurring in the data, the end-user often does not gain any insight into the interconnectiveity of the data or know when the classifier may be wrong.

4. Research Questions

This dissertation aims to broaden the scope of understanding into how the k-Nearest Neighbors and Decision Tree algorithms perform when applied to empirically collected IEMI effects data on microcontrollers. Emphasis is given to cross-training classifiers on three different MCU devices – with separate and combined sets of the collected effects data - to determine the extent to which an overarching predictive upset model can be built for MCU devices of a) the same serial number, b) the same architecture with different serial number, and c) a different architecture. Lastly, the datasets will be reduced in size by selection of random subsets (sparsed) to determine the impact of each waveform feature (frequency, pulse width, power, etc.) and the overall change in prediction capability when less data is present – for example, when only 10% of the data is available. These topics result in four key research questions to be addressed by this dissertation:

1. Are there easily-interpreted upset trends that can serve as useful ‘rules of thumb’ when considering untested devices?
2. To what extent can fast-to-train machine learning methods predict upset?
3. What features are needed to make high-quality predictions?
4. To what extent can a chip, architecture, and device-wide classifier predict upset in microcontrollers?

The information gleaned in this body of work will contribute to the overall understanding of IEMI upset on electronic devices because commercial devices rely on this class of circuit components. However, this research's primary and original aspect is the trade study assessment of machine learning algorithms, feature data sparsing, and prediction-making results from empirical effects data on four multi-architecture microcontroller devices. Furthermore, to date, there does not appear to be a prior application of the k-Nearest Neighbors or Decision Tree algorithms to electro-magnetic induced upset problems; therefore, a novel result will be produced there as well. Upon completion of this work, the ultimate goal, and related future work, is to expand the methodology and apply it to more complex electronic systems and free-field effects data.

5. Dissertation Structure

This remained of this dissertation is structured in the following way:

Chapter 2 will provide a literature review of relevant work to support this dissertation.

Chapter 3 will detail the experiment setup, devices under test (DUT), specific examples of upset for the microcontroller devices under test, data processing methods, and the testing parameters. This section will also detail the limitations and design choices made to enable automated data collection possible.

Chapter 4 will detail the machine learning algorithms used, a brief overview of their mathematical basis, and the relevant decisions associated with the algorithms implemented, such as training, validation, and feature selection methods.

Chapter 5 will present an overview of the IEMI upset data collected. Additionally, traditional and existing upset trends from the collected data will be presented and discussed.

Chapter 6 will present results to a survey of, and down-selection from, three machine learning algorithms (k-Nearest Neighbors, Decision Trees, and Support Vector Machines), with the best-of-class being moved forward for use in the rest of the dissertation.

Chapter 7 will present results to a machine-learning-focused comparison of classifiers trained using different combinations of IEMI waveform data. The best-performing classifier will be moved forward for use in the rest of the dissertation.

Chapter 8 will present results to compare and assess classifier predictions made using different combinations of devices' data.

Chapter 9 will present an interpretation of the results presented in Chapters 5-8.

Chapter 10 will present future work, recommendations, and conclusions.

Appendix and References will contain relevant content and citations.

Chapter 2: Literature Review

This chapter provides a literature survey on state-of-the-art and historically impactful research relevant to this dissertation. To date, there does not appear to be a large body of work at the intersection of IEMI effects and machine learning. Several papers have applied machine learning to electro-magnetics problems within the last ten years, but many of them do not specifically relate to this work. Regardless, it is valuable to review notable works in these topic areas. Two sections are found below, literature related to 1) Intentional Electromagnetic Interference (IEMI), and 2) Machine learning algorithms applied to electro-magnetics problems. The latter parts of both sections review papers that are of direct relation and importance to this dissertation

1. Intentional Electro-magnetic Interference

This work is spiritually the evolutionary outcome of Electromagnetic Pulse research after World War II. Given the secrecy associated with the Manhattan Project, it is hard to pinpoint the earliest work performed in the field of electro-magnetic disruption of electronic devices. However, work performed by Richardson, Puglielli, and Amadori on Bipolar Junction Transistors (BJT) was published in 1975 and 1979. [23] [30] There is a reference to an earlier paper by Richardson, Puglielli, and Amadori from 1973 titled, "Prediction methods for the susceptibility of solid-state devices to interference and degradation from microwave energy" however that paper is not readily available. Regardless of the exact start date, their work focused primarily on how external RF

signals, which couple to a circuit out-of-band, are rectified into a DC voltage signal which can adversely affect the operation of the BJT and circuitry connected to it.

Moreover, two other key concepts, electro-magnetic interference (also known as upset) & out-of-band circuit coupling, are found in this paper. In Richardson's opening paragraph, he comments that heart pace-maker interference was an "*often quoted example*" of electronic interference and that the "*interference signal is received on stray wiring or other unintended antenna…*" which is the phenomenon described in the modern-day as back-door coupling. Consequently, research on the adverse effects of electro-magnetic signals on electronic devices is not a new field of research. Over the years, the field has evolved with two topics weaving in and out of each other: 1) Electromagnetic Compatibility and Interference and 2) High Power Electromagnetic Weaponization. And in 2004, the term IEMI was minted to encompass the intersection of these two topics and the idea of EM terrorism. [31]

Complementary to back-door coupling is the idea of front-door coupling. In short, front-door coupling occurs on circuit elements that are in-band and meant to transmit or receive RF signals, whereas in back-door coupling, out-of-band RF signals are induced in circuit elements that are not meant to transmit or receive RF signals. [32] [33] The differentiation between these two coupling methods is a fundamental concept associated with IEMI effects, as damage often occurs on front-door type attacks, while only interference typically occurs in back-door type of effects. Moreover, back-door type effects can also manifest from unintentional electro-magnetic interference. For example, advanced devices such as cellphones and drones will employ shielding measures to ensure that the signals being broadcast by subsystems in the device do not cross-talk into,

or couple to, measurement devices or sensitive circuit elements. The concept of front door coupling has limited relevance to this dissertation because the signals are being direct-injected on a digital signal line. Digital signal lines are often isolated from analog radio wave signals via conditioning or discretizing circuitry thus making direct injection on this type of line, by definition, back-door coupling. However, it is important to understand the concepts and where the line is drawn between them. This dissertation investigates back-door type effects but does not pursue the coupling aspect of the problem because direct power injection is used to inject the signal into an explicitly chosen circuit node – the clock input line of the microcontroller.

Other work done by Wunsch and Bell established thresholds for semiconductor failure due to HEMP-induced RF heating which thermally damaged the circuit components. [11] The focus of this dissertation is not on damage. However, it is worth acknowledging for two reasons: 1) it states that semiconductor devices, such as diodes and transistors, are the most susceptible circuit elements to EM induced upset, and 2) because it suggests that EM effects can occur over a broad spectrum of frequencies and also suggests that the IEMI frequency does not need to perfectly match the circuit element (in some way) to cause upset.

One key difference between HPEM and HEMP signals is that HPEM tends to be narrow-band, meaning that a single frequency - or narrow slice of frequencies - contains the signal's energy. Alternatively, HEMP signals contain an ultra-wide-band spectrum of frequencies – often referred to as occurring from “DC to Daylight” to signify that only optical frequencies are not present in some manner. This suggests that HEMP signals will be able to couple to a circuit more easily by both front and back-door means, while

narrow-band signals may have lower efficiency in coupling based on how well matched they are to whatever circuit they encounter. In this manner, back-door coupling signals with a frequency well matched to the circuit will result in more energy coupling to the circuit element, while signals with a poorly matched frequency will result in less. For this reason, damage-type effects are not generally seen in back-door coupling but are more common in front-door coupling.

In 2004 a special issue of IEEE Transactions on Electromagnetic Compatibility was published by Radasky et al. and featured an introduction to fifteen separate papers which focused on defining standards, trends, and fundamental concepts associated with IEMI and High Power Electromagnetics (HPEM) effects. [27] The papers are grouped into four main topic areas: 1) IEMI waveforms and HPEM test capabilities to generate waveforms, 2) Coupling as applied to cables and systems, 3) Effects on equipment, systems, and communications, and 4) Protection, measurements, and standards. Only two of the fifteen papers are relevant to this dissertation because the rest focus on topics beyond the scope of this work. For example, source development, coupling models, and effects on complex devices can all be their own dissertations. However, this collection of papers is notable because it provides an essential overview of terminology, methods, and past work, which led to the development of the IEMI effects discipline. For individuals new to the discipline, it is a solid starting place. The first paper of relevance in this collection is by Giri and Tesche. It covers the definition of Intentional Electro-Magnetic Environments (IEME) and explicitly defines the frequency spectrums associated with wide-band HEMP and narrow-band IEMI. The second paper is by Camp et al. and focuses on the statistical analysis and modeling of microcontroller upset to EMP signals. [34] [35]

The Giri and Tesche paper provides three ways to categorize IEMI signals: 1) by the waveform characteristics, 2) by the sophistication of the source used to produce the signal, and 3) by the effects it produces. Although each method has pros and cons, Giri and Tesche conclude that the first method is preferred. Consequently, the IEMI community generally parameterizes upsets in terms of waveform frequency, bandwidth, amplitude, pulsewidth, and repetition rate. Giri and Tesche focus on two characteristics in particular: frequency spectrums by the amount of bandwidth they cover (see Table 2) and the relative magnitude of E-field strength, as seen in Figure 3. In addition, Table 2 details the percentage bandwidth for narrow to hyper-band signals based on a bandwidth ratio, while Figure 3 shows a relative set of examples for what kind of spectral density is expected from different kinds of signals. The specific details of the IEMI signals used in this dissertation are discussed in Chapter 3, but in general, it focuses on single frequencies from 20 to 800 MHz and with a peak power level below 20 Watts.

Camp et al. focus on free-field radiated UWB/EMP spectrum signal upset on three different microcontroller types manufactured using the complementary metal-oxide-semiconductor (CMOS) technology. [35] As suggested in the Wunsch and Bell paper [11], EMP effects are valuable from a generic standpoint but differ from narrow-band IEMI signals, so the results are not entirely applicable. Regardless, Camp studied the probability of effect as a function of the I/O port state, different signal line lengths, MCU clock rate, and EM pulse shape. Figure 4 presents a summary of Camp's results. Camp found that the reset port was the most susceptible to upset across all three devices, followed by the clock, power, and other I/O ports.

Work done by Vick and Habiger focused on the upset of multi-architecture microcontrollers using narrow-band frequencies and pulse widths which are on the same time scale as the device's operating clock. [24] The paper details automated experimentation to subject a microcontroller to an IEMI signal at different times during its operation. These times related to different assembly instructions that were executed by the microcontroller. A primary goal of the paper was to determine the probability of effect associated with different software instructions during execution. It was presumed by the authors that there is a hardware change within the microcontroller when different software instructions are executed, which could result in different hardware configurations, and consequently, a different probability of effect. This is important because it suggests that the upset susceptibility depends on the software being executed – see Figure 5.

Table 2: Giri and Tesche's IEME Bandwidth Classification Method

IEME CLASSIFICATION BASED ON BANDWIDTH

Band type	**Percent bandwidth** $pbw = 200\left(\frac{br-1}{br+1}\right)$ (%)	**Bandratio *br***
Narrow or hypoband	$< 1\%$	< 1.01
Moderate or mesoband	$1\% < pbw \leq 100\%$	$1.01 < br \leq 3$
Ultra-moderate or sub-hyperband	$100\% < pbw < 163.4\ \%$	$3 < br \leq 10$
Hyperband	$163.4\% < pbw < 200\%$	$br \geq 10$

In combination with the Camp paper, this paper makes a good case for the plausibility of making architecture-wide predictions on IEMI induced upset. Therefore, this paper is vital to this dissertation.

Consequently, both works influenced Guillette and Clarke to perform two studies before this dissertation: 1) to determine the repeatability of the probability of upset trends for multiple instances of the same microcontroller, and 2) to determine the impact of clock state and software instruction on the probability of upset for narrow-band IEMI signals. [25] [26]

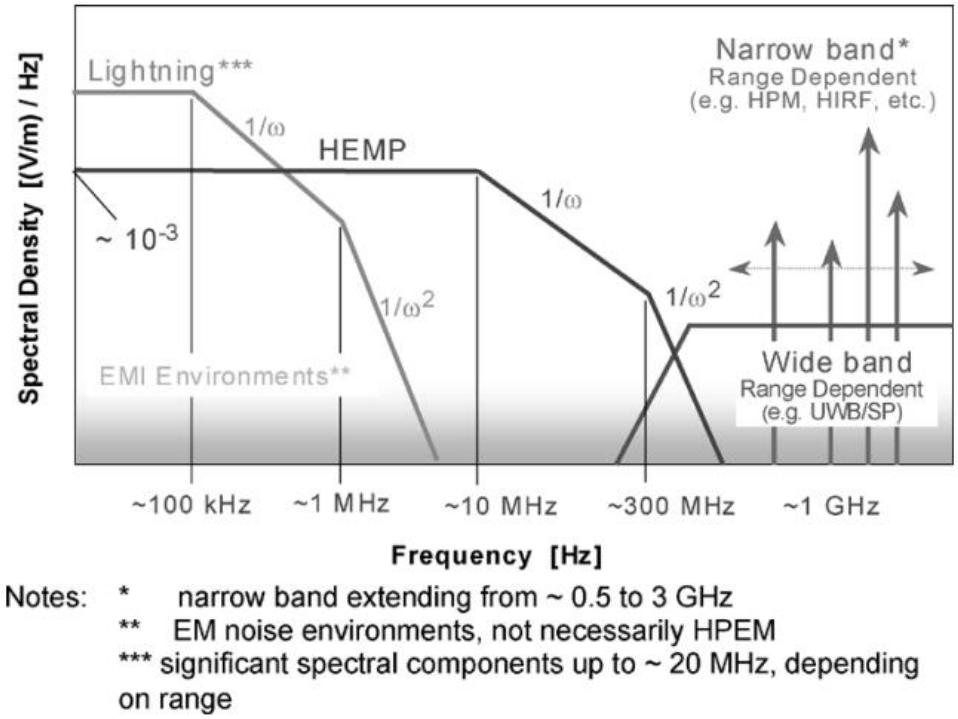

Figure 3: Giri and Tesche's Comparison of the spectra of several types of EM environments

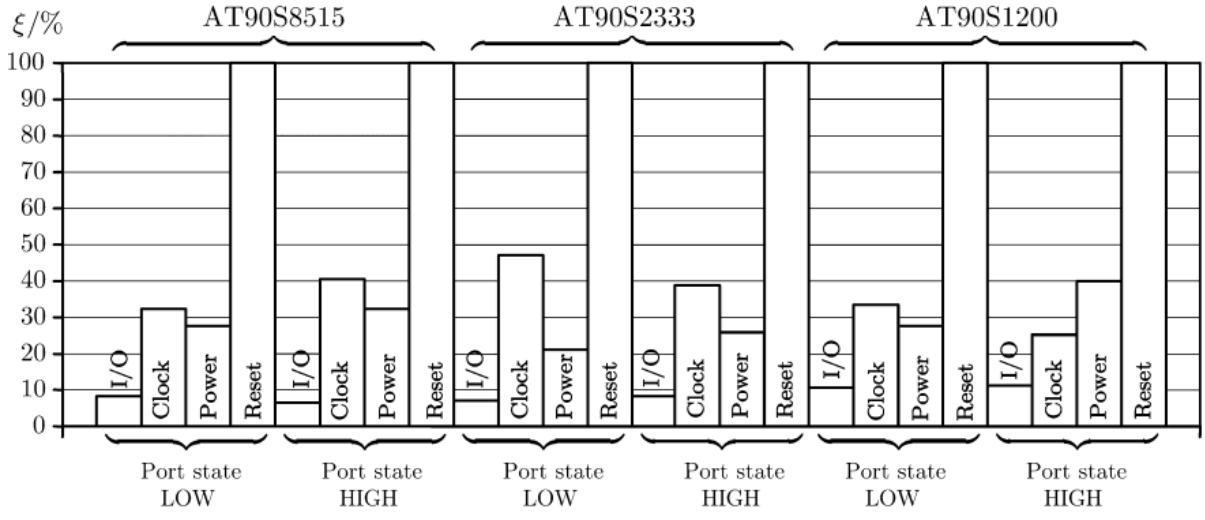

Figure 4: Camp's Susceptibility-percentage factor for three different microcontroller systems

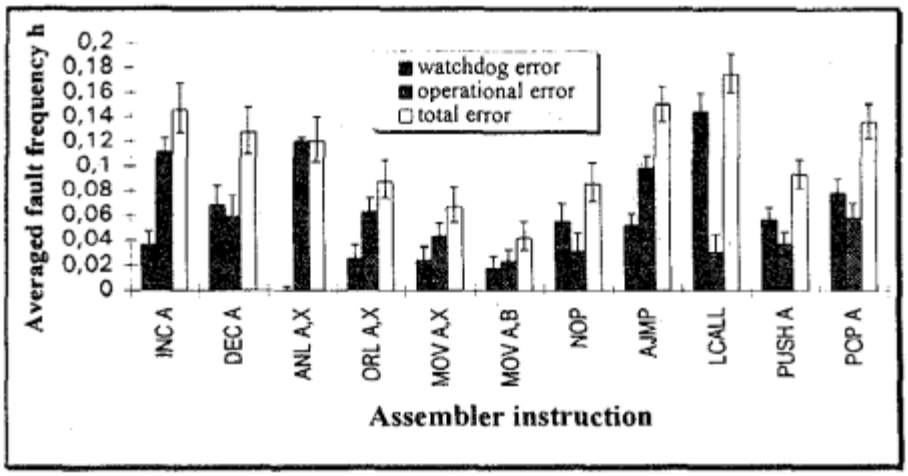

Figure 5: Example of Vick's Average Fault Frequency of Different Assembler Instructions

The first study found that nine of the ten microchips had a nearly identical probability of effect (PoE) response - see Figure 6. The chip that did not conform, M02, was observed to have a 50% PoE curve point nearly double the value the others had – this chip was presumed to have a manfacturing defect that may account for the difference in susceptibility. The naming convention for the MCUs continued with M02 after extensive preparatory experimentation was performed on device M01. M01 was not tested in the same trials because it could have been damaged during the exploratory, initial tests.

The second study assessed whether upset was more likely during certain clock states – i.e., during a clock HIGH vs. clock LOW. To robustly assess this question, IEMI was injected at different times with respect to the clock state and during different instructions at various power levels. An example of injection locations relative to the clock is seen in Figure 7.

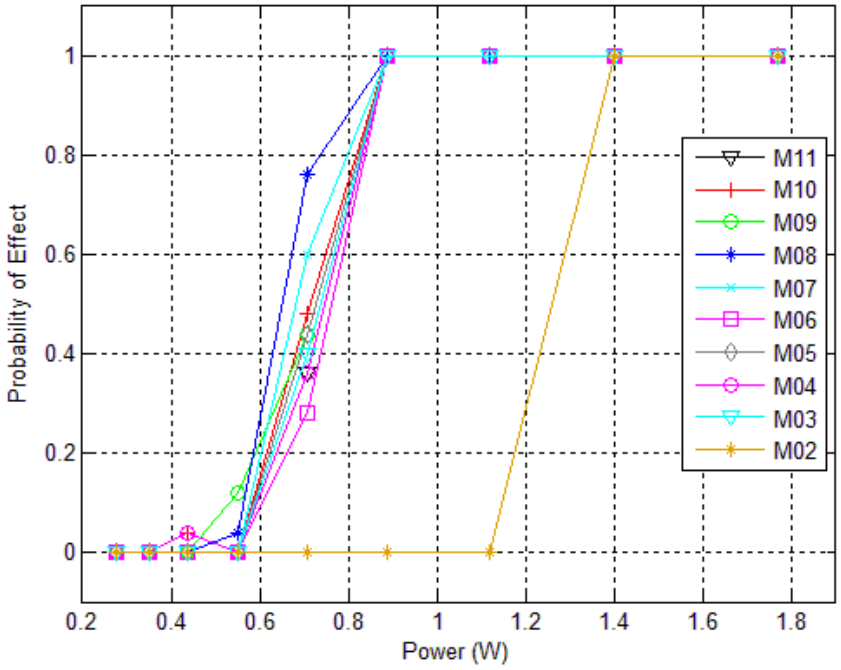

Figure 6: Raw Probability of Effect MCU chip variation @ 50ns pulse width

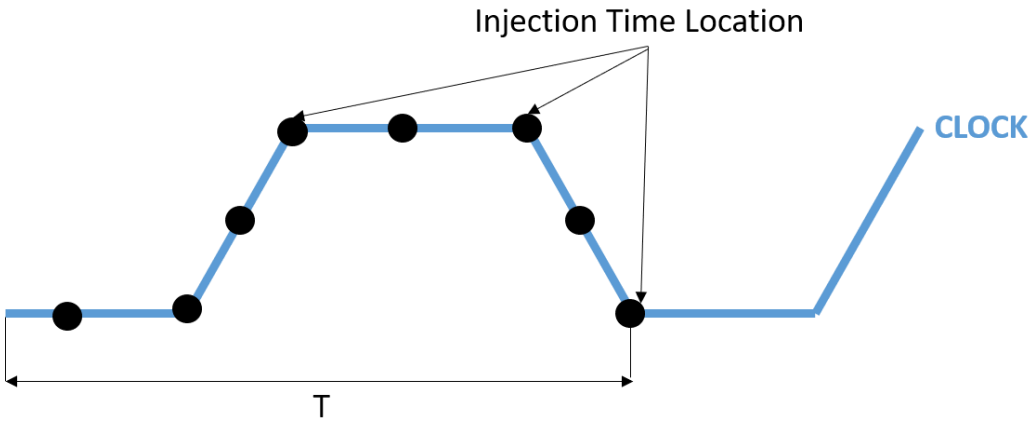

Figure 7: Diagram of RF Injection Time Relative to Clock Period.

Additionally, four different injection time windows (12, 14, 16, 18 µs) were considered. Each time corresponds to the first cycle of four instructions executed during the counting program. At the start of each time, a 100 MHz, 50 ns pulse of RF was injected, then stepped forward in time at a 50 ns resolution across the full first clock

period corresponding to each of the four software instructions. Seven different power levels were used at each point as well: -10 dB, -12 dB, -14 dB, -15 dB, -20 dB, -25 dB, and -28 dB, relative to a 20 W peak solid-state RF amplifier. These power levels are selected from experimental exploration, which showcased that they produced a varied range of effects.

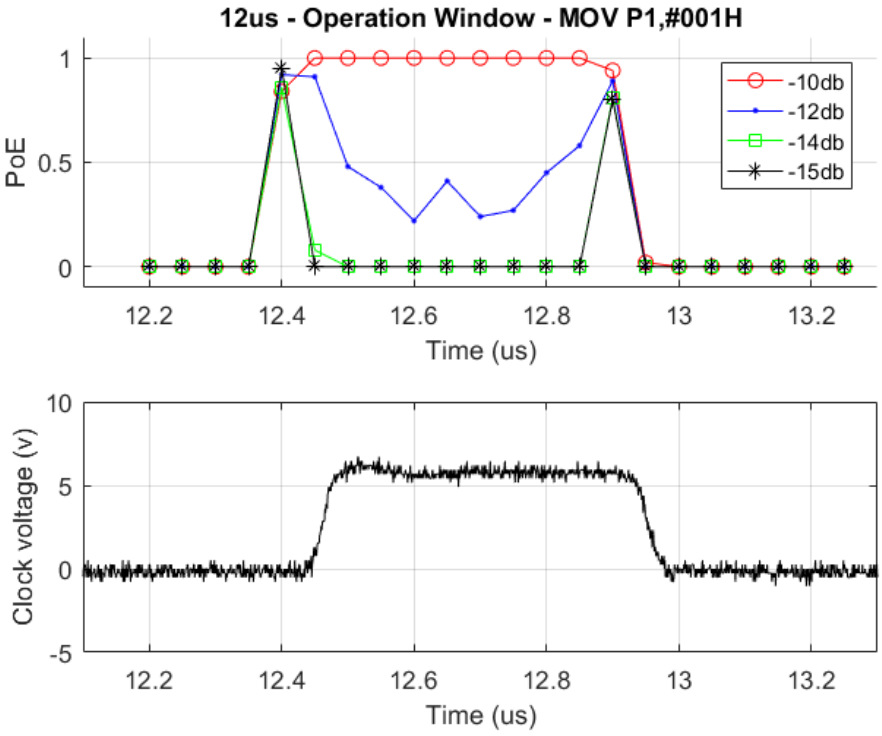

Figure 8: Probability of Effect for MOV instruction at 12 µs with power levels of -10 to -15 dB.

This study suggests that a high probability of effect is independent of the instruction occurring but is highly dependent on the clock state, as upset was rarely observed during the clock LOW states and highly observed during clock rise/fall and HIGH. An example of that response can be seen in Figure 8. The top subplot shows the Probability of Effect (PoE) for each injection time, while the bottom subplot shows the recorded clock state at those times. What should be taken away from this graph is that

injection around the rising and falling edges produced the most consistent upset. In addition, clock LOW states showed no upset regardless of injection amplitude, while clock HIGH states showed upset to be conditional based on the power level used.

IEMI event was present during different instructions - this result can be seen in Figure 9.

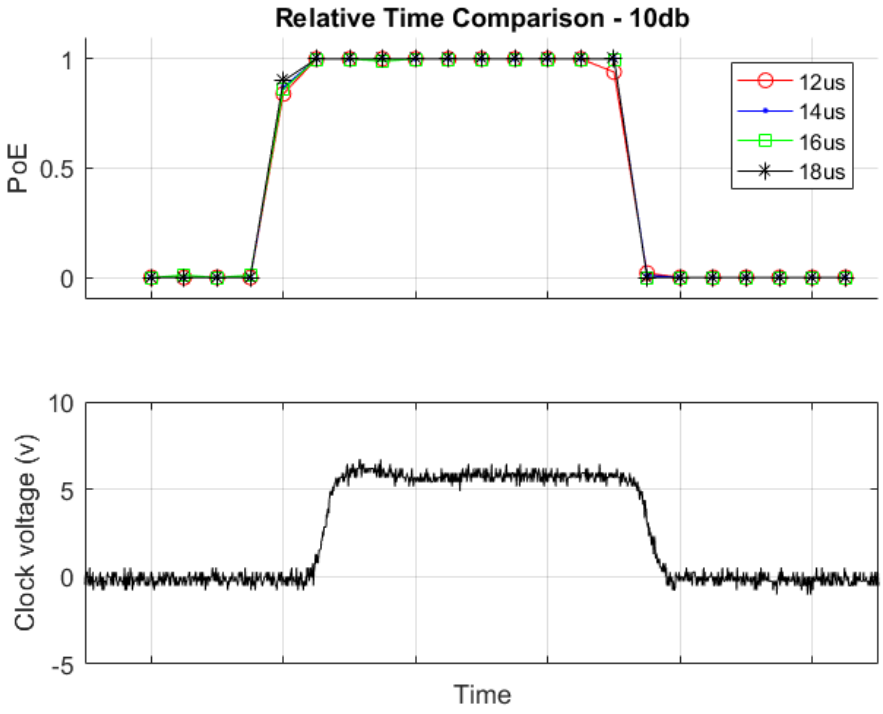

Figure 9: Probability of Effect of -10 dB power level for each instruction

Compared to Vick and Habiger's observations, the results from these two studies agree, in general, but differ in that Vick did not appear to increase the power high enough to observe the results shown in Figure 9. Alternatively, this dispute in upset occurrence for instructions could be a function of UWB vs. narrow-band IEMI signal injection.

2. Machine Learning in Electro-magnetics

Until recently, the machine learning and electro-magnetics disciplines were considered separate, and no research contained both. However, in the last 20 years, machine learning has been found to have practical and successful applications in many scientific fields, including electro-magnetics, as shown in the book “Neural Network Applications for Electromagnetics.” [36] There are many good examples of how machine learning can be used in electro-magnetics problems, but many focus on traditional problems such as propagation and coupling. Artificial Neural Networks were used to: predict EM field absorption inside a dielectric; [37] Predict induced AC voltages in underground metallic pipes due to high-voltage grid lines nearby; [38] And, model cross-talk between pairs of wires. [39] Support Vector Machines (SVM) were used to: Model the direction of arrival for impinging signals on an antenna array; [40] And model the electro-magnetic parameters of magnetic materials. [41] These papers provide a reasonable basis for continued research into how the application of machine learning can benefit electro-magnetics problems.

While literature on machine learning applied to IEMI is limited, several works show key proofs of concept.

Villian et al. focus on using a Support Vector Machine classifier on EM signals sent and received by devices using the 802.11n Wi-Fi signal protocol. [42] This body of data was then used to train a classification type SVM to classify the jamming method based on new data. Overall, this paper shows that SVM can successfully differentiate between EM waveforms and IEMI upset outcomes. Importantly though, it suggests that complex

communication protocols and devices are within the scope of capability for machine learning prediction.

Diving into less complex devices, [22] and [43] from Ceperic et al. focus on applying ANNs to improve simulation time and then SVMs to make predictions of IEMI upset. An Artificial Neural Network was trained on EM interference data to model immunity of integrated circuit chips using direct power injection of the IEMI signal and then compared to the simulation time required to determine the same thing. They found that the ANN provided a simulation time that was 2 to 3 orders of magnitude faster. [22] However, The separate study applying Support Vector Machines to make predictions on this data was focused on whether the SVM model produced the same EMI results as a SPICE simulation. [43] Their findings suggest that they achieved a testing accuracy of almost 100% compared to SPICE simulations of the same EM interference. Devabhaktuni et al. performed similar work with ANN to demonstrate how EMI analysis could be simulated and compared to standard EM solvers such as HFSS. Their findings reported errors of less than 5% compared to HFSS. [44]

Artificial Neural Networks were used to model the susceptibility of Integrated Circuits to direct power injected continuous-wave EM disturbances. [45] However, their model required confidential knowledge of the device's internal structure, such as the input/output buffer information specification (IBIS) files and SPICE technical data. Regardless, they achieved a near 100% prediction capability.

The most recent and noteworthy work is the dissertation by Bilalic in 2017, which established that machine learning was well suited to predicting upset on microcontrollers.

[46] His work focused on three machine learning methods: Support Vector Machines (SVM), artificial neural networks (ANN), and gaussian processes for machine learning (GPML). He determined that all three were successful but that there were tradeoffs between the methods related to data size, complexity, and time to train. Bilalic chose to train and validate against raw waveforms to make real-time predictions. However, given that each training dataset contained 191,500 samples, training time was considerable, especially for the best-in-class GPML, which in some cases took tens of hours to train. Regardless, the three methods were found to have an average prediction accuracy of almost 89% and proved the novel result that GPML was appropriate for IEMI prediction making. Moreover, Bilalic's work showed cyclical, but different, trends associated with instruction and clock state, which can be seen by looking at the PoE values both measured and predicted every 10 µs from 20 µs to 80 µs in Figure 10. These results are consistent with the results established in the Vick [24], Camp [35], and Guillette [25] works.

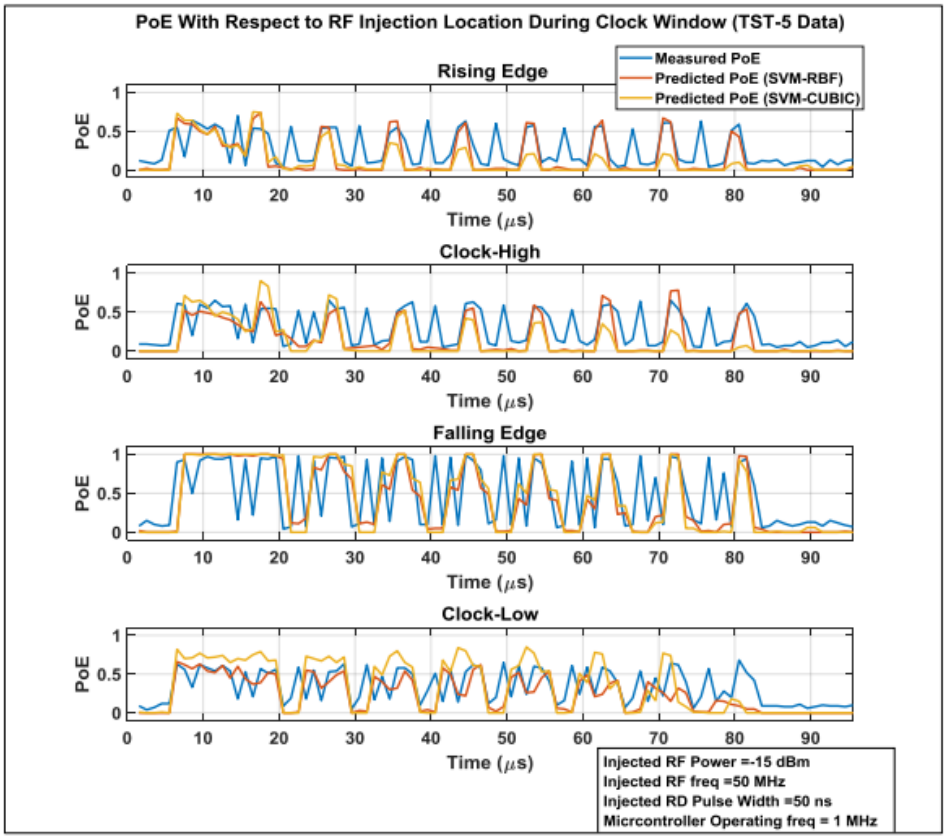

Figure 10: Bilalic's Prediction and Measured Results by Clock State.

This dissertation aims to expand upon Bilalic's work by evaluating fast-to-train machine learning methods, such as k-Nearest Neighbors and Decision Trees. To date, there is no evidence that k-Nearest Neighbors or Decision Trees have been used on IEMI effects type problems. Consequently, the work presented in this dissertation will present a novel result for the application of those methods in addition to the research questions mentioned in Chapter 1.

Chapter 3: Experiment Set Up, Instrumentation, and Devices Under Test

1. Instrumentation and Set Up

Empirical upset data was collected using a custom-built automated testing system. Upset is induced via Direct Power Injection (DPI) of an RF signal. This is referred to as the IEMI event. Direct Power Injection is a technique that pipes an RF signal onto the physically connected point of a DUT. [21] Specifically, in this research, the RF signal is injected onto the clock input pin of a microcontroller. The system – named SALVO (not an acronym) is designed to take in user-defined parameters, which define the waveform characteristics of the IEMI, and automatically initialize and execute injection of the IEMI event at the designed location relative to the clock signal.

After all data has been collected, a set of post-processing tools within SALVO diagnose when upset is present and pull out other data such as the minimum and maximum voltage on the channel. The experimental approach was to build a testbed that would support a wide range of testing articles – microcontrollers - while minimizing the amount of hardware and software changes. It was determined early on that an appropriate way to do this would be to make a break-out board for each microcontroller with a common set of connection inputs and outputs, which would streamline SALVO integration and data collection and processing. Figure 11 depicts the experimental setup and an MCU break-out board for MCU1. Figure 12 shows a diagram of the equipment used in the setup and the flow of all relevant signals.

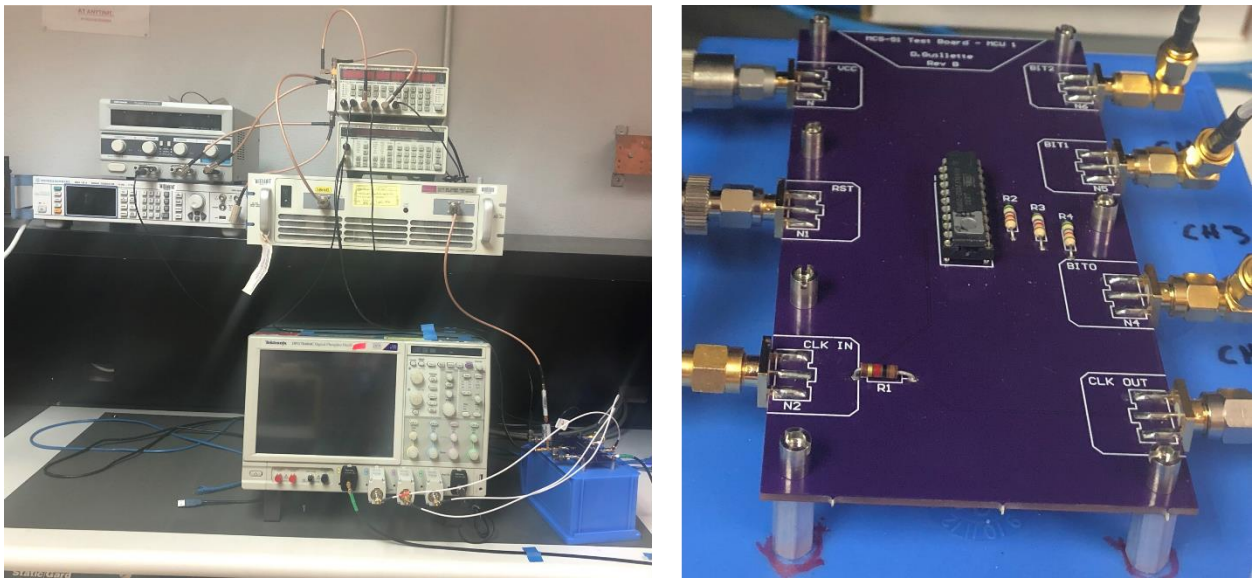

Figure 11: Direct Power Injection Setup (right) & MCU test board (left)

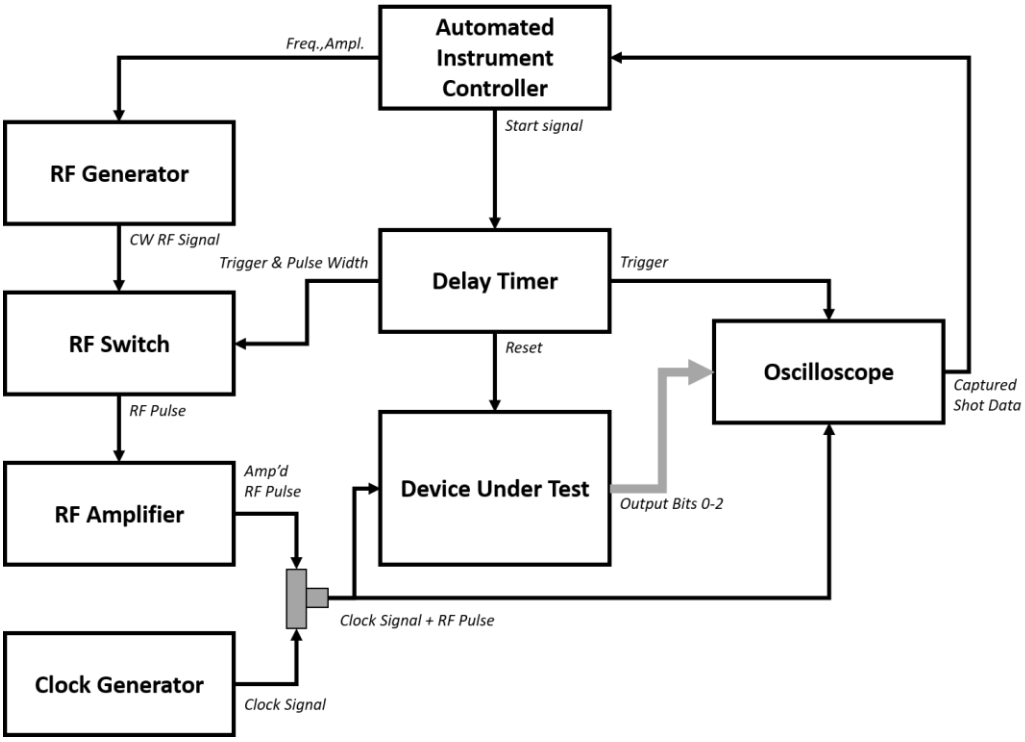

Figure 12: Block Diagram of Experimental Setup

Automated testing and data acquisition are controlled using an in-house developed MATLAB-based instrument control program called SALVO. SALVO takes in user inputs for the RF injection delay (from the start of the microcontroller operation), the

RF pulse duration, the RF frequency, RF amplitude, Injection Time, and the number of repetitions. Then, SALVO automatically iterates through all relevant parameters based on the user input file.

The automated instrument controller is a laptop PC running MATLAB and the SALVO codes. SALVO reads the input file and distributes the relevant information, such as Frequency, Power, # Shots, or Pulse Width timing parameters, to the correct instruments via General Purpose Interface Bus (GPIB) and Ethernet. Once all the equipment is initialized, SALVO tells the DG645 to activate. This activation signal is designated as the time t0, and all other timing commands are set relative to this. The DG645 then sends signals to reset the MCU, cue the oscilloscope to record, and toggle the RF switch open and close. Resetting the MCU ensures that the MCU's commands take place at the same time relative to t0. When the switch is toggled on, the continuous wave (CF) radio frequency (RF) signal is passed to the amplifier for the duration of the user set pulse width, which results in the RF pulse. The RF pulse continues onward through a bias tee and onto the clock signal line. The MCU output bit values are recorded by the oscilloscope and passed back to SALVO, which saves them to a data archive.

The oscilloscope connected to the MCU monitors the real-time state of output bits 0, 1, & 2. During normal operation, the MCU will toggle the output lines between a logic HI (~5V) and a logic LOW (~0V) such that the MCU will count upwards from zero in steps of one until the MCU is reset. An example of normal operation captured by the oscilloscope is shown and discussed in the section MCU Nominal Operation.

2. Amplifier & Bias Tee Characterization

The RF amplifier used in this setup is an OPHIR solid-state 5124 RF amplifier which provides frequency amplification from 20 MHz to 1000 MHz. Experience has demonstrated that amplifiers rarely have a constant gain; therefore, it is essential to characterize the output performance of the device. This was done both with and without the bias tee to evaluate the nominal and implemented cases. The amplifier's output was measured empirically using an oscilloscope. 30 dB of attenuation was added to ensure that the oscilloscope was not damaged. Multiple frequencies and power levels were evaluated. Figure 13 shows the resulting plot without the bias tee present, while Figure 15 shows the resulting plot when the bias is present. The main takeaway from both plots is that an injection power level of -33, -30, -27, and -24 dB for all tested frequencies is considerably less than 1 W and, given their low values, are hard to differentiate between. When testing at power levels greater than -15 dB, there can be a considerable difference in output power at different frequencies. For example, at -9 dB, a difference of more than 4 W between 20 MHz and 1000 MHz can be seen. This significant variation at higher powers is important because not all frequencies achieve the same injection power level. Therefore, the upset threshold for select frequencies may not be observed. Furthermore, this may inherently provide a skewed result that suggests upset is not present even though it simply was not tested across the same power levels. The prime example here is that at an input power of -3 dB, the 20 MHz frequency has an output of 12 W, while 1000 MHz has an output of nearly 8 W.

The bias tee used in this experiment was custom-made to ensure that it appropriately covered the relevant bandwidth range of this experiment. Moreover, the bias tee is required to isolate and mix the clock and injected RF signals. The bias tee is

comprised of 4 components: 1) A DC-1050 MHz low pass filter, 2) a high pass filter from 20 MHz to 1000 MHz, 3) a simple SMA three-post female-female-female-coupler, and 4) a male-male SMA coupler. An image of this bias tee is in Figure 14. Given that the filters are in line with the RF amplifier's output, it is important to understand what the power out looks like from the whole fixture. The output that is representative of the experimental setup is Figure 15. Consequently, the variation of output with respect to frequency with the -3 dB input power level as 50 MHz outputs the most power with about 13 W, while 1000 MHz only has an output of about 5 W.

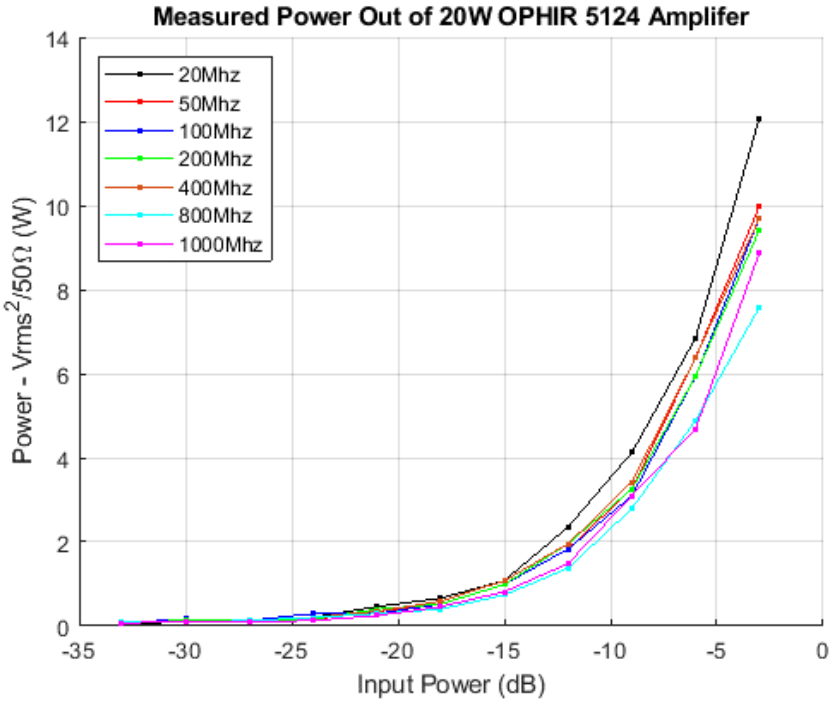

Figure 13: Plot of Amplifier Output vs. Input Power and Frequency

Figure 14: Custom Bias Tee

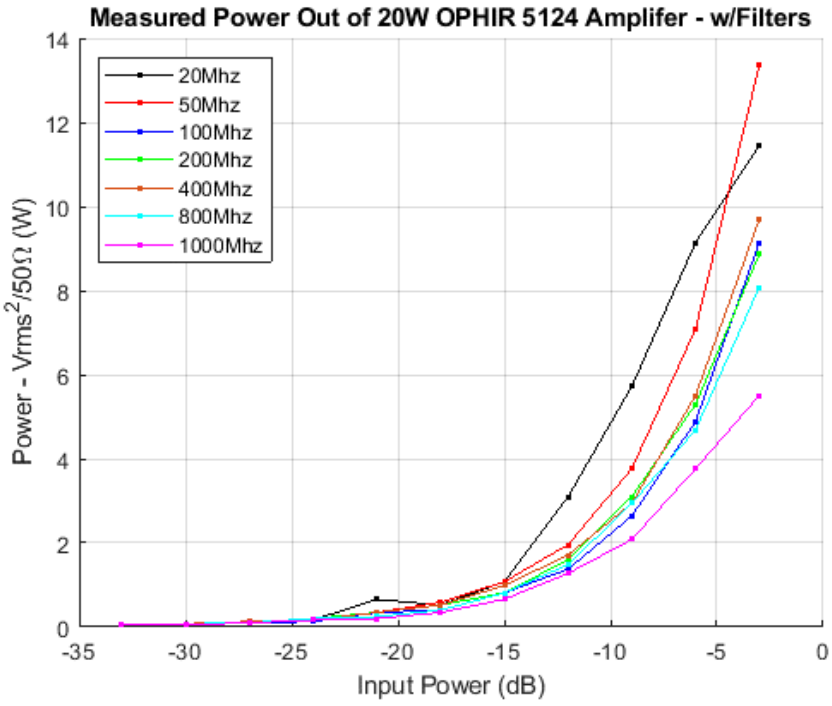

Figure 15: Measured Amplifier Power Out with Filters Present

3. Counting Program & Microcontroller Programming

Each microcontroller is programmed in assembly language to execute a simple binary counting program. The output lines of the MCU are monitored to observe the counting state and establish whether an upset has occurred. An exact list of the instructions which are executed - for MCU1, 2, &3 - and their timing relative to t0 can be found in Table 3, while MCU4's instructions can be found in Table 4. [47] [48]

MCS-51 and PIC developer boards from MikroElectronika were procured to program the microchips properly – see Figure 16. Each board has sockets that support the chosen MCUs. Furthermore, a basic flashing program is supplied with the boards to program the chips easily. ASEM51 to assemble the Assembly code written programs so that they are not abstracted or changed. ASEM51 is an open-source assembler that replaced the official assembler supplied by Intel when support was ended for the architecture.

The PIC device – referred to in this research as MCU4 - has a different instruction set with different timing. Furthermore, due to the complexity of the device, considerable code is required to set up the device. For example, the code required to perform the simple counting program contains only ten lines of code, while the setup for the device requires 36. Table 4 presents the ten lines of code, while the complete code can be found in the Appendix. In comparison, only 15 lines of code are required to initialize the registers, and run the MCS-51 counting program for MCUs 1-3. MPLABX was used as the assembler and programming development environment for the PIC device MCU4. MPLABX was chosen because it supported Assembly programming, which many other integrated development environments do not support. Manual Assembly programming is

considered a “thing-of-the-past” because most compilers prefer to optimize the instruction sets based on the device and therefore transform C-type code – or another similarly higher-level coding language - into the machine code interpreted by the actual MCU.

Table 3: MCS-51 Assembly Instructions for Counting Program

MCS-51 Instruction	**Start Time (μs)**	**Number of Cycles**
NOP	1	1
NOP	2	1
NOP	3	1
NOP	4	1
NOP	5	1
MOV SP,#080H	6	2
MOV 0C2H,#001H	8	2
MOV 0C3H,#001H	10	2
MOV -1,#001H	12	2
L0011:		
MOV A,P1	14	2
ADD A,#001H	16	2
MOV P1,A	18	2
SJMP L0011	20	3

Table 4: PIC Assembly Instructions for Counting Program

PIC Instruction	**Start Time (μs)**	**Number of Cycles**
Start		
CLRF LATA	4	1
CLRF TRISA	8	1
CLRF PORTA	12	1

Main		
MOVFF PORTA,WREG	20	3 (executes in 2)
ADDLW 01h	24	1
MOVWF LATA	28	1
GOTO Main	36	2
End		

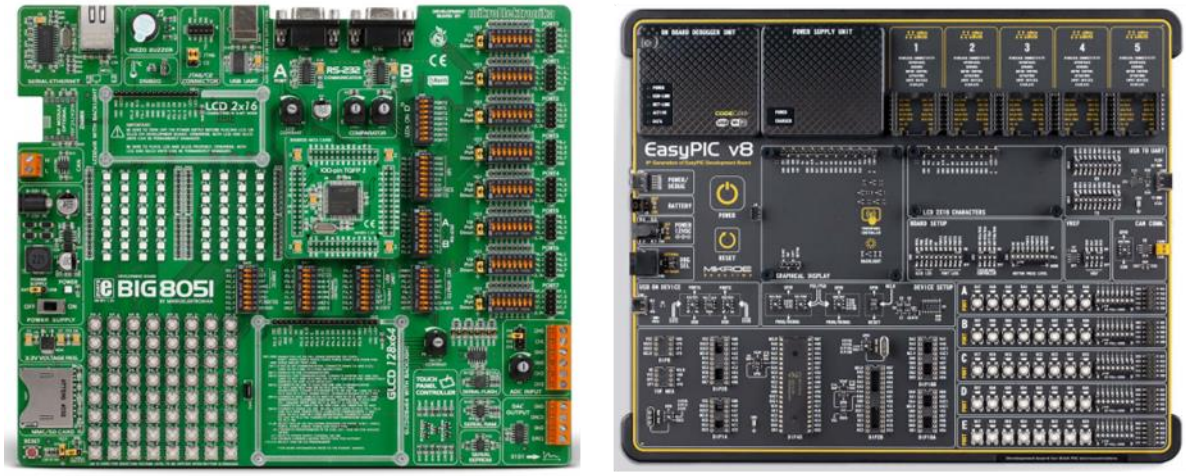

Figure 16: Developer Boards for MCS-51(left) PIC (right)

4. Nominal Operation of MCU Executing the Count Program

An example of nominal operation - captured by the oscilloscope - can be observed in Figure 17. As the clock (CLK) changes state from HIGH to LOW, the Assembly commands programmed into the microcontroller are executed in sequential order at the rate of the input CLK. The CLK frequency used in this experiment is 1 MHz; therefore, the period of the CLK is 1 µs. The exact time at which an assembly operation is performed can be determined if a consistent CLK frequency is provided to the MCU.

Furthermore, this means that the counting program instructions take place at the same relative timing location with respect to the start of the program and t0. The first instruction that changes the output bit values takes place at the rising edge of the 16th clock signal, changing the result to 000. At the 21st rising edge of the clock signal, Bit 0 is changed to one, altering the overall count to 001.

At any given time, the counting value of the MCU can be determined by inspecting the instantaneous value of the output lines (Bits 0, 1, & 2). These bits, when ordered correctly, describe a decimal number using the binary counting method. For example:

If,

Bit 0 = 0; *Bit 1 = 1;* *Bit 2 = 1*

Then, according to the binary counting method,

*[Bit 2] * 4 + [Bit 1] * 2 + [Bit 0] * 1 = Decimal value*

*[1] * 4 + [1] * 2 + [0] * 1 = 6*

5. Definition of Upset

Over the years, the term upset has been used to define a broad range of observed and unobserved responses following (immediately, delayed, or otherwise) an asset's exposure to IEMI. Therefore, to determine the upset threshold of a particular asset, the criteria for upset must be clearly defined. Throughout this experiment, the MCUs have displayed three different responses when IEMI was present:

Response 1: No effect. In this case, the output count values match the expected values at their corresponding times, suggesting that the MCU performs nominally. Figure 17 shows an example of the RF pulse injected on the CLK (around 20 μs mark) and the Bit

count values toggling correctly. For example, the count changes correctly from 000 to 001 at around the 25 µs mark.

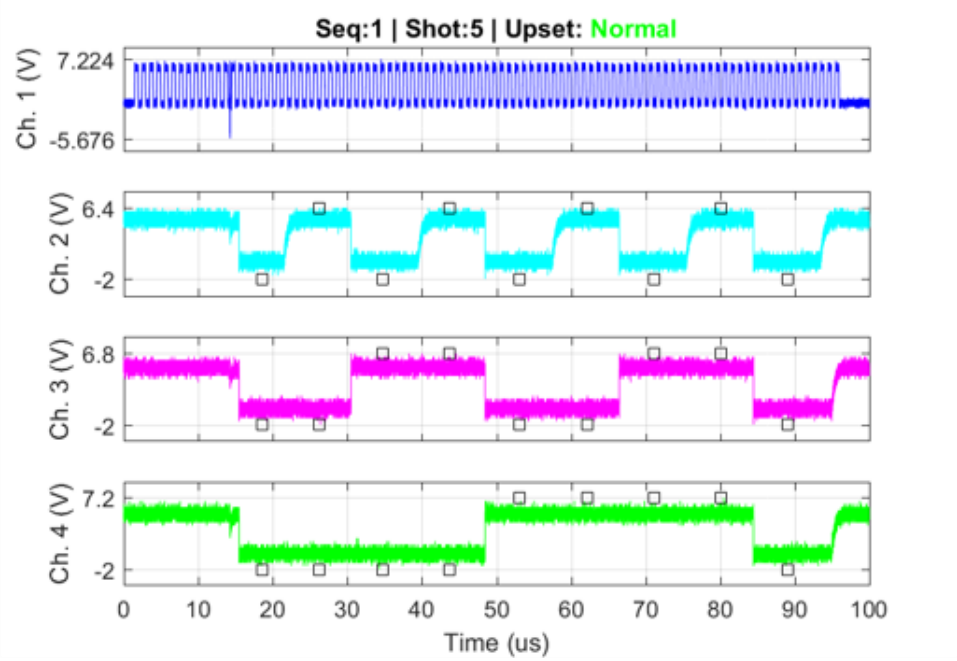

Figure 17: Example of Normal Operation in MCS-51 MCU

Response 2: Latch upset response. Figure 18 shows that all three output lines are latched to zero, which means they are not toggling on and off at their expected times. The superimposed black square markers represent the expected clock states over the 100 µs time period. Note that the black square markers in Figure 17 line up with the recorded output state for each Bit.

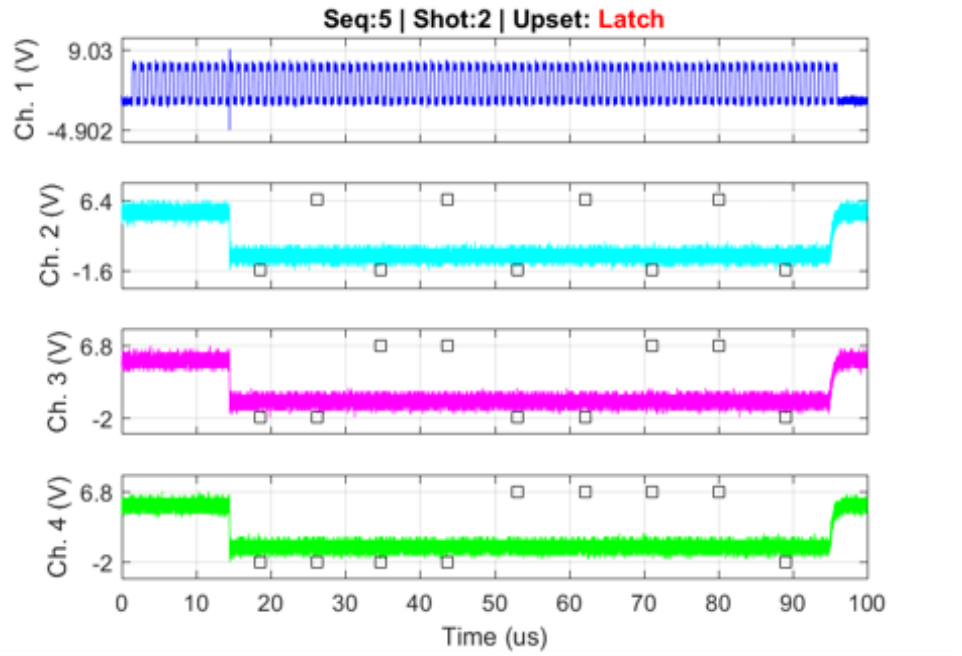

Figure 18: Example of Latch Upset on MCU

Response 3: Shift or miscount upset response. Figure 19 shows that the three output lines are shifted a couple of cycles later. Consequently, a miscount occurs when computing the values at the expected times. In certain time-synchronous implementations, a shift upset could be just as debilitating as a latch upset. Consequently, any recorded state that is not exactly what the count should be, based on the known nominal state, will be considered an upset.

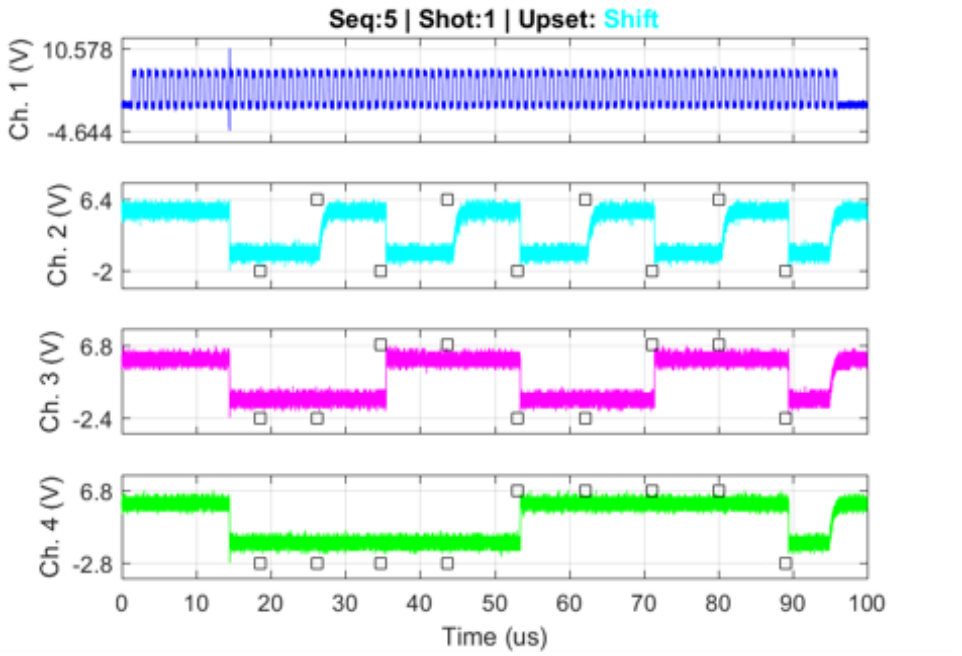

Figure 19: Example of Shift Upset MCU Response

When processing the datasets for this effort, data points resembling Response 2 or 3 are categorized as upset data. Although there is a clear difference between Response 2 and Response 3, both suggest that the pulsed RF compromised the output bitstream. Future studies may desire to explore the nuanced difference between these two responses, but at this time, they are claimed to both be representative of system upset.

6. Probability of Upset

Probability of Upset (PoU) is defined and computed by dividing the number of upsets observed by the number of samples taken. For example, if there were ten upsets over 100 samples, the PoU would be 0.10 or 10%. Figure 17, Figure 18, Figure 19 would each be considered a sample of data. This method of computing PoU is typically considered the traditional way for discussing the stochastic nature of effects data. However, in this research, only Chapter 5 will utilize this method of quantifying upset.

7. Microcontroller Selection

Four microcontrollers were methodically selected to quantify the differences in upset trend between: a) the same device but a second chip from the same lot, b) a different chip but the same architecture, and c) a chip with a different architecture. MCU1&2 are an MCS-51 architecture device with model number AT89LP2052-20PU. MCU3 is an MCS-51 architecture device with a different package, pinout, and model number of AT89LP213-20PU. Finally, MCU4 is a PIC-based architecture chip with model number PIC18F26K42.

These microcontrollers were selected based on their simplicity, availability, and suitability to integrate easily into the testing apparatus. In addition, the PIC18 device was chosen specifically for its dissimilarity – the MCS-51 based microcontrollers were developed based on the principles of some of the very first microcontrollers ever developed, while the PIC18F is considered a modern and advanced microcontroller even when compared to other PIC architecture chips.

8. Break-out Boards

To ensure nominal operation, proper injection of our IEMI, and consistency when changing microcontrollers, it was determined that Printed Circuit Boards (PCBs) should be designed and manufactured - one for each of the MCUs used. Each board would need minor changes based on the device's pinout but would ultimately have the same general functions, inputs, outputs, and supplementary components. Furthermore, Direct Power Injection requires that these boards have some sort of compatibility with RF cable connections, such as SMA or N-Type connectors. Consequently, the boards were designed to have an identical set of matched 50-ohm input and outputs, each placed in the same relative location to the fixed size PCB. The only other circuit components included on these boards are three pull-up resistors, one for the output lines, a socket for each MCU, and a frequency-stable carbon 1000 Ohm resistor to act as a voltage probe for the clock measurement line.

The PCBs for MCU1-4 can be seen in Figure 20. The circuit boards were designed using Circuit Studio. Each board is a simple two-layer board with ground planes on the top and bottom copper layers to minimize cross-talk between the pins.

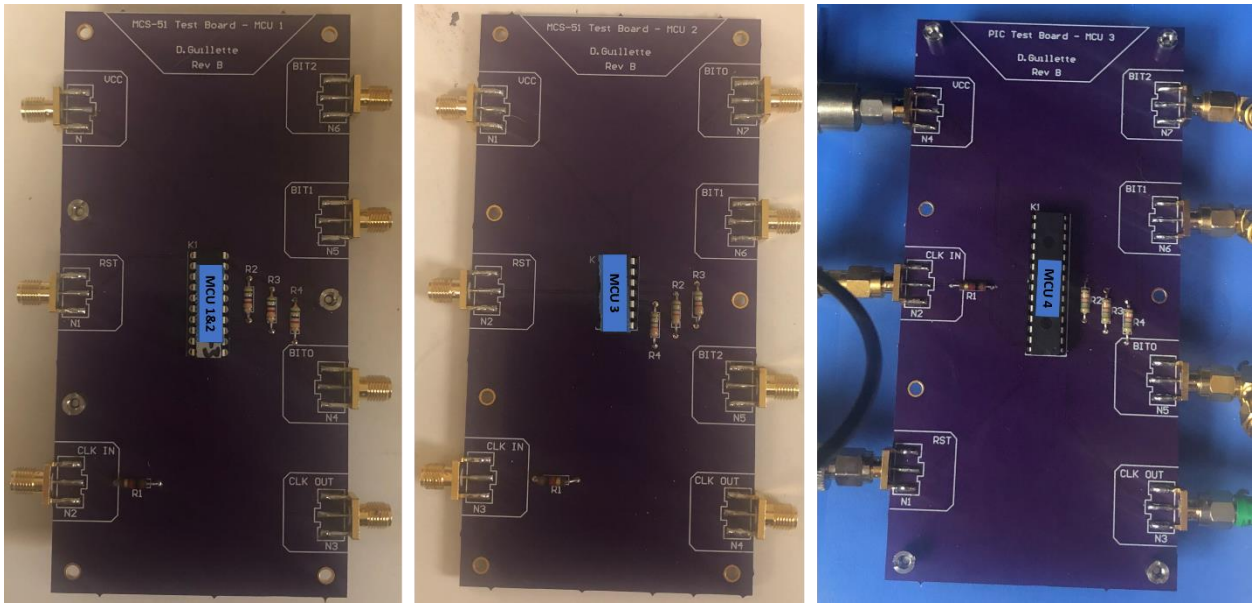

Figure 20: Printed Circuit Boards for MCU1&2 (left) MCU3 (middle) & MCU 4 (Right)

9. Testing Parameters

The instrumentation connected to SALVO can be configured to perform various actions and tasks. However, only a handful of the "experiment knobs" are adjusted for this work. The SALVO setup was exercised and checked in an iterative approach to ensure consistency and proper operation during IEMI testing. The list of variables below directly corresponds to experiment knobs that can be tweaked and their minimum and maximum values – see Table 5:

Table 5: Potential Testing Parameter Trade Space

Variable	**Min**	**Max**
Frequency (MHz)	20	1000
Pulse Width (ns, FWHM)	25	1000

Input Power (dB, 20W CW)	-150	0
Injection Time (us)	0	100

Considering the full parameter range, it was clear that the entire testing space would not be appropriate to collect. The rationale for this is the following, 1) the testing space must be discretized at some level, 2) data collection takes time, 3) data processing takes time, 4) a key aim of this work is to determine the scope of data required to produce a classifier with 75% accuracy or greater & 5) it is against the philosophical approach of this research to collect everything. However, experimental best practices dictate that it is best to collect all the data desired before changing the experiment setup; Otherwise, unknown or untracked changes may be introduced into the datasets when the experiment was changed. Consequently, a series of trials were performed to assess the spectrum of test criteria that would be most suitable for this research effort. Upon conclusion, the following testing parameters were chosen – see Table 6:

Table 6: Chosen Testing Parameters

Variable	**Discrete Samples**	**Steps**
Frequency (MHz)	20, 50, 100, 200, 400, 800,1000	7
Pulse Width (ns, FWHM)	25, 50, 100, 200, 400, 800	6
Input Power (dB, 20W CW)	-27, -24, -21, -18, -15, -12, -9, -6, -3	9
Injection Time (us)	6 to 8 us with even spacing	80

Given the lower powers identified in the amplifier & bias tee characterization section, powers below -33 dB nor frequencies beyond 1000 MHz make sense, as they produce no useful information and needlessly increase the amount of data to collect. Therefore, the frequency values were selected to capture the entire frequency spectrum of the RF amplifier while minimizing the number of discrete steps in the space.

The pulse width values were selected by simply successively doubling the minimum pulse width. The minimum value of pulse width was determined experimentally. Best of class RF switches have a rise & fall time that can often be significant with respect to the timescale of operation. For the installed RF switch, the rise time was on the order of 15 ns, while the fall time was about 5 ns. Therefore, the absolute minimum amount of time the switch could be toggled on and off would be 20 ns. However, this 20 ns time period would not guarantee that the amplitude of the CW RF would be fully observed depending on the frequency present and where the CW wave was in its phase when the switch was toggled on. Therefore, it was decided that leaving the switch on for an additional 5 ns between the rise and fall would ensure that the peak amplitude would be observed.

It should be noted that a pulse width of 1000 ns is equal to the clock period of MCU operating at 1 MHz. Therefore data at this pulse width may be uninteresting at high powers since the injected RF would completely dwarf the clock period guaranteeing upset. However, it should be collected in a future experiment for posterity.

The signal amplitudes –also known as the input power - were selected to correspond to a change in 3 dB steps. A minimum amplitude of -27 dB was chosen as this value was identified in experimentation to show no effect and, therefore, would be

considered a good starting point to capture nominal operating conditions against all other variables. An amplitude of -3dB was selected as the maximum based on firsthand experimentation as that power commonly produces an upset response.

The injection timing was set to capture eight points evenly spaced along the rise and fall of a single clock period. For clarity, they are as follows: 1) the middle of the low, 2) the start of the rising edge, 3) the middle of the rising edge, 4) the high of the rising edge, 5) the middle of the high, 6) the start of the falling edge, 7) the middle of the falling edge, and 8) finally the end of the falling edge. The diagram seen in Figure 7 shows this pictorially.

Since the MCUs operation takes place in a loop ending after the 22^{nd} clock cycle, it was concluded that it would be redundant to collect data past the 23 µs time period. Furthermore, it was shown in [26] that changing the instruction did not change the probability of upset, so instead, only a single instruction worth of data would need to be captured. Consequently, the time 6 to 8 µs was selected as this time range corresponds to the instruction, which increments the counter by one. The instruction requires two clock cycles to complete; therefore, both cycles were captured.

10. Collected Data a.k.a Feature Data

The data collected and synthesized for machine learning is summarized by the following ten features: frequency, pulse width, input power, inject time, measured power, V_{rms}, V_{p2p}, V_{min}, V_{max}, E_{max}.

Of the ten features, only frequency, input power, pulse width, and inject time are variables that were adjusted in the experiment. Therefore, they are the core features,

while the other features are considered ‘engineered’ features – or rather, they are features that are a byproduct – derivative - of one or more of the core features. For example, measured power is both a core feature and an engineered feature because the instrumentation used to change the amplitude of the IEMI signal is executed via the input power feature, but the instrumentation has a different output power for a constant input power based on the frequency.

A specific definition of each feature can be found below:

Frequency describes the rate at which the IEMI signal oscillates. This value is in the units of megahertz (MHz).

Pulse Width describes the duration of the IEMI signal pulse. This value is in the units of nanoseconds (ns).

Inject Time describes the time location relative to the clock signal start where the IEMI pulse is injected. This time relates to the start of the IEMI event, meaning that the IEMI takes place from the inject time to the inject time plus the pulse width. This value is in the units of micro-seconds (µs).

Input Power describes the amplitude of the IEMI signal at the peak and trough of the Frequency oscillation. This value is in the units of decibels (dB) and is created by a 1 mW signal from an RF sweeper amplifier by a 20W solid-state amplifier. The characteristics of the amplifier and its frequency to gain response can be found in [4]. The power output of the amplifier is known as the measured power.

Measured Power describes the power out of the IEMI generating instrumentation which is injected into the DUT. Since every DUT will have a different coupling response

(transmission and reflection based on frequency), tracking the known power sent into the DUT is important. This value is in the units of watts (W).

E_{max} describes the integrated total of energy directed into the DUT. Combining two features creates E_{max} – In this work the value is computed by multiplying pulse width by measured power. This value is in the units of joules (J).

V_{min} describes the minimum voltage measured of the IEMI pulse. This value is in the units of Volts (v).

V_{max} describes the maximum voltage measured of the IEMI pulse. This value is in the units of Volts (v).

V_{p2p} describes the peak-to-peak voltage measured between oscillating cycles of the IEMI pulse. This value is in the units of Volts (v).

V_{rms} describes the root mean square voltage measured between oscillating cycles of the IEMI pulse. This value is in the units of Volts (v).

When combined, the set of results for each microcontroller device contains 30240 different feature combinations whose upset outcome can be quantified. Each sample contains all ten of these features and a label describing whether upset resulted with the criteria or not.

Chapter 4: Theoretical Considerations of Machine Learning

This chapter focuses on the fundamental concepts related to the machine learning (ML) algorithms applied in this dissertation. The theoretical considerations for the three methods - k-Nearest Neighbors, Decision Trees, and Support Vector Machines - are detailed. Additionally, a discussion of the rationale for selecting the methods and fundamental concepts germane to all three algorithms, such as supervised vs. unsupervised learning, training accuracy vs. error, regression vs. classification, and the confusion matrix, is provided. Readers should know that based on the experimental results presented in Chapters 6-8, k-Nearest Neighbors is the primary focus of this dissertation, but, for continuity, the other two methods are described in detail.

1. Transparency of the Black Box

A primary challenge of machine learning - especially in methods like Support Vector Machines and Artificial Neural Networks – is interpreting how the trained classifier ended up with its conclusion. Consequently, machine learning has gotten a bit of a reputation for being a black box with inputs and outputs, where users and experts do not know what is happening to the inputs to create the output. This often occurs because ML algorithms implement complex mathematics to crunch the data, which is then expanded into more than four features. Human beings are known to interpret trends and patterns up to three and four dimensions but struggle beyond that. This concept is explored in literary texts such as Flatland, which suggests that living beings can, at best, understand and interpret the number of dimensions in which they inhabit or naturally observe. [49] Human beings inhabit this world in three dimensions and can observe a change over time

in those dimensions, which gives them observation of four dimensions. The work performed here has four core features and six derivative features, meaning that, at best, humans may be able to understand the interdependency between the four core features but will not be able to handle all ten. Adding complexity is that the data has two states, upset and not upset, which are associated with each of the ten features. Effectively, this increases the problem's dimensionality to five features using only the core variables or eleven if all are contained.

Machine learning is well suited to tackling this problem at both five and eleven features, which is why it is the primary focus of this work. However, it is critical to understand that although the ML classifier may predict phenomena occurring in the data, the end-user also needs to trust that the result makes sense. Moreover, it is also desirable for the end-user to gain further inherent knowledge of the interconnectivity between the features through the ML classifier. Artificial Neural Networks (ANN) are particularly notorious for being “black boxes” because they often have multiple layers of nodes and will even nest other ANNs within one another, making it nearly impossible for humans to understand how the ANN reached its outcome. When complex algorithms – particularly programming codes - are obscured, it is often referred to as abstraction.

Some researchers believe that the way to overcome these abstraction issues is to have a strong foundation of machine learning coupled with a deep understanding of the data. However, some mathematical operations or data sets are still too complex for a human to interpret correctly. Consequently, this work focused on machine learning methods that are mathematically straightforward.

In practice, there are no hard and fast rules for determining whether one method or set of parameters for a given method are better than another. Instead, it is advised that a few should be tried, and the user should select the best performer – assuming the result is not an overfit solution.

Overfitting is the term often used to describe what happens when a classifier is trained with data that negatively biases its classification predictions. [50] A generic example would be a classifier trained so carefully to predict round-shaped fruit to be oranges that it has excellent performance in only that and will often misclassify grapefruits, grapes, and apples as oranges. Training classifiers can sometimes be considered an art because the "right data" needs to be used, the "right way," to result in a "useful" classifier. Because of this, understanding the scope and quality of training data is important. A common adage associated with machine learning is, "Garbage in. Garbage out." This phrase embodies the concept that, when training a classifier, the quality of the data will influence the quality of the classifier. [51] Therefore, considerable effort is put into matching data with the proper machine learning method and then testing and refining the results iteratively.

2. Classification vs. Regression

In machine learning, the result of a trained classifier is usually one of two primary options: Regression or Classification. The most straightforward difference between the two is that Regression results in a continuous output value, while Classification results in one that is discrete. [52] Regression is described to be a numeric probability value that describes the likelihood to which a new data point is given the same label as another data

point assigned during training. Consider the example of a classifier trained with images of “cats” and “dogs” - A new data point is presented to the classifier, which responds with an 89.7% likelihood of being a “dog.” Moreover, Regression can also specify a value as part of a range. For example, a dataset of the height, weight, and gender data of a series of family members at different ages in their lives. A classifier trained with this information may predict the future numeric height of a young individual based on their current physical characteristics.

Alternatively, Classification specifies an answer without a continuous numerical value. Using the cat and dog example again, the classifier would predict that the new data point is classified as a ‘dog.’ However, this method would not be appropriate for determining the future height of a person because it would not produce a numerically tied value.

This work applies the Classification method because the key goal is to predict whether new data is ‘upset’ or ‘not upset.’ It can also be implemented to classify between more than two outcomes, but this work does not pursue that. Furthermore, Regression is not applicable in this case because the intent is not to quantify a probability of likeness.

3. Supervised vs. Unsupervised Learning

Once the outcome of a machine learning algorithm is defined, it is important to determine how to train it – there are two methods: Supervised and Unsupervised. The difference between Supervised and Unsupervised learning is that Unsupervised learning

does not specify what the classification label outcomes are, while in Supervised learning, the outcome is known ahead of time. [53]

For example, consider a dataset of fruit in a supermarket. After training, the classifier has identified four different types of fruit. When new data is presented, the classifier simply states which of the four groups the data point belongs to. It would then be the role of the user to try and relate each of the four types of data to names that are meaningful to the user. Therefore, Unsupervised learning is more commonly used in datasets in which the user wants to learn whether data can be binned and grouped, and if so, how many bins there are. Supervised learning is the opposite because the user provides example data with labels for each fruit in the market for the classifier to train on.

This work applies the concept of Supervised learning. Unsupervised learning could also be applied here. However, it was decided early on that Supervised learning was better suited because 1) Data was available, 2) The outcome for prediction was known, & 3) Supervised learning is well-matched with Classification type problems.

4. Feature Selection and Validation

There are many ways to select data for training and validation, and there is no one right answer. UNM Professor Manel Martinez-Ramon has been quoted many times saying that Machine Learning is an art. However, some researchers are limited by computational power, data, or practical consideration, which will lead them to select specific methods that will help reduce the time required to train or error within a model.

This work was fortunate not to have computational or time limitations. Therefore, the datasets were exhaustively tested using all combinations of the data and to compare them based on their accuracy – this would be considered a quasi-brute force approach. The specific combination method is described in the next section.

5. Feature Combinations

The Combination Formula - also known as the "n choose k" equation - was used to determine an efficient solution for iterating through the entire range of feature pairings. *k* is replaced with *c* in this paper to reduce confusion between the "k" defined in the k-Nearest Neighbors algorithm. This method is appropriate because 1) the order of the features does not matter, and 2) no feature can be repeated. The Permutation Formula would not apply here because the order of the features would matter in that formula. Consider the example of a two-feature classifier training with frequency & pulse width is the same as a classifier training with pulse width & frequency. The Combination Formula is of the form [54]:

$$\binom{n}{c} = \frac{n!}{(n-k)!\,c!} \qquad (1)$$

Where 'n' is the number of unique samples in the dataset, and 'c' is the number of samples to be selected at a time. The result of this formula, when combined with the ten features and letting 'c' go from one to ten, is 1023 different combinations. The number of combinations for each value of 'c' is shown in Table 7.

Table 7: Number of Combinations for a Given c

c	*10*	*1,9*	*2,8*	*3,7*	*4,6*	*5*
Combinations	1	10	45	120	210	252

An alternative way to think of this is that each of the features can be perceived as present or not, which results in 2^10 combinations, which equals 1024. However, one of the combinations is that none of the features are present, which is irrelevant for this work, so 1024 is reduced by one.

6. Holdout Validation Method

The method of validation used was a 25% holdout method. The holdout method works by randomly removing a user-defined amount of data before training the classifier. Then once the classifier is trained, the removed data is used to test how well the classifier can classify the data. [55]

For a specific example, consider a dataset with 100 data points (n = 100) and a user set 25% holdout; 25 data points would be set aside for validation, while 75 data points would be used in training. Some researchers will describe this as creating two new datasets with labels of D0 and D1 to differentiate between the held-out data (D1) and the remaining data (D0). This method is well suited for large and small datasets because the set of data held out is randomly selected and based on a percentage number of samples – for this reason, whole numbers which match up to the dataset must be selected so that the

data is evenly separated. An example of an incompatible match would be if a user selected a holdout of 27% with a dataset containing 80 samples (n = 80). This combination of samples and percentages would split the data into two sets: one with 21.6 data points and the other with 58.4. Should a user want to do such a thing, an additional rounding function would be needed to ensure whole numbers.

However, this method suffers from a problem of repeatability. Conceptually, the data points removed from the datasets are randomly selected and not the same every time. The impact this has on machine learning is largely dependent on the variety of the data and whether the dataset is well balanced. For example, consider a dataset where ten samples are classified as apples, and 90 are classified as oranges. If the random selection chooses none of the apples, the dataset would be less valuable for training a classifier that differentiates between the two fruits because there would be no data to learn from on one of the two fruit. This example underlines the concept of "trash-in-trash-out" because it suggests how the outcome of a machine learning method can be adversely affected simply by the data used in training. Therefore, it is important to assess which feature combinations result in high-quality classifiers and understand how repeatable this type of result is with respect to data selection and validation.

Consequently, two repeatability studies (where different random sets of traning data were used) were completed to quantify different aspects of the problem. The first study can be found in Chapter 6, while the second can be found in Chapter 7. The difference between the two can be summarized as an economy of scale type trade study. In Chapter 6, the goal was to get a rough answer of repeatability for multiple variants of machine learning methods. In contrast, in Chapter 7, a more refined answer was needed to solidify

the predictive capabilities of each feature or combination of features when used as the training input.

7. Time to Train

Successful application of fast-to-train methods is an attractive outcome for two primary reasons: 1) fast-to-train methods tend to have a straightforward "kernel" which may allow the models to be suitable for, and explained by, human interpretation; and 2) fast-to-train methods are not computationally heavy, nor do they take long to produce a result, making them great for large datasets.

For a method to be fast-to-train, it typically needs to have a basic mathematical back-bone – sometimes referred to as a kernel - for classifying the data quickly, even with large datasets and many features. It is well understood that certain mathematical operations are computationally more complex or time-consuming than others. Therefore, using time-consuming operations would decrease the speed at which a machine learning method is trained [29].

Two algorithms were identified as mathematically straightforward and fast to train: k-Nearest Neighbors (k-NN) and Decision Trees (DT). However, k-Nearest Neighbors was empirically determined – see Chapter 6 - to be the better performing algorithm, and therefore this work focused more significantly on that method than the others.

8. Support Vector Machines

This section introduces the Support Vector Machines (SVM) machine learning algorithm, focusing on the linear classification of two-dimensional datasets. Readers will find a summary of the theory and mathematical constructs which enable SVM in this section. This method is not considered a fast-to-train method, but it was used for comparing the other two methods since it is a commonly used algorithm.

To begin, SVMs aim to classify data by processing a subset of given data to "train" a model. Consequentially, the training model is built to minimize the error associated with the classification function and the training data points. Figure 21 shows a generic example set of data with '+' and 'o' markers cleanly distributed in four clumps. The data points are classified as label Y_1 and label Y_2, while the plot's axis is representative of features related to the data labeled X_1 and X_2. For example, say a scientist wanted to categorize apples and oranges using an SVM, apple and oranges would be the labels. At the same time, color – measured in wavelength – and diameter size would be the features – X_1 would be the color, and X_2 would be the size. Typically, a SVM will select a few data points to define the classifying function – sometimes referred to as a hyperplane – and attempt to maximize the separation distance between the differentiated labels – often called the margin. These anchoring/selected data points are referred to as the support vectors, which can demonstrate a simple SVM in action. A graphical representation of the hyperplane, margin, and support vectors can be seen in Figure 22. The support vectors can be found as the data points intersecting the margin gutters. A simple determination of whether the new data point is above or below the separating hyperplane line will allow it to be cast as either Label 1 or Label 2. [56]

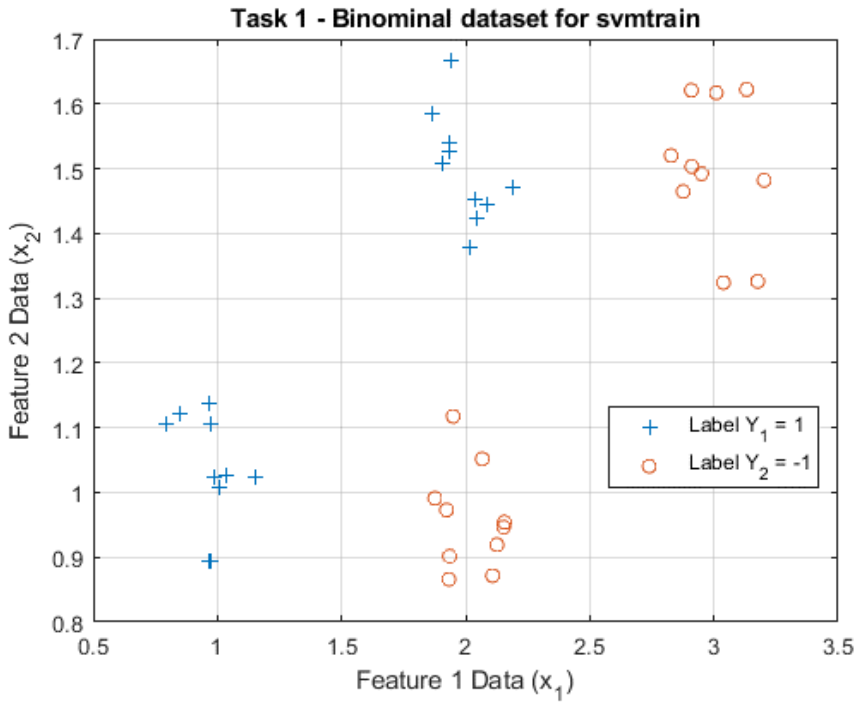

Figure 21: Binomial Data to showcase SVM classification

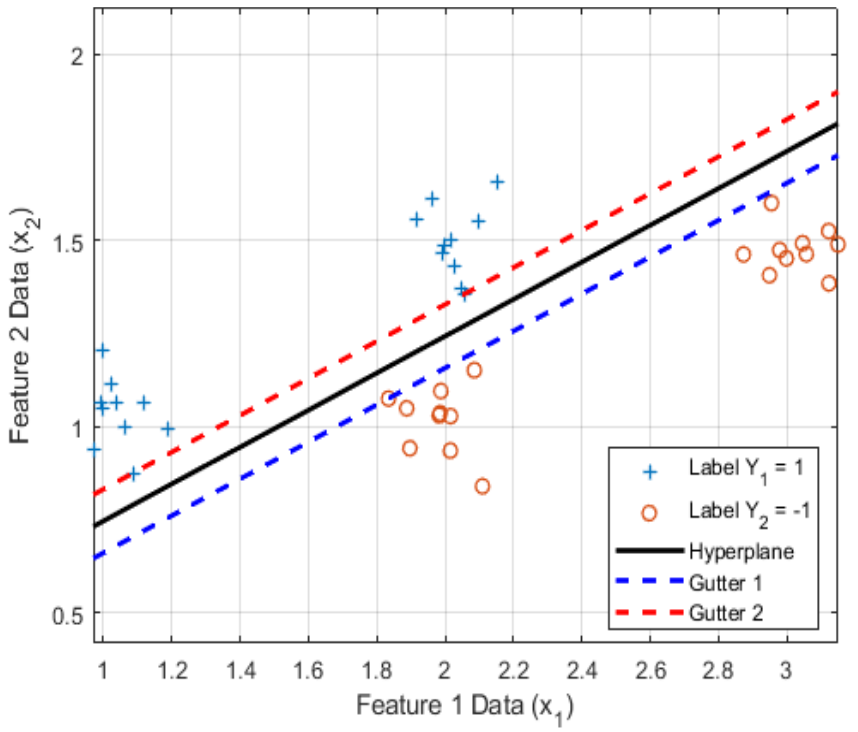

Figure 22: Binomial Data with Overlaid SVM Hyperplane Model and Gutter Margin Bounds

To begin the formal description of a support vector machine, we need to define the variables. First, we need a data set with vectors X and associated labels Y. These values will be used as the training data. The labels will classify the X value as 1 or -1 to define whether the value is above or below the hyperplane. Next, we need to define the loss function because it reduces the error in predicting the unseen data.

$$L(X, Y, w) = E[(Y_n - w^T X)^2] \qquad (2)$$

Where E is the total error and w^T is the weight value for a given X. Solving the loss function can be accomplished by use of the Least Means Square Algorithm [57], and consequently transforms the loss function into the following expression,

$$L(X, Y, w) = \frac{1}{N}\sum y_n^2 + w^T R w - 2w^T p \quad (3)$$

where 'N' is the number of samples, '*n*' is the instance of data selected, 'R' is the auto-correlation of the data X and Y, and 'p' the cross-correlation of the data X and Y. 'R' and 'p' roughly take the forms $x_n x_n^T$ and $x_n y_n$ respectively. Finally, '*w'* will need to take the form,

$$w^{k+1} = w^T - \mu[x_n^T w^k - y_n]x_n \qquad (4)$$

where 'μ' is used to optimize the Least Means Squares Algorithm.

Next, we need to understand that the SVM output is of the form,

$$f(x_n, \alpha) = y_n + e_n \qquad (5)$$

Where $'e_n'$ is the prediction error, and $'\alpha'$ is a parameter to adjust the risk. Consequentially, the risk function needs to be defined as the following,

$$R(\alpha) = \int L(f(x_n, \alpha), Y)\, dF(x, y) \qquad (6)$$

However, the actual value of the risk equation is not often computed; instead, it is approximated as the empirical risk by the equation,

$$R_{empirical}(\alpha) = \sum L\big(y, f(x_n, \alpha)\big) \qquad (7)$$

The goal here is to optimize the risk value towards its minimum. However, this is also where the user must make their first trade. As the system complexity increases, the machine begins to over-fit the data and start to be influenced by the bias of the presented data, not the trend the data is suggesting. To bound this problem, we aim to minimize the structural risk of the system. The structural risk is of the form,

$$R_s = \sqrt{\frac{h\left(\log\left(\frac{2N}{h}\right)+1-\log\left(\frac{\eta}{4}\right)\right)}{N}} \qquad (8)$$

Where 'h' is the Vapnik-Chevonenkis dimension, and $'\eta'$ is the dimension number. The purpose of the VC dimension is to define the maximum number of points which can be shattered – or separated - by the hyperplane – which turns out to be $h = \eta + 1$. Moreover, 'h' acts as a metric by which the complexity can be judged while the actual risk of the system is of the form,

$$R(\alpha) = \sum L\big(y, f(x_n, \alpha)\big) + \sqrt{\frac{h\left(\log\left(\frac{2N}{h}\right)+1-\log\left(\frac{\eta}{4}\right)\right)}{N}} \qquad (9)$$

which is just the addition of the empirical and structural risks.

In summary, the Support Vector Machine method assesses the training data points and casts an appropriate mathematical function which can weave between the different labels. Values above the function are classifier as one label while values below are classified as another. Given the method's high likelyhood to overfit to the data, the risk and error variables are optimized to find a middle ground which results in the highest prediction accurary without being directly anchored to specific data points.

9. Decision Trees

This section acts as an introduction to the machine learning algorithm called Decision Trees. Decision Trees are popular for data mining because they can quickly break down processes and data into functional decision points with consequences or results. [58] Decision Trees can be applied in either a regression or classification method; this research only uses the classification variant.

During training, each branch has an assigned weighted probability of increasing or decreasing the likelihood of an outcome. Because of this data dependency, Decision Trees are usually considered a supervised learning method, and it is unclear how an unsupervised learning version of a Decision Tree might exist. A brute force look up table could be considered similar to this method because the raw data is considered and directly associated to the outcome.. However, the model's value over a brute-force approach is that it interpolates the regions between data points to provide continuity where data is not present, while a brute force look up table would not be able to provide that answer without additional interpretation.

Decision Trees take a similar approach to a brute force look up table except that the data is methodically separated based on a series of IF statements which branch out to

more IF statements leading to a declaration of the outcome. IF statements result in what could be best described as a decision point where there are only two branches: the one leading where the IF statement is true, and the one leading to where the IF statement is false. Both statements will lead to their respective follow-on statements or a terminating outcome based on the programmer's intent. For clarity, IF statements are often paired with ELSE statements to differentiate between the True (IF) and False (ELSE) outcomes. Consider the example: IF the frequency is 10 MHz and IF the pulse width is 50 ns, then the device is not upset; IF the frequency is 10 MHz and IF the pulse width is 60ns, then the device is upset.

Every Decision Tree is filled with IF: ELSE statements that lead to an outcome (a classifier label in classification type implementations). The organization of the tree is then based on the data's connectedness. An easy way to consider the structure would be the components of a biological tree: root, branch, leaf. [59] A diagram of this can be seen in Figure 23. A simple tree would start with one root node that has two outcomes, leading to two "branch" nodes; those branch nodes then lead to four "leaf" nodes.

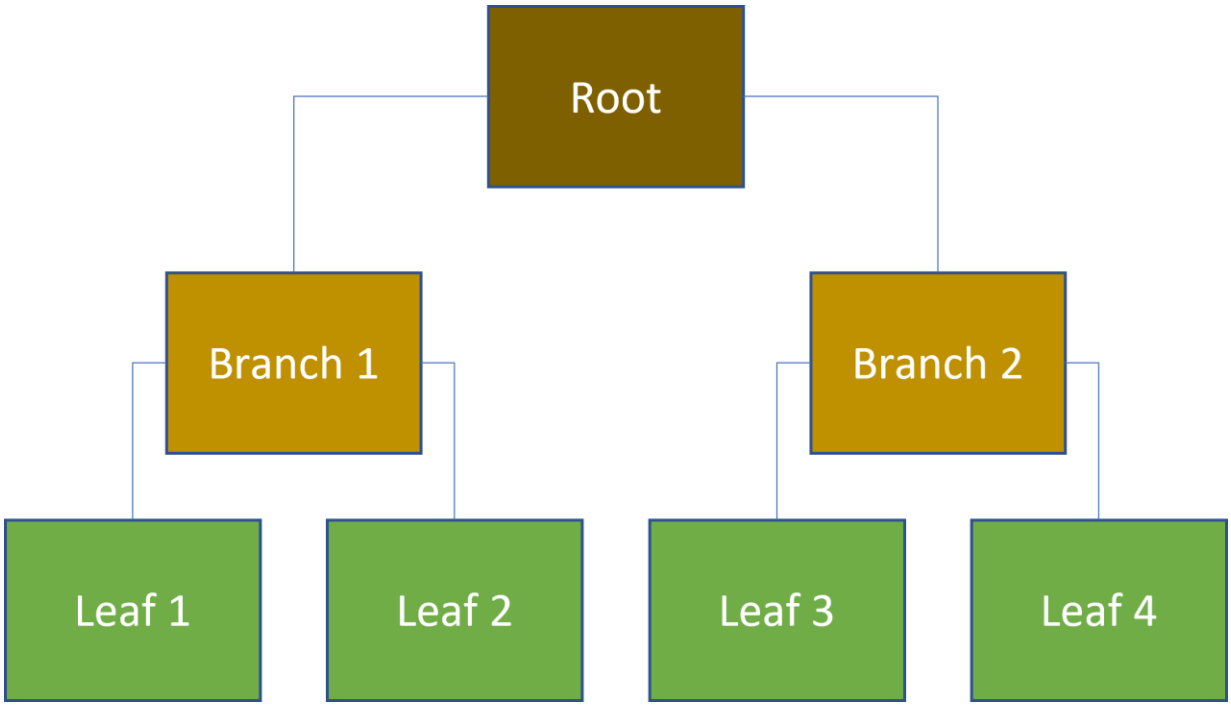

Figure 23: Basic Structure of a Decision Tree

It is essential to understand that based on the user's desired structure, and the number of classification states, there may be many leaf nodes or only two in the case of a binary outcome state. Moreover, there may be many branches or few based on the user's input. Multiple values spanning two orders of magnitude were tested in this work and can be seen in Chapter 6. The desired classification outcome is binary in this work, with the labels "upset" or "not upset" being used. Consequently, the Decision Trees in this work would only have two leaves with those labels.

The mathematics behind how the tree is defined is determined using the "standard CART" algorithm, which is a classification implementation that specifies the branch nodes' thresholds by the Gini Impurity/ Gini Index of the dataset. [60]

The CART algorithm gets its name from the portmanteau of classification (C) and (A) regression (R) tree (T). Therefore CART can produce both classification and regression

classifiers but is limited to a binary outcome. CART works by finding each feature's (and consequently each node's) best split based on the best Gini's Impurity index, which is defined by node (o) as:

$$Gini\ Impurity = 1 - Gini = 1 - \sum_{i=1}^{n} p_i^2 \qquad (10)$$

where 'p' is the fraction of items in the class 'i'. Once the value of Gini Impurity has been determined for a node. The best split is chosen from the full list of possible splits as the one with the lowest value. The last rule of thumb is that if the Gini Impurity of a child node is larger than the Gini Impurity of the parent, then that node should not be split any further.

Consider the following example to best explain the implementation of this equation and how a split is computed.

There is a collection of 100 apples and oranges that need to be classified. Two data features are available: 1) the size of the fruit in centimeters, and 2) the color of the fruit in nanometers. The oranges have diameters of 6-10 cm and a wavelength of 590-620 nm, while the apples have diameters of 4-8 cm and wavelength of 620-750 nm. Consequently, the first split condition found by the CART algorithm may be based on size, and therefore the rule used is "Diameter ≤ 7 cm," while the second split would be based on color and is the rule "Color ≤ 620 nm". The tree resulting would consist of three simple rules, which correctly distinguish between the two fruit – see Figure 24. [60] Frequently the rules are specified directly by the features as a robust tree would contain at least one mode for each feature.

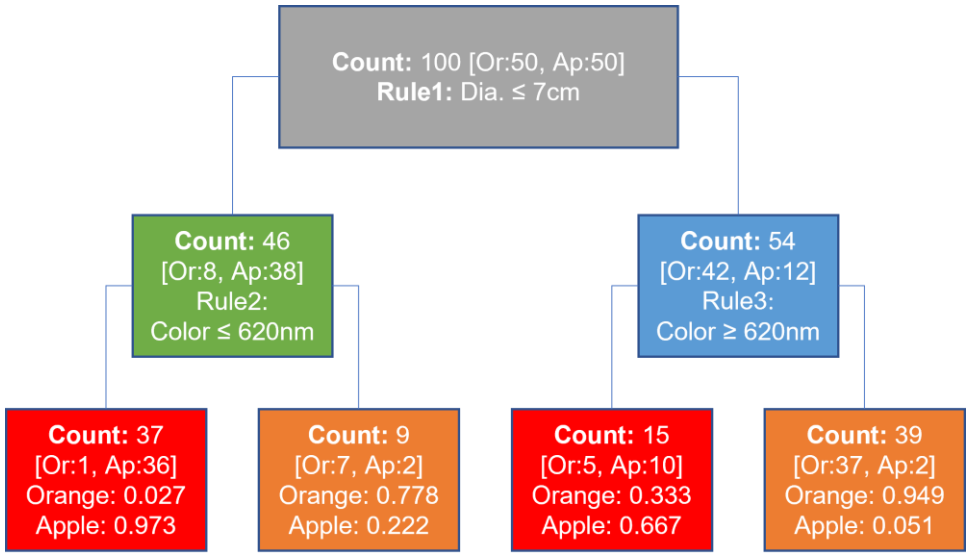

Figure 24: Example of Decision Tree with sample data

Figure 24 shows an example of a decision tree with data. At the top is the root node, in the middle are the branch nodes, and at the bottom are the leaf nodes. In each node is the number of data points relevant to each split, and in the bottom leaves is the fraction of each fruit with respect to the above node's rule criteria. Specifically, there was one orange and 36 apples, so the fraction of oranges is 1/37 or 0.027. The Gini Impurity of this leaf node would then be:

$$1 - (0.027^2 + 0.937^2) = 0.053.$$

Computing the Gini Impurity for the tree seen in Figure 24 results in Figure 25. It should be observed that the leaf nodes 2 and 3 have a Gini Impurity that is greater than their parent node, which suggests that this tree is complete.

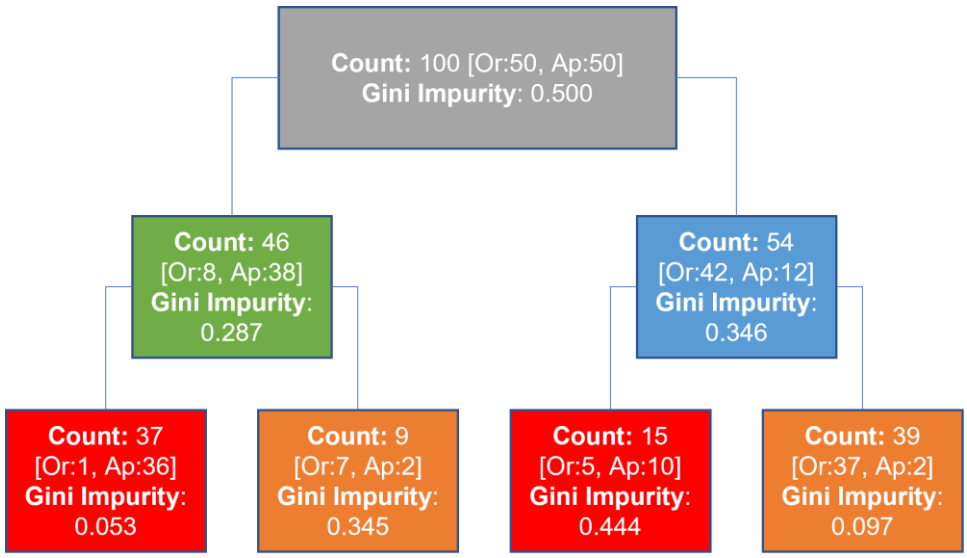

Figure 25: Gini Impurity for Each Node of Example

10. k-Nearest Neighbors Algorithm

This section introduces the machine learning algorithm called k-Nearest Neighbors. The k-Nearest Neighbors (k-NN) algorithm is effectively a data clumping technique that can be used for classification or regression. [61] This research only uses the classification variant. In short, a k-NN classifier evaluates the distance from a new data point, Q, to the "k" data points nearest to it. Then, data point Q is classified to be the same response label as the majority of the "k" nearest neighbor response labels. The majority value is computed using a weight metric and distance metric. There are many ways in which the distance between data points can be computed and multiple ways in which the data can be grouped based on those distances. Multiple books [50] [29] [62] explore the impact of changing the weight and distance metric for a weighted k-NN classifier and suggest that there is no one right answer. However, one of the most common parings contains the Euclidean Distance and Squared Inverse Distance Weight.

The Euclidean Distance metric is of the form:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} \frac{(p_i - q_i)^2}{q_i^2}} \qquad (11)$$

Where 'p' and 'q' are points in the dataset, 'n' is the number of dimensions, and 'i' goes from 1 to 'n'. [63]

The Inverse Square Distance is of the form:

$$\text{ISD} = \frac{1}{d^2} \qquad (12)$$

Where 'd' is the distance computed. As implemented with the k-NN algorithm, the distance would be computed from each of the data points in the set to the new data point 'Q' and would result in a separate distance value for each. Once these values are computed, the 'k' number of lowest values would be used to determine the label of 'Q.' Should an even number of data points be used as the 'k' value, the weight metric is used to break the tie for the label. Once again, this is done by selecting the datapoint whose ISD has the lowest value to 'Q.'

Fundamentally, the k-NN method is considered one of, if not the, simplest machine learning algorithm. Its primary complexity is in the computation of the distance between points, which can be an issue based on the size of the data set. Consequently, feature selection methods are often used to scale down the number of features to improve performance or reduce computation burden. Some methods may even combine features using additional mathematics. However, a mathematical combination of the data can result in features with units that do not have a sensible meaning, and more importantly,

will inform the algorithm to make predictions on mathematically consistent but naturally inconsistent phenomena.

For a classic example of the k-NN method, consider the example situation pictured in Figure 26 of a dataset with features measuring the diameter and color of fruit known to be either Apples or Oranges. A k-NN classifier would be presented with diameter and color data, and it would classify a new point as an Apple or Orange. If k was one and the nearest data point had a predictor value of Apple, then data point Q would be assigned the response label of Apple as well.

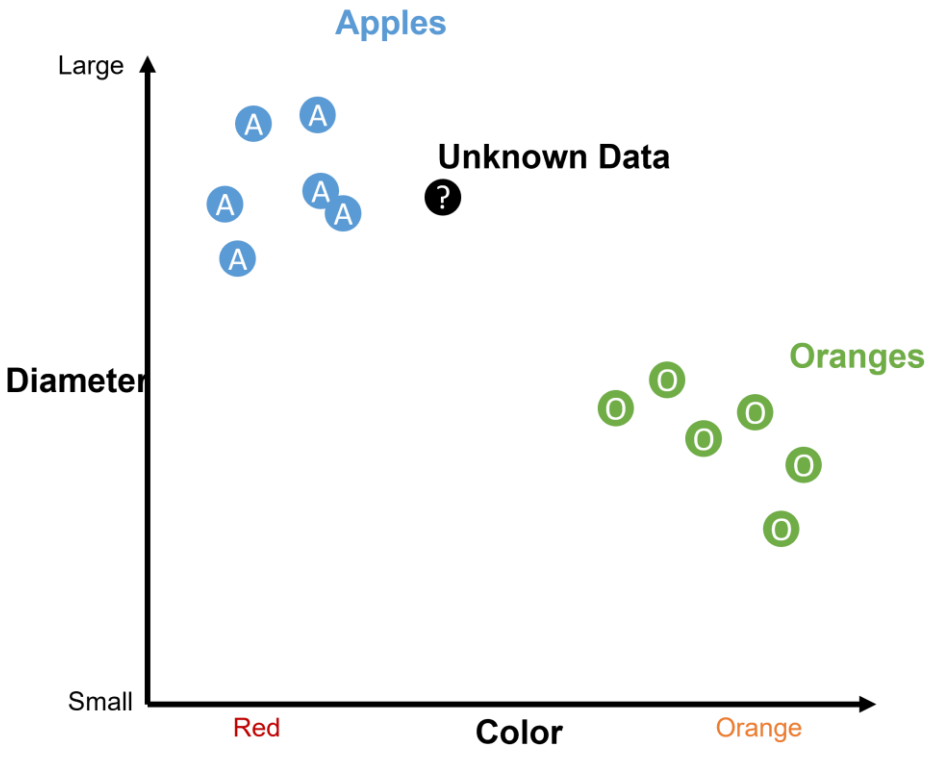

Figure 26: k-NN Apples and Oranges Example

11. Confusion Matrix and Training Error

A machine learning concept called a confusion matrix was used to assess the performance of different machine learning algorithms. [29] [64] For a binomial state classifier, like what is used in this work, a confusion matrix summarizes the number of times the classifier can predict upset when upset data is presented and null-upset data when null-upset data is presented – see Table 8. These two results are referred to as the true-true and false-false prediction outcomes of a confusion matrix. Two other states, which are the false-true and true-false outcomes, are also tracked. In total, these four values make up the confusion matrix and detail how well a classifier can perform the task asked of it. The true-true and false-false values are averaged to give an overall metric for how "accurate" a classifier's predictions are. In this paper, when the accuracy of a classifier is stated, this average value is used.

Table 8: Confusion Matrix for Upset

Prediction / Actual	**Upset**	**Null Upset**
Upset	*True-True*	*True-False*
Null-Upset	*False-True*	*False-False*

For example, the values in Table 9:

Table 9: Example of Confusion Matrix for Upset

Prediction / Actual	**Upset**	**Null Upset**
Upset	*92.2%*	*7.8%*

Null-Upset	*15.4%*	*84.6%*

The value reported as the accuracy for the fake classifier in Table 9 would be the average of 92.2% and 84.6%, which is 88.4%. These two numbers are the numbers that represent the values that were correctly classified during validation. The other two numbers, 15.4%, and 7.8%, would be described as the training error as they represent the percentage of incorrectly classified values. These metrics exist simply to give the user a measurable means of evaluating the performance of a classifier. Therefore, either can be used based on the researcher's preference.

Chapter 5: Traditional and Existing Upset Trends

This chapter focuses on identifying the extent to which upset trends can be observed using traditional means. The upset trends of the core four features – frequency, pulse width, injection power, and injection time – are presented and described for the four microcontroller devices described in Chapter 3.

1. Data and Approach

Examples of where upset does and does not occur with respect to a change in one or more IEMI waveform characteristic is required to perform a trends analysis. The experimental set-up provided the means to change the IEMI waveform in four ways: 1) frequency, 2) input power, 3) pulse width, & 4) injection time. A list of the parameters and their trade space can be found in Chapter 3's Testing Parameter section and is repeated below - see Table 6 or Table 10. A discussion of why these parameters were

selected can be found in Chapter 3. A parametric sweep of these parameters was performed, and then each permutation was assessed to determine whether upset was present or not. Each parameter combination was tested once,resulting in 30240 samples for each MCU device.

Table 10: Chosen Testing Parameters

Variable	**Discrete Samples**	**Number of Sequences**
Frequency (MHz)	20, 50, 100, 200, 400, 800,1000	7
Pulse Width (ns, FWHM)	25, 50, 100, 200, 400, 800	6
Input Power (dB, 20W CW)	-27, -24, -21, -18, -15, -12, -9, -6, -3	9
Injection Time (us)	6 to 8 us with even spacing	80

The following graphs depict the number of upsets observed when a single parameter – frequency, pulse width, input power, or injection time – is fixed at a specific value. Consequently, the Probability of Upset (PoU) is used to show a normalized representation of the data for comparison between the graphs – these line graphs are referred to as the aggregate trend for a given parameter. In any given graph, such as Figure 28, the number of samples, N, is the same for each data point, but N is different for each graph - see the comparison to Figure 29 - since each of the parameters has a different number of discrete values.

In the graphs that break out the trends by frequency, the black line with the circle markers is the aggregate trend used to compare the MCU devices. It is important to understand the distinction between the two trend types because the comparison plots

between MCU devices are of the aggregate trends and not of the frequency trends. However, the frequency trends must be considered to understand whether the aggregate trend is consistent with frequency. Only the frequency trends for MCU1 are broken out and explicitly assessed, as the comparison between aggregate trends is the focus and used as a similarity metric.

2. Frequency vs. Upset Trends

Figure 28 suggests all four MCU devices have a similar upset response since all are most susceptible to upset between 20 MHz and 100 MHz. Unfortunately, this trend is partially misleading because the power injected is not the same for all frequencies. The input power is set on the RF signal generator as a dB value relative to a 1mW output, where all frequencies have the same amplitude before they enter the amplifier and downstream components. The downstream components such as cables, the RF switch, filters, and connectors subtly adjust the signal's amplitude based on its frequency content. However, the most prominent change comes from the RF amplifier, as it has a changing gain profile for each frequency. This means that even if the input power is held constant, a different output power will result based on the frequency. To understand the output, Figure 27 was compiled to show the change in output power by frequency for a given input power.

In Figure 27, it is important to understand that from -33 dB to -15 dB, the measured power is highly similar for all frequencies. However, beyond -15 dB, the measured power can vary significantly between the frequencies. For example, at the maximum input power -3 dB, the 50 MHz frequency has a value of about 13 W, while the 1000 MHz

frequency at the same input power level only has a value of around 5 W. Therefore, the trend seen in Figure 28, needs to be looked at in tandem with the power output graph, Figure 27, and the power upset trend (as a function of frequency) seen in Figure 29. Table 11 presents the values seen in Figure 27 for immediate comparison. The stand-out aspect to note from Figure 28 is that for MCU 1 and 2, the peak Probability of Upset value occurs at 50 MHz, while for the other two devices, the peak occurs at both 50 MHz and 100 MHz. For all devices, there is a sharp decline in PoU beyond 100 MHz. The input power trends also need to be reviewed to understand this frequency trend best.

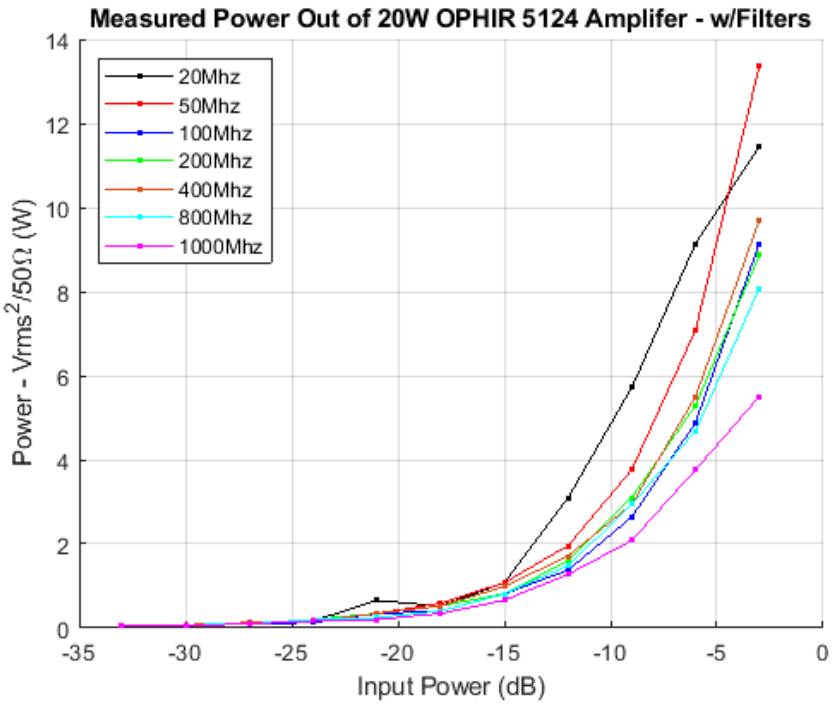

Figure 27: Measured Amplifier Power Out with Filters Present

Table 11: Amplifier Output (W) by Frequency and Input Power

	Output Power (W)						
Input Power (dB)	20MHz	50 MHz	100 MHz	200 MHz	400 MHz	800 MHz	1000 MHz
-33	0.05	0.05	0.05	0.05	0.07	0.05	0.05

	Output Power (W)						
Input Power (dB)	20MHz	50 MHz	100 MHz	200 MHz	400 MHz	800 MHz	1000 MHz
-30	0.07	0.07	0.05	0.07	0.07	0.07	0.05
-27	0.13	0.09	0.09	0.09	0.13	0.09	0.09
-24	0.16	0.20	0.13	0.20	0.16	0.20	0.16
-21	0.65	0.34	0.34	0.34	0.34	0.24	0.20
-18	0.52	0.58	0.39	0.52	0.52	0.40	0.34
-15	1.07	1.07	0.81	0.81	0.98	0.81	0.65
-12	3.09	1.95	1.37	1.59	1.71	1.48	1.27
-9	5.71	3.76	2.63	3.09	2.94	2.94	2.08
-6	9.14	7.08	4.88	5.29	5.50	4.69	3.76
-3	11.45	13.35	9.14	8.86	9.69	8.08	5.50

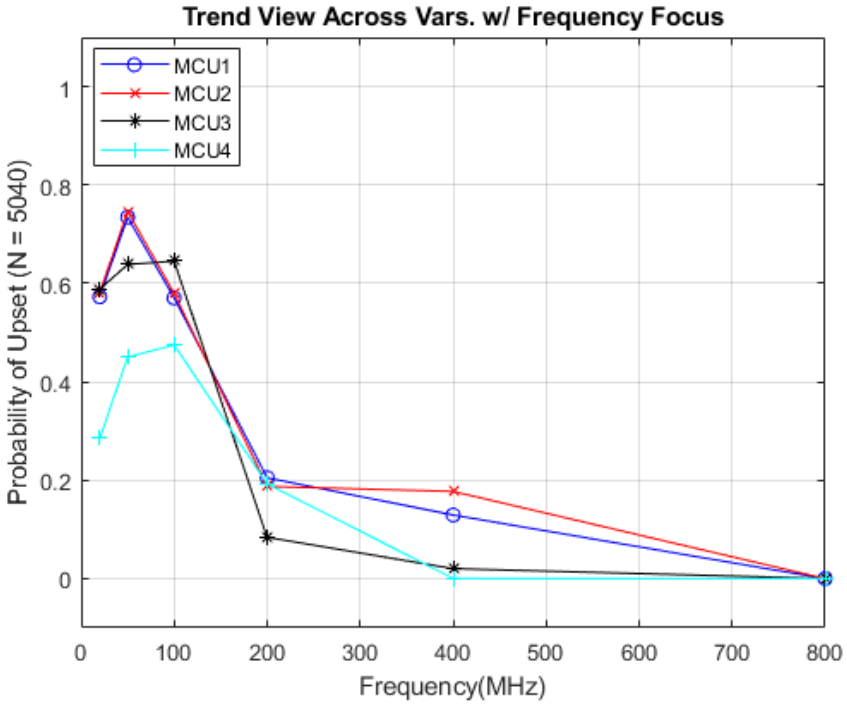

Figure 28: MCU1-4 Probability of Upset vs. Frequency Trend Comparison

3. Power vs. Upset Trends

Figure 29 shows that the power upset trend is different based on the frequency of the IEMI, which is further confirmed by Figure 30, because the power into the device is nearly identical from -35 dB to -15 dB, and similar enough from -15 dB to -3 dB. An important detail to observe is that the 200 MHz and 400 MHz frequencies have equally high PoU at powers above -9 dB but are effectively zero at powers below -9 dB. This suggests that the frequency trends seen in Figure 28 are genuine, in an aggregate sense, but do not tell the whole story. There are a handful of aspects to note from Figure 29. 1) The trend for all frequencies is not the same. 2) A power threshold is suggested by the fact that all frequencies show a high PoU when above 8 W, while the 200 MHz and 400 MHz frequencies show little to no PoU when below this value. 3) From a practical (and theoretical) standpoint, the trends are expected to approach a PoU value of 1 asymptotically. This means that as the power increases towards the maximum (20 W), a near 1.0 Probability of Upset is expected to be achieved. 4) The trends for 100 MHz and 200 MHz do not match even though they have nearly identical output powers, as shown in Figure 27. Instead, they have opposing convex vs. concave trend shapes. RF coupling may explain the shift in power vs. upset as the frequencies between 20 MHz and 100 MHz steadily grow towards the maximum Probability of Upset, while other frequencies do not. However, RF coupling cannot fully explain the change in shape seen most prominently with the 100 MHz and 200 MHz trends. Jumping ahead slightly, the data and analysis of the pulse width and injection time trends do not provide further understanding of why there is a change in power trend shape for these frequencies. Instead, the data suggests that their trends are more (or less) prominent based on whether the frequency is

presumed to couple well (or not couple well). Therefore, further investigation into this matter is recommended. Collection of IEMI upset data with a greater resolution with respect to frequency may reveal a transition node between these trend shapes. In general, the data suggests that the output power may not couple to – get into – the circuit and therefore would produce no upset. Regardless, the data collected here is insufficient to explain the frequency-based upset trends fully. However, the aggregate trends can still be used to compare the four devices and determine their similarities.

Figure 30 shows the aggregate upset vs. power trends for the 4 MCUs. The trends suggest that an increase in power results in an increase in PoU. MCU1-3 all have nearly identical upset responses suggesting that the upset to power trend does not largely change within an architecture. What is surprising is that MCU4 does not have a significantly different response from MCU1-3 but is more resilient to upset. At powers above -15 dB, MCU4's PoU hits a sort of asymptote around about 0.35 while MCU1-2 continues to climb, and MCU3 flattens out around 0.50. From a practical standpoint, it is reasonable to expect that a significant increase in power would overcome these asymptotes. At CW powers beyond 20 Watts, physical (or thermally induced) damage occurs. This type of damage is extensively reported in a white paper from the Industry Council on ESD Target Levels titled "White Paper 4 - Understanding Electrical Overstress – EOS". [65] When physical damage occurs on a Microcontroller, the PoU will almost always be 1.0 because nearly every electronic device does not have sufficient redundancy to overcome such phenomena.

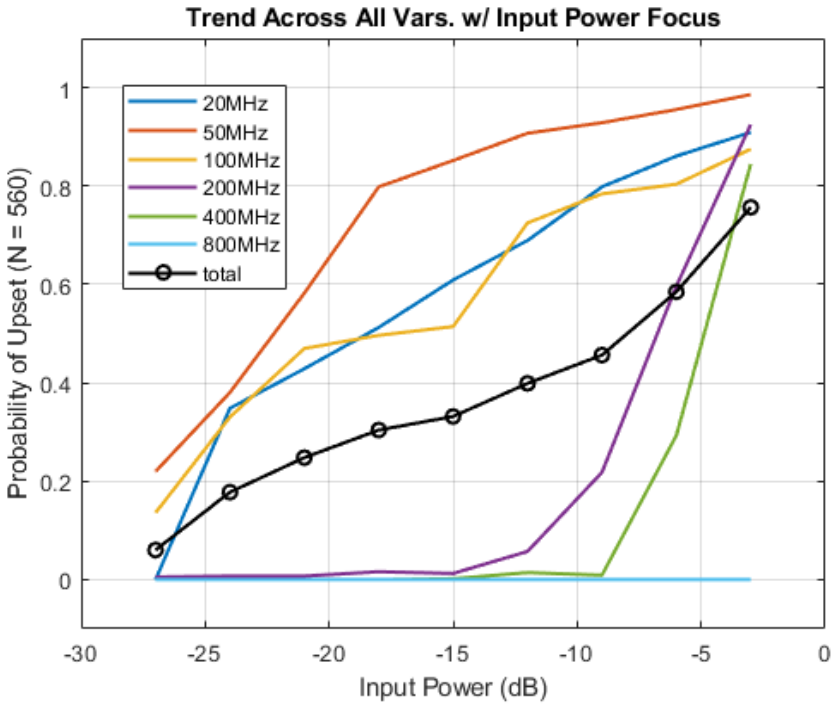

Figure 29: MCU1 Probability of Upset vs. Input Power Trend by Frequency

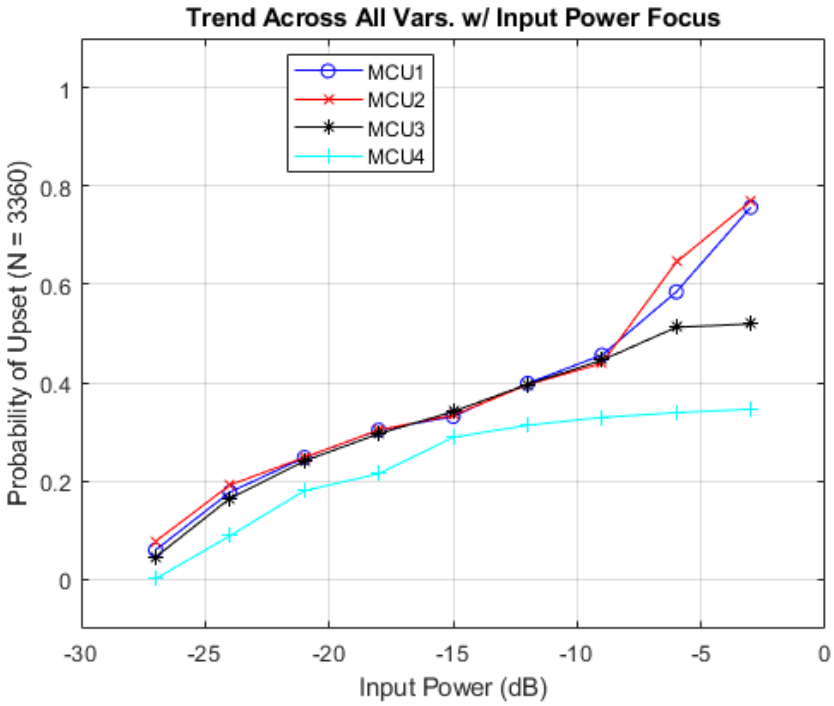

Figure 30: MCU1-4 Probability of Upset vs. Input Power Trend Comparison

4. Pulse Width vs. Upset Trends

Figure 31 shows the pulse width trends by frequency for MCU1. The trends resemble a logarithmic curve shape that flattens as the pulse width increases – except the 800 MHz frequency. This frequency does not show any response, which makes sense because the previously discussed upset trends for frequency and input power suggest that upset is not occurring at 800 MHz. An interesting observation here is that nearly all frequencies have a rapid increase in probability before leveling out asymptotically. From a theoretical standpoint, this suggests that the good coupling frequencies that occur between 20 MHz and 100 MHz have the propensity to achieve a near 1.0 Probability of Upset as the pulse width increases. Alternatively, the frequencies that couple poorly are not expected to achieve a high PoU because the trends show them leveling out at or below 0.5. From a mathematical optic, the area under each curve appears to be proportional to the upset probability. Here, the 20 MHz to 100 MHz have the most pronounced logarithmic curve (with the largest area under the curve), while the other frequencies are more damped (and have less area under the curve). This analysis does not make an effort to tie the actual area under a given curve to upset phenomena. Instead, it is used to describe the difference between the trend curves qualitatively. The data also suggests a quasi-threshold around 100 ns and 400 ns. The 50 MHz, 200 MHz, and 400 MHz curves show a significant diminishing return towards PoU at pulse widths beyond 100 ns. However, the 20 MHz and 100 MHz trends do not achieve this similar level of diminishing return until around 400 MHz.

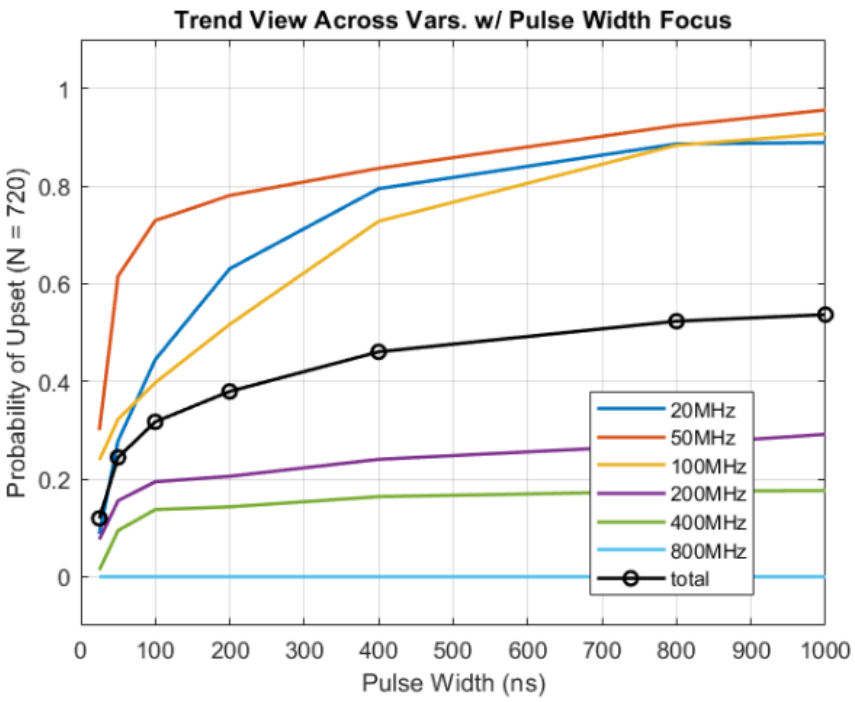

Figure 31: MCU1 Probability of Upset vs. Pulse Width Trend by Frequency

Figure 32 shows the aggregate pulse width vs. upset trend response for MCU1-4. As suggested in the previous variables' trends, MCU1-3 have a very close agreement while MCU4 is not far off. Furthermore, all of the aggregate trends asymptotically approach the 0.6 PoU point. A key takeaway from this analysis is that although the aggregate trends approach a maximum PoU of 0.50, the reality of the matter is that three of the six frequencies contributing towards that result achieve a maximum PoU of 0.3 or lower, while the other three frequencies achieve a maximum of 0.8 or higher. This significant difference in maximum PoU results in the aggregate being averaged down to around 0.5.

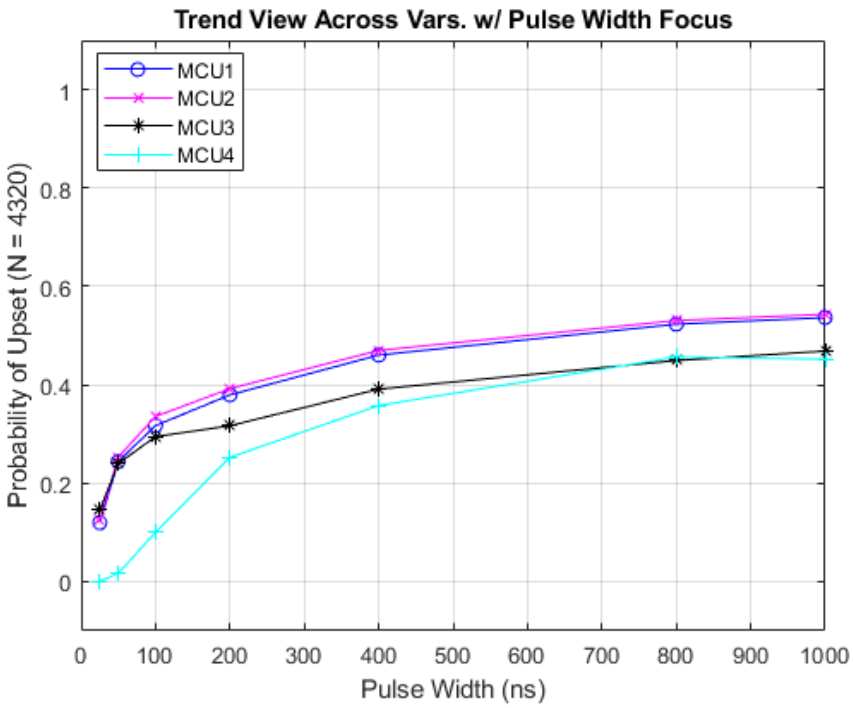

Figure 32: MCU1-4 Probability of Upset vs. Pulse Width Comparison

5. Injection Time vs. Upset Trends

Figure 33 shows the upset trends while changing the injection time for each frequency on MCU1. Moreover, the damping of the response can be observed based on how well the RF couples to the circuit – for instance, at 50 and 100 MHz, the trend is most pronounced because the IEMI event couples well at those frequencies.

Figure 34 shows the aggregate injection time vs. upset trend comparison between the four devices. In general, MCU1 through MCU3 agree and show upset being most likely during clock rise and fall edges. MCU4 does not conform to the MCU1-3 trend. Instead, upset happens more often during the clock high state for MCU4. This makes sense given that MCU4 is a different architecture and latches data differently. Specifically, MCU4 executes instructions in parallel and gives each an extra cycle of waiting time to finish,

compared to the MCS-51 architecture, which performs instructions sequentially and as rapidly as possible.

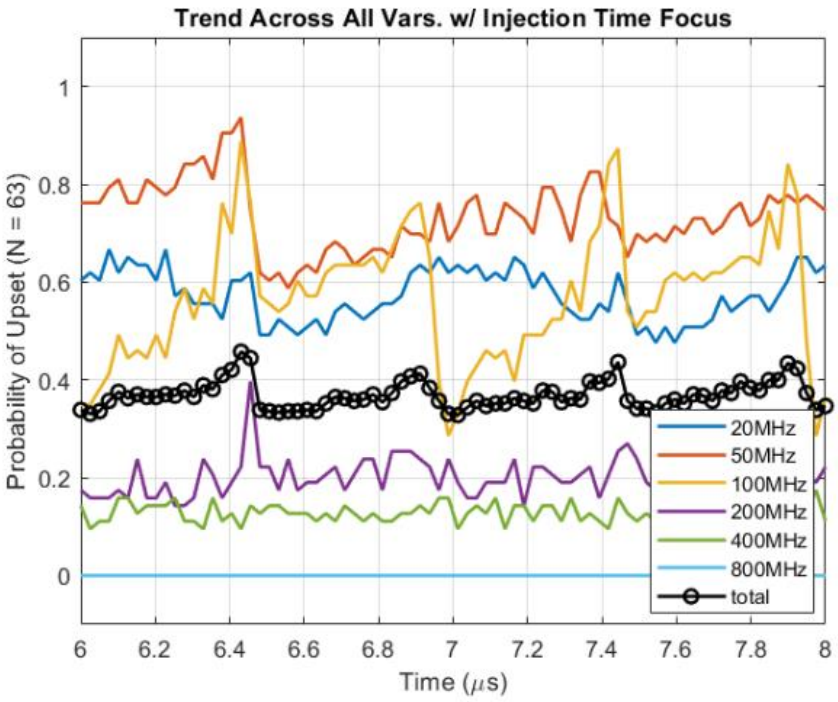

Figure 33: MCU1 Probability of Upset vs. Injection Time Trend by Frequency

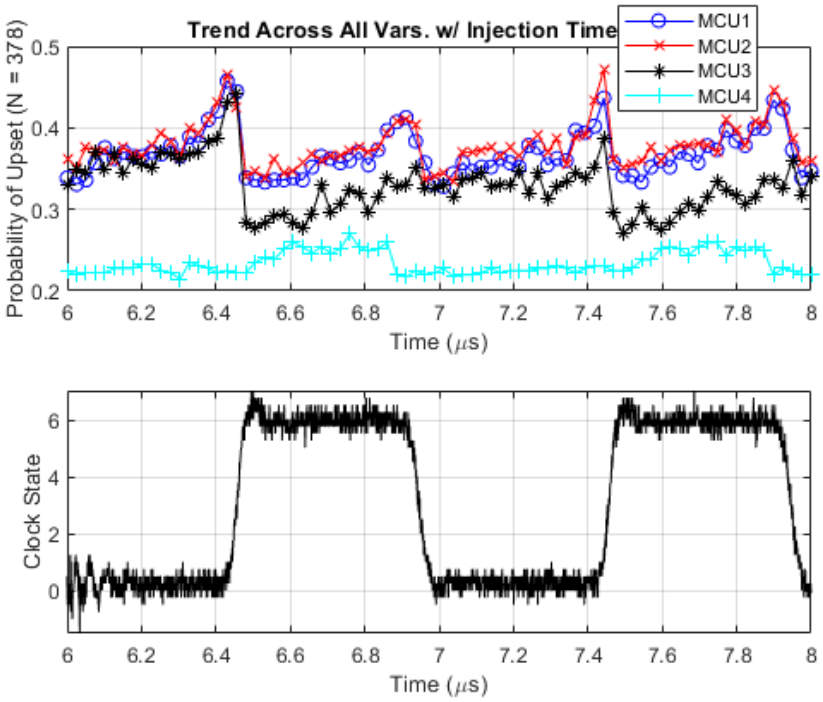

Figure 34: MCU1-4 Probability of Upset vs. Injection Time Trend Comparison

Chapter 6: Survey and Selection of Machine Learning Methods

This chapter focuses on identifying the appropriate Machine Learning method to be used in Chapters 7 and 8. Three Machine Learning Algorithms are compared: k-Nearest Neighbors, Support Vector Machines, and Decision Trees. This research aims to broaden the scope of understanding into how the methods perform when applied to empirically collected IEMI Effects data on Microcontrollers. Furthermore, it will determine which method is best suited for the subsequent experiments.

1. Experiment Description

This experiment is focused on using upset data collected with the SALVO apparatus to assess the performance of the k-Nearest Neighbors and Decision Trees classifiers while using Support Vector Machines as a control since that method has been previously shown to work.

Using MATLAB's Classification Learning toolbox, the upset data was used to train 14 different classifiers: three Decision Trees, six k-Nearest Neighbors, and four Support Vector Machines. The Decision Tree and k-Nearest Neighbors were selected as they are classified as fast-to-train methods. Support Vector Machines are not classified as fast-to-train.

Although this research focuses on Decision Trees and k-NN classifiers, Support Vector Machines classifiers were trained as a litmus test with reach back to Bilalic's 2017 dissertation work, which showed that Support Vector Machines could produce accuracies greater than 90%. Since the dataset used in this research is not the same as the dataset

used by Bilalic, it is crucial to get a baseline of whether a 90% classifier - using one of his chosen methods Support Vector Machines, Gaussian Processes, or Artificial Neural Networks – is possible so that there is continuity between the two sets of work. If the datasets were sufficiently different, then any new claims made about the application of machine learning methods could not be extrapolated to the existing body of work.

Multiple variants of each classifier method were used to assess generic and specific performance trends. Their differences come from the criteria used to set up each classifier. Table 12, Table 13, & Table 14 provide a named list of the specific classifier variants as well as a basic overview of what the significant differences are between the methods. A specific example would be in Table 12, where the difference between the three Decision Trees used is the number of branches – 100, 20, 4.

The main takeaways from each table are that the variants are defined by how the methods are set up with regard to the number of branches or neighbors, and then finally with which mathematical construct – kernel - is implemented as the backbone of the method.

In practice, there are no hard and fast rules for determining whether one method, kernel or setup parameters are better than another. Instead, it is advised that a few should be tried, and the user should select the best performer – assuming it is not an overfit solution.

Good machine learning toolkits will provide the ability to change the kernels for the given algorithm allowing the user flexibility to try multiple variants.

All classifiers were validated using a 25% hold-out method on the dataset – meaning that 25% of the data was not used in training, ensuring that it could be used to evaluate the

classifier's performance. Since this hold-out method inherently selects 75% of a random group of data, the training dataset is technically different every time, which means that there could be variation in performance based on what data points are used. The Training Performance Repeatability section provides a small study into this variation for the three tested algorithms.

Each of the 14 classifier variants were trained with the same dataset – MCU1 – containing ten feature predictors and using the upset feature as the response. The ten features used are: Frequency, Pulse Width, Inject Power, Inject Time, Measured Power, Min Voltage, Max Voltage, RMS Voltage, Peak to Peak Voltage, and Energy (Emax). The first four are user-controllable variables, while the other six are collected data or engineered features. For example, Energy (Emax) is an engineered feature of Pulse Width multiplied by Measured Power.

The following tables and graphs show classifier accuracy performance data in percentage. This means that every table value, or data point in a plot, is a trained classifier's end resulting performance.

Table 12: Decision Tree Variants

Classifier Name	**Branches**
Fine Tree	100
Medium Tree	20
Coarse Tree	4

Table 13: k-NN Variants

Classifier Name	**Neighbors**	**Dist. Metric**	**Dist. Weight**
Fine k-NN	1	Euclidean	Equal
Medium k-NN	10		
Coarse k-NN	100		
Cosine k-NN	10	Cosine	
Cubic k-NN	10	Minkowski	
Weighted k-NN	10	Euclidean	Square Inverse

Table 14: SVM Variants

Classifier Name	**Kernel Function**	**Kernel Scale**
Medium Gaussian SVM	Gaussian	3.2
Fine Gaussian SVM	Gaussian	0.79
Cubic SVM	Cubic	Auto

2. Comparison of Method Performance

Table 15 summarizes the performance accuracy results of the classifiers using all features and upset as the response. All trained classifiers performed well, achieving 88.2% or better accuracy. On average, k-NN had an average accuracy of 92.68%,

Decision Trees had an average accuracy of 91.16%, and SVM had an average accuracy of 90.75%. The standard deviation between the performance of DT, k-NN, and SVM is 1.4%, 1.5%, & 1.6%.

The Weighted k-Nearest Neighbors algorithm was the best performer with a 94.2% accuracy, while the worst performer was the Coarse Gaussian SVM with 88.2% accuracy.

Table 15: Classifier Accuracy Comparison

Name	**Accuracy (%)**
Fine Tree	92.9
Medium Tree	91.4
Coarse Tree	89.2
Fine k-NN	93.6
Medium k-NN	93.1
Coarse k-NN	89.8
Cosine k-NN	92.7
Cubic k-NN	92.7
Weighted** k-**NN	**94.2**
Medium Gaussian SVM	91.1
Fine Gaussian SVM	92.7

Cubic SVM	91.0
Coarse Gaussian SVM	88.2

3. Comparison of Method Time-to-Train

Moreover, the *k*-NN algorithms also had the second-best training time. Table 16 summarizes the training time for each of the methods. The Fine *k*-NN was the fastest to train - taking only 1.55 seconds, while the Cubic SVM was the slowest to train at 702.46 seconds. Excluding the Cubic SVM, the mean time-to-train for each method shows that the Decision Trees and *k*-NN are closely matched with times of 3.46 and 3.45 seconds, while the SVM is at 18.23 seconds - which is more than five times longer than the DT and *k*-NN times. For bulk data collection a five-fold difference is noteworthy.

Table 16: Classifier Training Time

Name	**Time to Train (s)**	**Mean (s)**
Fine Tree	3.8559	3.4654
Medium Tree	3.4049	
Coarse Tree	3.1354	
***Fine** k-NN*	**1.5573**	**3.4545**
Medium k-NN	1.5639	
Coarse k-NN	2.4705	

Cosine k-NN	8.9286	
Cubic k-NN	4.3741	
Weighted k-NN	1.8330	
Medium Gaussian SVM	12.905	189.29 (all) || 18.233 (no cubic)
Fine Gaussian SVM	29.899	
Cubic SVM	702.46	
Coarse Gaussian SVM	11.896	

4. Training Performance Repeatability

Early on, it was concluded that every time the classifiers were trained, the training accuracy changed in a non-negligible amount. Consequently, a quick repeatability study was performed to assess the impact of the 25% hold-out method on classifier accuracy by training each classifier ten times and recording the result. Select variants from each algorithm were selected for the study to get a sense of variation. The results can be found in Table 17 and suggest that repeatability is good, with a standard deviation of between 0.2 and 0.4% depending on the method. Therefore, it was decided that the classifier accuracy would be bounded by the worst-case observed - ±0.3% - to denote that the performance value may differ by this amount due to how the training data is selected.

Table 17: Repeatability Performance

Name	**Performance %**	**Mean**	**STD**

Coarse Tree	89.2, 89.7, 89.7, 89.4, 89.8, 89.7, 89.7, 89.3, 88.9, 90.1	89.6	0.3
Fine k-NN	93.6, 94.0, 93.9, 93.6, 93.7, 94.0, 93.8, 93.4, 93.3, 93.8	93.7	0.2
Coarse Gaussian SVM	88.2, 88.4, 87.9, 88.4, 88.2, 88.5, 88.5, 88.1, 88.2, 88.7	88.3	0.2
***Weighted** k-NN*	**94.2, 94.4, 94.6, 94.2, 94.1, 94.2, 94.0, 94.3, 94.5, 94.7**	**94.3**	**0.2**

5. Weighted *k*-NN Variations

Since the Weighted *k*-NN has been established as the best overall performer for this experiment, it is essential to explore its performance knobs further. There are three knobs to vary the: Number of Neighbors, Distance Metric, & Distance Weight.

Identifying the highest accuracy and most stable combination of these variables is important for a few reasons: 1) it suggests the suitability of a classifier regardless of if the dataset is flush or sparse, 2) it specifies the consistency to which predictions can be made. If changing the number of neighbors from say five to ten resulted in a 20% difference in accuracy, the classifier would be considered extremely sensitive and not well matched to reasonable variations in the dataset, and 3) it suggests the extent to which the data can be separated. Data which can only be separated by a narrow margin will inherently be less stable or subject to wild swings in classification accuracy.

There are three Distance Weights and nine Distance Metrics to choose from in the MATLAB toolbox. Figure 35 presents the variation in performance as the Number of Neighbors is changed for each of the three Distance Weights – Square Inverse, Equal, and Inverse – when the Distance Metric used is Euclidean. The Inverse Distance Metric was included – for completeness - in this analysis as it is an available technique that was

not previously considered in the down-selection process. Figure 35 suggests that Square Inverse is the most stable and provides the best prediction accuracy, which means that as the data size changes, the predictions will continue to provide solid performance.

Figure 36 presents the variation in performance as the Number of Neighbors is changed for each of the different Distance Metrics when the Distance Weight is set to Square Inverse. It is a surprise that the performance of the Euclidean Distance Metric noticeably rolls off when the number of neighbors is increased because this distance metric is often the default – or often most used – paired with k-NN [9]. It should be noted that the Euclidean Distance performance only decreases by 3% when spanning four orders of magnitude, while the Minkowski changes by 7%, so the performance loss is not extreme.

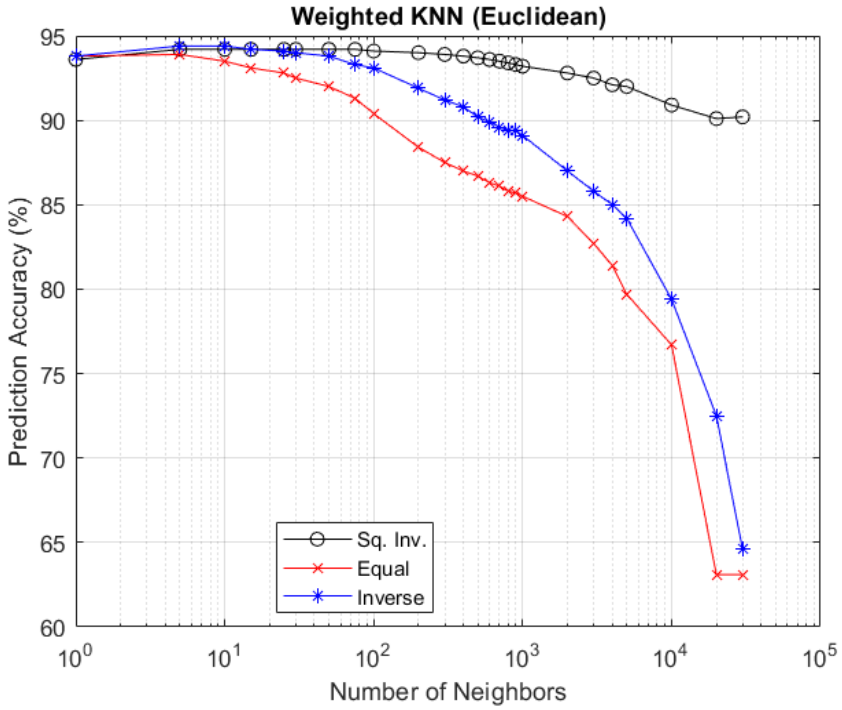

Figure 35: Weighted k-NN Performance vs. Number of Neighbors and Distance Weight

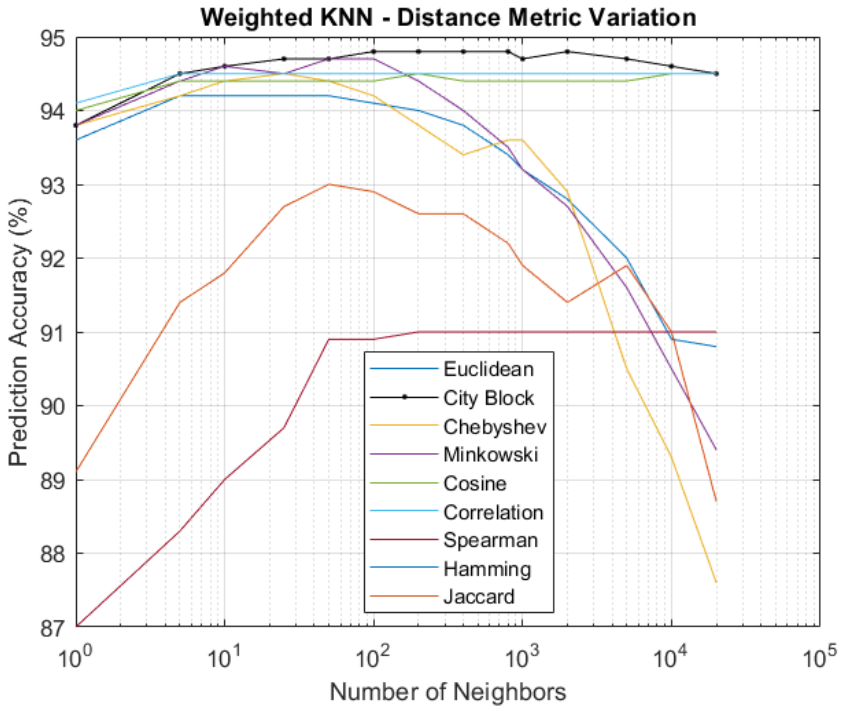

Figure 36: Weighted k-NN Performance vs. Number of Neighbors and Distance Metric

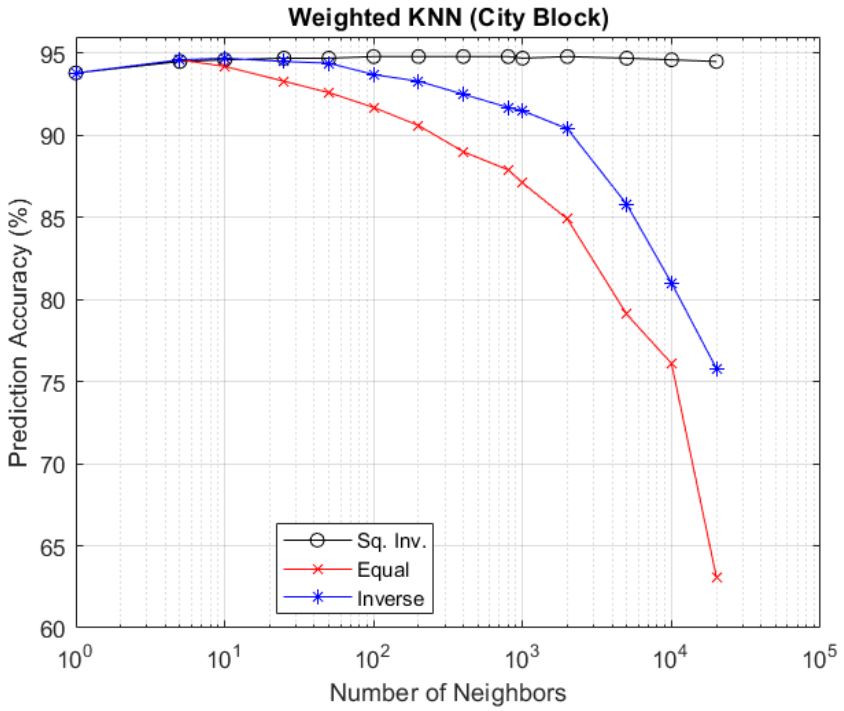

Figure 37: City Block Weighted k-NN Performance vs. Number of Neighbors and Distance Weight

The Euclidean Distance Metric is often used in data clumping techniques because it easily can compare data through a simple measurement of how far – relatively – two data points are from one another. Short distances suggest similar data, while long distances suggest dissimilar data.

As a function of the Number of Neighbors, stability is an important metric to consider because it speaks to the consistency of performance based on the number of samples available in training. For example, the Spearman Distance Metric would not be desirable because the best performance occurs when using a k value of 50 or greater. In datasets of less than 100 samples, performance is inferior – compared to the Correlation Distance Metric - simply because there is insufficient data.

The best performer is the City Block Distance Metric, as it has the overall highest and most stable performance when the number of neighbors is varied. It should be noted that the Correlation Distance Metric is definitively the most stable but has slightly lower performance, so the leading edge goes to the City Block Distance Metric – see Figure 36.

The City Block distance method has the following mathematical form:

$$D = \sum_{j=1}^{k} |a_j + b_j|$$

Where ‘D’ is the distance, k is the number of dimensions, and ‘a’ and ‘b’ are two points. The concept of the city block distance is derived from the idea that the shortest distance between two points is the hypotenuse (this is the Euclidean distance), but the sum of the other two sides of the triangle would make up city block distance. There is no apparent reason why this method is more stable than other options.

Figure 37 summarizes the analysis by verifying that the Square Inverse is the best Distance Weight for the City Block Distance Metric. Again, the Square Inverse Weight is determined to be the most stable and highest performing option compared with the other two options.

Chapter 7: Prediction Accuracy by Selection of Features

This chapter focuses on determining the extent to which high accuracy (75%+) classifiers can be trained when using as few features as possible. The 75% threshold is set relative to the scale of 50 to 100%. Accuracies of 50% or lower are not valuable because they are akin to a guess or a coin flip. However, accuracies of greater than 75% are considered beneficial because they suggest the prediction is based on information that can be mathematically tied to the correct label. Moreover, this chapter investigates the impact each data feature has on a classifier.

The Weighted k-Nearest Neighbors (k-NN) classifier was trained using each combination of the ten features to formally evaluate the predictive contribution of the features and determine the optimal training set. This approach resulted in 1023 classifiers, whose performance could be compared.

1. Experiment Description

A primary theme of machine learning is to train with the most data possible without causing the resulting classifier to become overfit. The problem of data quantity (and quality) can often be a challenge as certain experiments do not enable the collection of many samples. Upset data is often scarce. Therefore, it is attractive to evaluate the quality of predictions made using limited data. Fortunately, this effort does not have limited data, and therefore a methodical – exhaustive - approach can be taken to understand the scope of prediction accuracy by adding and removing feature information – this is referred to as feature starvation or more generally as a different version of data sparsing.

This experiment focused on training a weighted k-Nearest Neighbors classifier with different combinations of features to carefully sparse the data. The Combination Formula defined all possible combinations of a ten-feature dataset. The resulting combination list was used to define the feature inputs for each classifier while holding the number of neighbors (k = 10), distance weight (square inverse), and distance metric (city block) constant. For example, using a combination c = 10 value would produce one classifier trained with all ten features; Alternatively, a combination c = 1 value would produce ten classifiers, each trained using only one feature.

A primary goal of this work was to determine whether the ten features have the same predictive power individually and whether specific combinations produce classifiers with greater than 75% prediction accuracy.

2. Repeatability

A limited repeatability study was performed in Chapter 6, which concluded that prediction accuracy could vary up to ±0.3% based on how the data was selected. In early experimentation, it was noted that the top-10 performing classifiers only varied by 0.18% from 1st to 10th place yet used a wide variety of feature inputs. No clear trend could be identified. Moreover, the variation was well within the previously identified performance variation. Table 18 presents the top-10 classifiers for interested readers.

Table 18: Top-10 Highest Accuracy Classifiers

Rank	Features	Features Used	Accuracy (%)

1	*7*	*Freq., Input Power, Inject Time, Pulse Width, V_{rms}, V_{p2p}, V_{max}*	***94.92***
2	*8*	*Freq., Input Power, Inject Time, Pulse Width, V_{rms}, V_{p2p}, V_{min}, E_{max}*	*94.87*
3	*7*	*Freq., Input Power, Inject Time, Pulse Width, V_{rms}, E_{max}, V_{min}*	*94.85*
4	*8*	*Freq., Input Power, Inject Time, Pulse Width, Meas. Power, V_{rms}, V_{p2p}, E_{max}*	*94.85*
5	*6*	*Freq., Input Power, Inject Time, Pulse Width, Meas. Power, E_{max}*	*94.80*
6	*5*	*Freq., Inject Time, Pulse Width, V_{max}, E_{max}*	*94.79*
7	*6*	*Freq., Inject Time, Pulse Width, $V_{rms,}$ V_{max}, E_{max}*	*94.78*
	8	*Freq., Input Power, Inject Time, Pulse Width, V_{rms}, V_{p2p}, V_{max}, E_{max}*	*94.78*
9	*6*	*Freq., Inject Time, Pulse Width, Meas. Power, V_{max}, E_{max}*	*94.74*
	8	*Freq., Input Power, Inject Time, Pulse Width, Meas. Power, V_{rms}, V_{max}, E_{max}*	*94.74*

It was concluded that a single iteration of training was insufficient to provide a qualitative assessment of classifier performance – and therefore also insufficient to rank feature significance. Consequently, the baseline classifier containing all ten features was trained 1000 times to determine the scope of performance and point at which the mean

and median results converged. The purpose of training multiple times was because the classifiers were trained with data selected using the hold-off method. The hold-out method randomly selects a percentage (75% for this implementation) of data to use in training the algorithm but leaves the remaining data (25%) to be used for validating the model's prediction accuracy [62].

Figure 38 provides the output of this 1000 iteration training study. The main takeaways are: the maximum accuracy achieved was 95.19%, the minimum accuracy achieved was 93.66%, and the mean and standard deviation accuracy are 94.43% and ±0.23%, respectively. Although the standard deviation is only ±0.23%, the delta between the minimum and maximum accuracy classifiers is 1.53%, suggesting that training data selection can alter the prediction accruacy significantly.

There are 30,240 data points available to select from in training, which provides an immense number of different combinations when selecting 75% of the points. It is therefore unreasonable to thoroughly test all possible training outcomes for this dataset – however, the silver lining is that with so much variation possible in training, overfitting is highly unlikely because data spans a large enough space that not all of it can be considered and force an undesirable bias.

A practical solution to determining the actual performance of a trained classifier is to collect multiple training outcomes and use the median as the "true" performance result. However, in the case mentioned above, 1000 iterations are more than necessary since the mean and median outcomes first converge – to within two decimal places – at around 125 iterations; Therefore, there is no need to go beyond that number of iterations.

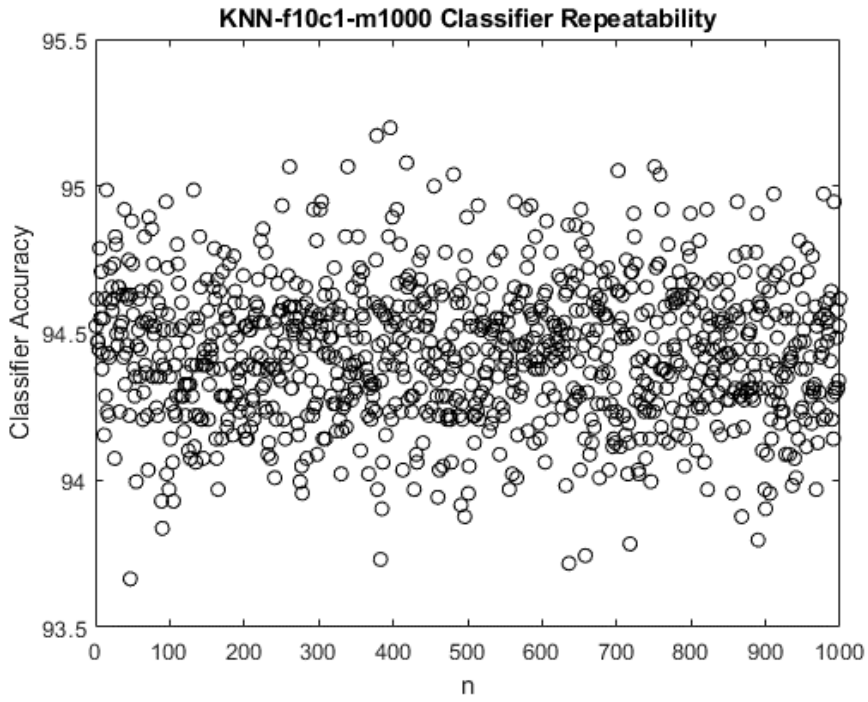

Figure 38: Weighted k-NN Performance over 1000 iterations (n)

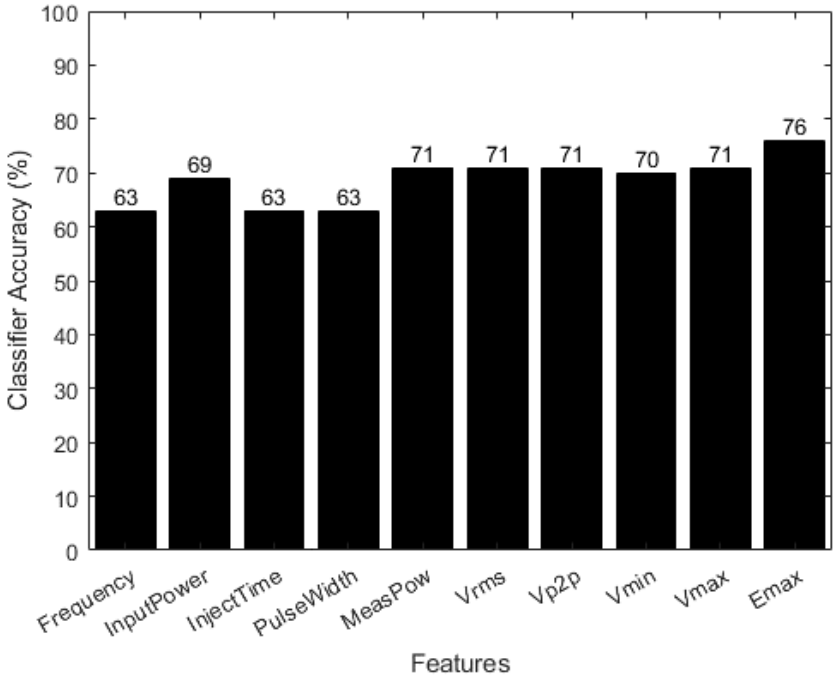

Figure 39: Weighted k-NN Performance vs. Single Feature Used to Train

The following tables and figures show classifier accuracy data in percentage. The percentage value is established from the confusion matrix's true-true and false-false prediction results from the validation step of building a classifier.

Each classifier accuracy presented is the median value resulting from 125 iterations of that classifier being trained on a different random 75% data selection and then evaluated. This value is considered the "truth" value since both the mean and median results have converged.

3. Single Feature Trained Classifier Accuracy Comparison

Each feature was used singly to train a classifier. The purpose of this was to determine if the features have equal prediction ability. Figure 39 shows the rounded prediction accuracy as a function of each feature, while Table 19 presents an ordered list of the features and their accuracy. Input Power had the highest predictive capability of the core features, resulting in a classifier with 68.82% accuracy, while the other three core features had a 63.13% accuracy. The best overall prediction accuracy of the ten features is E_{max} with 76.38%, while the other engineered features performed with 69-71% accuracy. Measured Power placed third overall with a 71.19% accuracy.

Table 19: Rankings for Single Feature Classifiers

Rank	**Feature**	**Accuracy (%)**
1	E_{max}	*76.38*
2	V_{p2p}	*71.28*

3	*Measured Power*	*71.19*
4	V_{max}	*71.15*
	V_{rms}	*71.15*
6	V_{min}	*69.55*
7	*Input Power*	*68.82*
8	*Pulse Width*	*63.13*
	Inject Time	*63.13*
	Frequency	*63.13*

4. Multi-Feature Trained Classifier Accuracy Comparison

The 1023 mean/median classifier results were binned by the number of features used in training. Figure 40 presents an overview of all 1023 classifier accuracies plotted against the number of features. The trend in Figure 40 is that the spread of classifier performance narrows as more features are used and accuracy, generally, improves. For example, in the two-feature classifiers, the minimum and maximum prediction accuracies are 58.13% and 90.61%. In contrast, the nine-feature classifiers have a minimum and maximum prediction accuracy of 91.36% and 94.46%. Figure 40 also shows that certain combinations of features provide results that are best defined as outliers. For example, in the six-feature bin, most data points are between 79.47% and 94.51%, while five classifiers perform at around 71% - these five classifiers are well removed from the rest and are separated by nearly 8% in prediction accuracy – This figure has been summarized into a more digestible form in Table 19.

Table 20 presents the minimum and maximum classifier prediction accuracy by the number of features used in training. The highest classifier accuracy noted was using six features with 94.49% - Frequency, inject time, pulse width, V_{rms}, V_{max}, and E_{max}. The lowest classifier accuracy was 58.13% using two features – Input Power and Inject Time. The minimum accuracy trend suggests that training with more features results in a higher accuracy. However, the maximum accuracy trend suggests that training with two or more features can achieve a 90.60% accuracy prediction or greater. Furthermore, the accuracy bounced between 94.39% and 94.49% when more than four features were used. This suggests a diminishing return on performance for more data and, in some cases, "worse" performance.

A closer look at the best-of-class 90.60% accurate two-feature classifier reveals that the frequency and E_{max} features were used in training. This is not unexpected, given that E_{max} was previously identified as the highest one-feature trained classifier, and the trends analysis performed in [4] suggested that frequency impacts all other feature trends via instrumentation power and device-level coupling. Given the information known, these two features would be expected to result in a high accuracy classifier. However, E_{max} is the multiplied result of the pulse width and measured power, so one would expect that the performance of E_{max} as a single feature should track the two-feature classifier using pulse width and measured power. Surprisingly, this is not the case. The two-feature classifier containing pulse width and measured power results in a higher (79.62%) accuracy, while E_{max} alone provides a *76.38%* accuracy. This suggests that the two features complement each other but that the total energy metric provides a different, more separable trend.

Excluding E_{max}, the next highest two-feature performance was 86.61% using Pulse Width and V_{p2p}.

Table 21 presents the top ten highest accuracy classifiers of the 1023 combinations – i.e., the max results seen in Figure 40. The top ten classifiers vary by an accuracy of only 0.04%, which given the training variation, and established standard deviation, means they all perform about the same. The apparent trends which may explain why these classifiers performed best are 1) the number of features and 2) the features used. Table 22 shows the number of times each feature is present in the top ten. For example, Frequency is present in all ten classifiers, while V_{min} is only present in four. Nine of the top-ten classifiers use six to seven features, and all ten include Frequency, Inject Time, E_{max}, and Pulse Width.

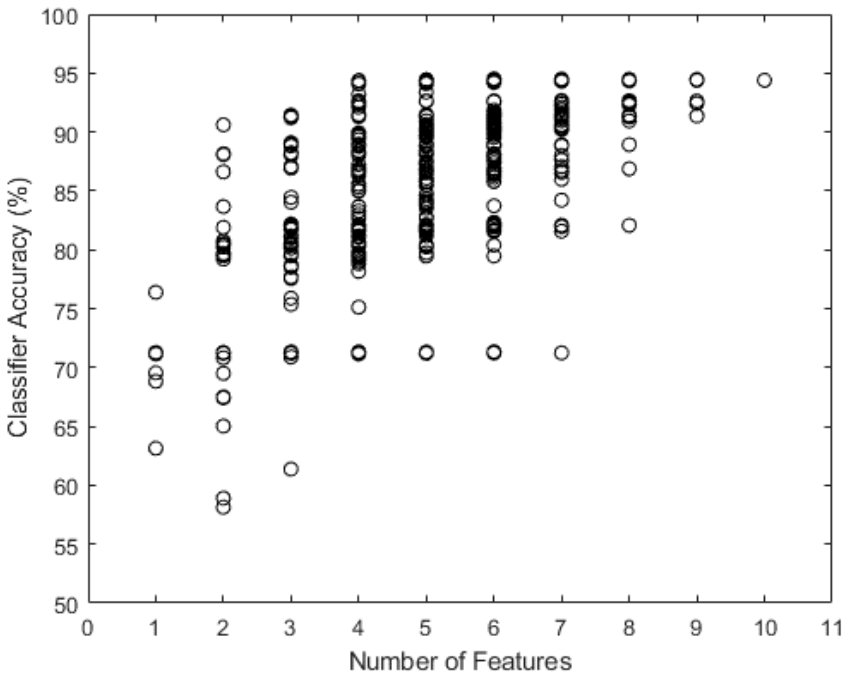

Figure 40: Weighted k-NN Performance vs. Number of Feature Used to Train

Table 20: Prediction Accuracy by Number of Features

Features	**Min Accuracy (%)**	**Max Accuracy (%)**
1	*63.13*	*76.38*
2	*58.13*	*90.60*
3	*61.37*	*91.42*
4	*71.15*	*94.37*
5	*71.20*	*94.44*
6	*71.23*	***94.51***
7	*71.24*	*94.49*
8	*82.06*	*94.47*
9	*91.36*	*94.45*
10	***94.39***	*94.39*

Table 21: Updated Top-10 Highest Accuracy Classifiers

Rank	**Number of Features**	**Features**	**Accuracy (%)**
1	*6*	*Freq., Inject Time, Pulse Width, V_{rms} V_{max}, E_{max}*	*94.51*
2	*6*	*Freq., Inject Time, Pulse Width, Measured Power, V_{min}, E_{max}*	*94.49*
	6	*Freq., Inject Time, Pulse Width, Measured Power, V_{max}, E_{max}*	*94.49*
	7	*Freq., Input Power, Inject Time, Pulse Width, Measured Power, V_{min}, E_{max}*	*94.49*

5	*6*	*Freq., Inject Time, Pulse Width, V_{rms}, V_{p2p}, E_{max}*	*94.48*
	7	*Freq., Inject Time, Pulse Width, Measured Power, V_{rms}, V_{p2p}, E_{max}*	*94.48*
	7	*Freq., Inject Time, Pulse Width, Measured Power, V_{rms}, V_{max}, E_{max}*	*94.48*
8	*6*	*Freq., Inject Time, Pulse Width, Measured Power, V_{rms}, E_{max}*	*94.47*
	7	*Freq., Inject Time, Pulse Width, V_{p2p}, V_{min}, V_{max}, E_{max}*	*94.47*
	8	*Freq., Input Power, Inject Time, Pulse Width, Measured Power, V_{rms}, V_{min}, E_{max}*	*94.47*

Table 22: Features Ranked by Times Present in the Updated Top-10 Highest Accuracy Classifiers

Rank	**Feature**	**Times Used**
1	*Frequency*	*10*
	Pulse Width	*10*
	Inject Time	*10*
	E_{max}	*10*
5	*Measured Power*	*7*
6	*V_{rms}*	*6*
7	*V_{min}*	*4*
	V_{max}	*4*
9	*V_{p2p}*	*3*

10	*Input Power*	*2*

5. Ranked Features by Classifier Performance Threshold

A reasonable question an experimentalist may ask would be, *"…which features should be prioritized in data collection, such that high performing classifiers can be produced?"* Table 22 gives one answer to this question but is only based on the top-ten classifiers and may not fully represent the other high-performing classifiers present. For example, the top-ten classifiers all had above 94% accuracy, but there are an additional 114 classifiers that also have a 94% or above accuracy. All 124 classifiers should be considered when evaluating the priority of features to collect. Since not all classifiers performed at this level, other thresholds ("bins") were defined to capture their performance.

The best performance seen in one-feature trained classifiers was around 76%, while the best accuracy of any classifier was around 94%. To cover the minimum and maximum accuracy bins, thresholds of 75%, 85%, 90%, and 94% were chosen. 885 (of 1023) classifiers have an accuracy of 75% or better; 658 classifiers have an accuracy of 85% or better; 451 classifiers have an accuracy of 90% or better; and 124 classifiers have an accuracy of 94% or better.

Table 23 through Table 26, show the ten features ranked based on the number of times they were present in classifiers for each accuracy bin. Across these tables, the pulse width feature was always in either first or second place, while the frequency feature took the other slot in all but the 75% threshold table. Surprisingly, E_{max} was consistently ranked ninth or tenth.

The highest accuracy without overfitting is the most desirable outcome when making predictions. Therefore, the results of the 124 classifiers with 94% or greater accuracy are most valuable for ranking the priority of the features.

Frequency, pulse width, and inject time were present 100% of the time for classifiers with 94% or greater accuracy and, therefore, would be the features most highly recommended for data collection.

Table 23: Feature Rankings For 75%+ Accuracy Classifiers

Rank	**Feature**	**Times Used**	**% Of Classifiers (885)**
1	*Pulse Width*	*506*	*57.18*
2	*Inject Time*	*504*	*56.95*
3	V_{rms}	*448*	*50.62*
	Measured Power	*448*	*50.62*
5	*Frequency*	*446*	*50.40*
6	*Input Power*	*444*	*50.17*
7	V_{p2p}	*440*	*48.72*
8	V_{min}	*416*	*47.01*
9	E_{max}	*379*	*42.82*
10	V_{max}	*375*	*42.37*

Table 24: Feature Rankings for 85%+ Accuracy Classifiers

Rank	Feature	Times Used	% Of Classifiers (658)
1	*Pulse Width*	*437*	*66.41*
2	*Frequency*	*382*	*58.05*
3	V_{p2p}	*368*	*55.92*
4	V_{min}	*350*	*53.19*
5	V_{rms}	*336*	*51.06*
6	*Measured Power*	*334*	*50.75*
7	*Input Power*	*331*	*50.30*
8	*Inject Time*	*324*	*49.24*
9	E_{max}	*314*	*47.72*
10	V_{max}	*270*	*41.03*

Table 25: Feature Rankings for 90%+ Accuracy Classifiers

Rank	Feature	Times Used	% Of Classifiers (451)
1	*Frequency*	*381*	*84.47*
2	*Pulse Width*	*305*	*67.62*
3	*Inject Time*	*260*	*57.64*

4	V_{rms}	*235*	*52.10*
5	*Input Power*	*233*	*51.66*
6	V_{min}	*232*	*51.44*
7	*Measured Power*	*229*	*50.77*
8	V_{p2p}	*226*	*50.11*
9	E_{max}	*186*	*41.24*
10	V_{max}	*172*	*38.13*

Table 26: Feature Rankings for 94%+ Accuracy Classifiers

Rank	**Feature**	**Times Used**	**% Of Classifiers (124)**
1	*Frequency*	*124*	*100*
	Pulse Width	*124*	*100*
	Inject Time	*124*	*100*
4	*Input Power*	*64*	*51.61*
	V_{rms}	*64*	*51.61*
6	*Measured Power*	*62*	*50.00*
7	V_{p2p}	*56*	*45.16*
8	V_{min}	*44*	*35.48*

9	V_{max}	*32*	*25.81*
10	E_{max}	*20*	*16.13*

6. Comparison of Frequency, Pulse Width, and $E_{max\ Trained}$ Classifiers

Given the suggestion from Table 22 that Frequency, Pulse Width, and E_{max} are present in all of the top ten highest performing classifiers, it is worthwhile to determine what performance is possible using only combinations of these three features. However, readers will likely note that to build a classifier containing $E_{max,}$ one would need the Measured Power feature data, and since available, it should be used in training. Consequently, Measured Power will be added as a fourth feature.

Table 27 presents the prediction accuracy of all 15 possible classifiers. The best performing classifier in Table 27 is the three-feature classifier containing Frequency, Pulse Width, and Measured Power. The second place (Freq., E_{max}, Meas. Power) and third place (Freq., E_{max}, Pulse Width) are also three-feature trained classifiers. However, the fourth three-feature classifier (Pulse Width, Meas. Power, E_{max}) is down in seventh place. Most importantly, the first through fourth place classifiers only differ by 0.02%, which is firmly within the variation of data selection. Any of these classifiers would be considered a worthwhile classifier.

An interesting result is the performance of the fourth three-feature combination (Pulse Width, Meas. Power, E_{max}). It has an accuracy of 79.61%, which is on par with the other two feature classifiers. This result makes sense, given that Pulse Width and Measured Power make up E_{max}, so the classifier is effectively trained by only two features worth of information presented in three different ways. Classifiers that use E_{max} and do not contain

Pulse Width or Measured Power are implicitly fed two features instead of one. When assessed in this way, the top four from Table 27 are, more or less, the same classifier.

Table 27: Frequency, Pulse Width, and Emax Trained Classifier Comparison Ranking

Rank	**Features**	**Training Inputs**	**Accuracy (%)**
1	*3*	*Freq., Pulse Width, Meas. Pow*	*91.38*
2	*3*	*Freq., E_{max}, Meas. Power*	*91.37*
	3	*Freq., E_{max}, Pulse Width*	*91.37*
4	*4*	*Freq., E_{max}, Pulse Width, Meas. Power*	*91.36*
5	*2*	*$E_{max,}$ Freq.*	*90.60*
6	*2*	*Pulse Width, Meas. Power*	*79.62*
7	*3*	*Pulse Width, Meas. Power, E_{max}*	*79.61*
8	*2*	*E_{max}, Meas. Power*	*79.60*
9	*2*	*E_{max}, Pulse Width*	*79.47*
10	*1*	*E_{max}*	*76.38*
11	*2*	*Freq., Meas. Power*	*71.25*
	2	*Freq., Pulse Width*	*71.25*
13	*1*	*Measured Power*	*71.19*
14	*1*	*Pulse Width*	*63.13*

	1	*Frequency*	*63.13*

7. Comparison of Freq., Pulse Width, and Inject Time Trained Classifiers

The evidence gleaned from Table 26 that frequency, pulse width, and inject time are always present in 94% accuracy binned classifiers suggests that it is worthwhile to compare the performance of classifiers trained using only these three features.

Table 28 presents the Classifier Accuracy of all seven combinations of these classifiers.

Table 28: Frequency, Pulse Width, and Inject Time Trained Classifier Comparison Ranking

Rank	**Features**	**Training Inputs**	**Accuracy (%)**
1	*3*	*Freq., Pulse Width, Inject Time*	*91.37*
2	*2*	*Freq., Pulse Width*	*67.42*
3	*2*	*Freq., Inject Time*	*65.02*
4	*1*	*Inject Time*	*63.13*
	1	*Pulse Width*	*63.13*
	1	*Frequency*	*63.13*
7	*2*	*Inject Time, Pulse Width*	*58.90*

The best performing classifier is the three-feature classifier with a 91.37% accuracy which is exactly on par with the other three-feature classifiers seen in Table 28 – ignoring the fourth combination (Pulse Width, Meas. Power, E_{max}) for the reasons stated in the

previous section. The other one and two feature classifiers do not perform well as they all have an accuracy of less than 68%.

These results are consistent with what is expected from two and three feature trained classifiers, although these features result in fewer useful (greater than 75%) classifiers.

Chapter 8: Comparison of Prediction Accuracy by Training Data

This chapter focuses on determining the extent to which a Weighted k-NN machine learning classifier can predict upset in microcontrollers of the same model, same architecture, and most importantly, of different architectures. Moreover, it is unclear to what extent prediction accuracies of 75% or better can be obtained by using different combinations of the features for each of the four datasets in training and testing for multi-device trained classifiers.

Classifiers were trained using data from four different microcontrollers. The datasets contained ten features relating upset to the IEMI waveform characteristics. Different combinations of the variables were used as the predictor inputs for machine learning training as performed in Chapter 7. Additionally, combined datasets such as a dataset containing data from MCU1, MCU2, & MCU3 were built to assess the impact of training accuracy on mixed datasets. The new dataset contained between 30,240 and 120,960 upset/null-upset data samples based on the number of combined datasets. The datasets were simply appended to one another – for example, the dataset containing MCU1, MCU2 & MU4 would contain 120,960 data points and would just be a new dataset containing the data from MCU1, MCU2, & MCU4.

After training and validation, each classifier was tested, separately, against each of the datasets for MCU1–4 to determine the overall accuracy of the classifiers. Just as in Chapter 7, the median performance classifier was used to establish repeatable results, which would otherwise be inconsistent due to the percentage-based data partitioning method.

Additionally, the prediction accuracy trends of the four original datasets were compared (as a function of the number of features used in training).

The following tables and graphs show classifier accuracy performance data in percentage. The percentage value is established from the confusion matrix's true-true and false-false prediction results from the validation step of building a classifier. These two numbers were averaged to give an overall metric for how well the classifier can predict upset data when shown upset data and how well the classifier can predict null-upset data when shown null-upset data.

Each classifier was trained 125 times to determine the median value. This value is considered the "truth" value since both the mean and median results have converged upon one another.

1. Same Model Chip Classifier Accuracy Trends

An easy way to evaluate whether the classifiers built for MCU1 can apply to another device, for example, MCU2, is to build the same set of classifiers made in Chapter 7 for MCU2 and then compare them. Figure 41 shows three plots, one for the minimum, median, and maximum classifier based on the number of features used for MCU1 and MCU2. For example, the minimum classifier plot shows that the performance observed for MCU1 and MCU2 is near 70% when five features are present. Conversely, in the maximum performance classifiers, when five features are present, the performance observed for both devices is approximately 94%.

It is important to understand that MCU1 and MCU2 contain the same range of data but are separately collected datasets. Therefore, comparing the classifier trends for each of the three plots suggests how similar the two datasets are. More importantly, the datasets produce the same classifier performance. Given this encouraging similarity, it is expected that when the MCU1 trained classifiers make predictions on the MCU2 data, and vice versa, the performance will be nearly the same regardless of what classifier is chosen. Furthermore, this suggests that only a single instance of empirical data on a chip needs to be gathered to make high-quality upset predictions. Comparison of the maximum performance trends shows that MCU2 is nearly identical and only varies by an average of 0.53% from MCU1.

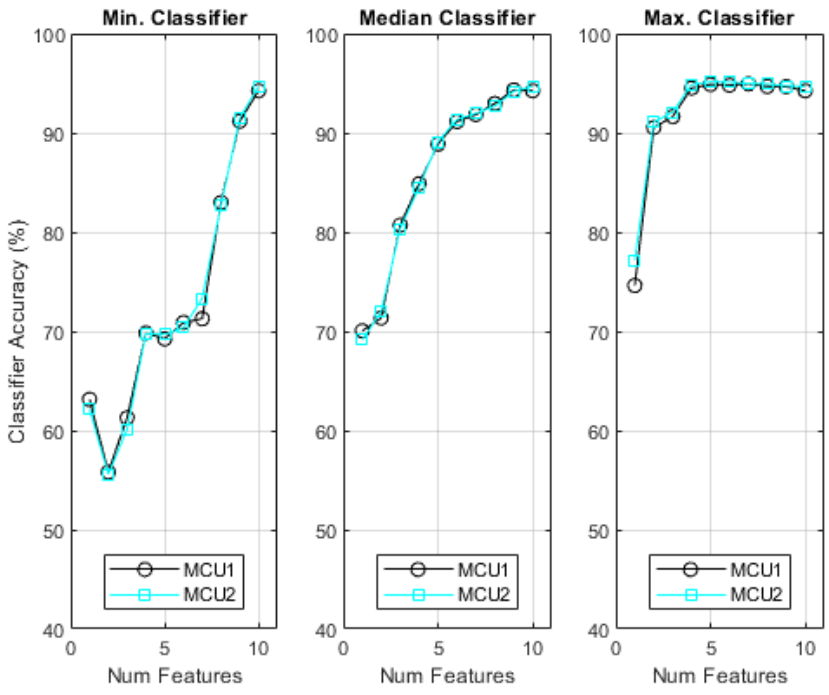

Figure 41: MCU1 & 2 Comparison of Minimum, Median, and Maximum Classifier Accuracies

2. Same Architecture Classifier Accuracy Trends

The minimum, median, & maximum classifier performances, as a function of the number of features, were compared with the addition of MCU3. MCU3 has the same architecture as MCU1 &MCU2 but is a different model and footprint.

Figure 42 shows three plots containing the minimum, median, and maximum classifier accuracy achievable based on the number of features used in training. The trends for MCU3 are similar to those seen in MCU1 and MCU2. There is a noteworthy difference in the minimum performance for MCU3 as around the five-feature region of the plot, MCU3 has an accuracy of about 80% while MCU1 and MCU2 only have accuracies of 70%. In general, this difference suggests that the MCU3 dataset is more separable – meaning that the distance between upset and null upset data clumps are more easily differentiated between - than MCU1 and MCU2 because a higher minimum accuracy is achievable in that region. However, it is worth noting that identical testing criteria and instruction code were used on all three devices, so the only known difference is the test asset containing a new chip and interface board. Comparison of the maximum performance trends shows that MCU3 is very similar to MCU1 and MCU2 and only deviates an average of 0.49% from MCU1 and 0.034% from MCU2. It is worth mentioning that the 5% difference seen between MCU2 and MCU3 when using only one feature is largely unsurprising as there was a nearly 2.5% difference in that data point between MCU1 and MCU2. It is valuable to note this slight change in performance as the next section shows the trends associated with a microcontroller that is a different chip and a different architecture.

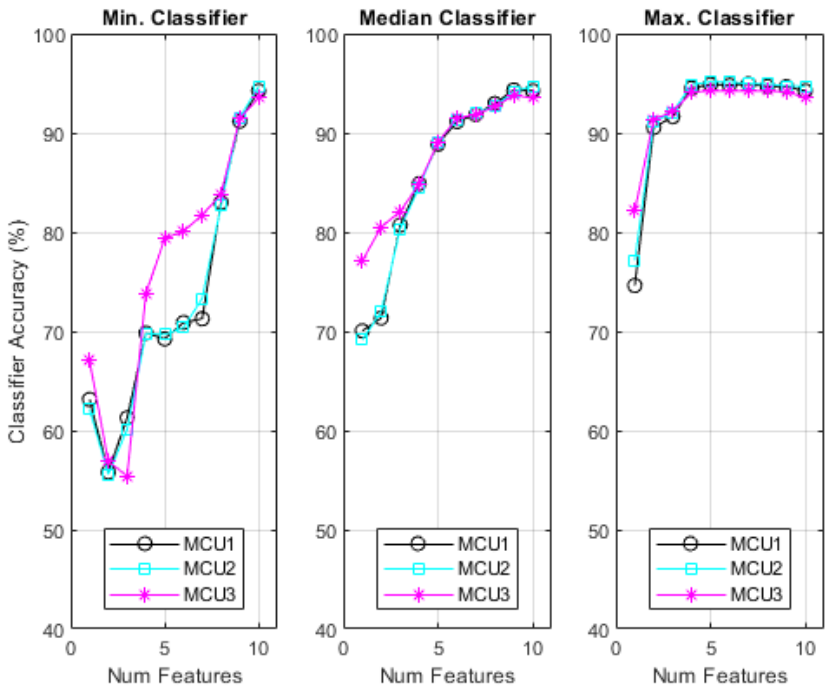

Figure 42: MCU1, 2 & 3 Comparison of Minimum, Median, and Maximum Classifier Accuracies

3. Multi Architecture Classifier Accuracy Trends

Figure 43 shows the trends for MCU1 through MCU4. MCU1 and MCU2 are the same chip, MCU3 is a different chip but the same architecture as MCU1, and MCU4 is a different chip and architecture from MCU1. Overall, the trends are similar for all four devices but have noticeable differences in actual performance values. For example, MCU4 shows the same shape trend in the maximum classifier accuracy plot as seen in MCU1-3 but is shifted upward towards an accuracy of 98% compared to the MCU1-3's 94%. MCU4 deviates on average about 4.23% from MCU1.

The major aspects worth taking away from this analysis are: 1) that same architecture devices, and the data collected on them, are incredibly similar, and therefore classifiers trained with MCU1, MCU2, or MCU3 should be able to make highly accurate (greater

than 90%) predictions on the other two unused datasets and 2) MCU4's performance trends suggest that predictions made on it when using a classifier trained with MCU1, MCU2, or MCU3 data will be favorable (>70%) but will likely not have the same level of accuracy expected of MCU1-3.

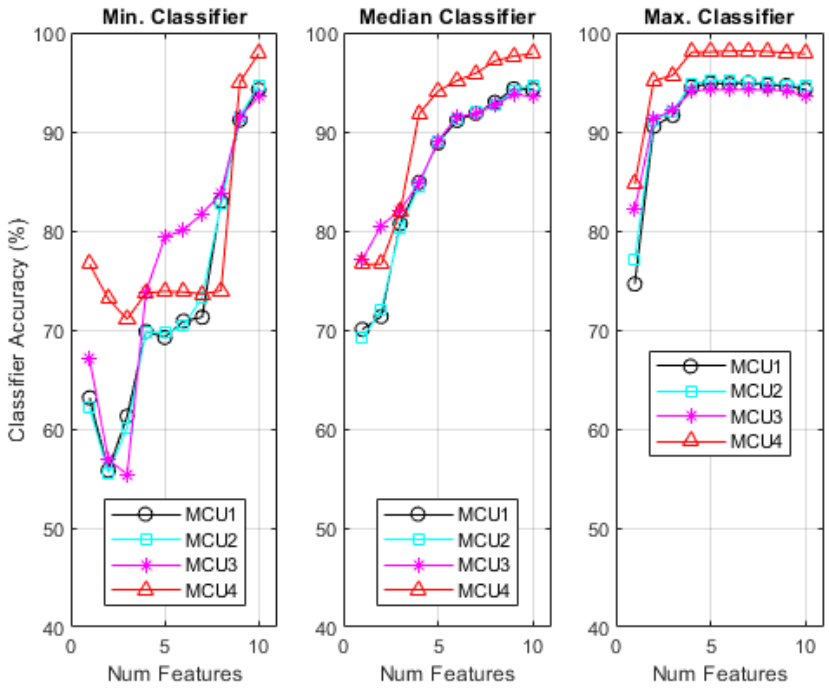

Figure 43: MCU1,2, 3 & 4 Comparison of Minimum, Median, and Maximum Classifier Accuracies

4. Cross Chip Predictions with MCU1 trained Classifier

The six-feature classifier using MCU1's data and containing the Frequency, Inject Time, Pulse Width, V_{rms}, V_{max}, and E_{max} features was established in Chapter 7 to be the best performing classifier with a prediction accuracy of 94.51%. This classifier will be tested – separately - against data from MCU1, MCU2, MCU3, MCU4, MCU23 & MCU234. MCU2 through MCU4 each contain 30240 examples of upset, while a combined data set such as MCU23 contains 60480 examples of upset.

Table 29 shows the prediction results from each dataset. This data suggests a handful of key takeaways: 1) the classifier is well suited to the MCU1 dataset because it was able to identify all 30240 of the examples of upset correctly; 2) the classifier was able to correctly predict 94% of the MCU2 dataset which suggests that the two datasets – and therefore devices – differ in an unknown way by about 6%; 3) the classifier was able to correctly predict 89.16% of upset data originating from the same architecture, but a different MCU device; 4) the classifier predicted 80% of MCU4's data correctly; And lastly, 5) that the prediction accuracy – unsurprisingly - does not change when the data is combined, it simply just sums.

It should be noted that the 6% difference seen between MCU1 and MCU2 could be attributed to a manufacturing difference, systematic variation in the apparatus, or the well-documented phenomena where seemingly identical IEMI waves injected on a device provided both upset and no upset outcome with repeated sampling – See Chapter 9 for further discussion.

Overall, this analysis suggests that IEMI induced upset predictions of 80% accuracy, or better, can be made on MCUs performing similar tasks regardless of make, model, or architecture, with an appropriately trained Weighted k-NN classifier.

Table 29: Maximum Classifier Prediction Accuracy by Number of Features

Dataset	**Accurate Predictions**	**Prediction Accuracy (%)**
MCU1 (30240)	30240	100
MCU2 (30240)	28434	94.02
MCU3 (30240)	26962	89.16
MCU4 (30240)	24216	80.07

MCU23 (60480)	55396	91.59
MCU234 (90720)	79612	87.75

5. Single, Multi-Device Trained Classifier Testing

The next step in this analysis is to determine whether a given combination of the four datasets may result in a higher-performing classifier for all the devices.

Fifteen ways to combine the four datasets were examined. Each combination was used as the training input. Each classifier was iterated 125 times, and then the median performing iteration was used to make predictions on each of the four original MCU datasets individually. It is important to understand that making predictions on combinations of the dataset does not result in more information since the order in which the data is presented does not matter. Each data point is assessed separately – therefore, testing data from MCU1 & MCU2 is the same as testing data from MCU1 then MCU2. Combined dataset predictions are simply the sum of the predictions for each dataset.

Table 30: Prediction Accuracy with Different Training Inputs

Training Data (MCU)	**Prediction Accuracy (%) MCU1 // MCU2 // MCU3 // MCU4**				**Mean**
1	**100**	94.03	89.16	80.08	90.82
2	94.03	**100**	88.6	79.01	90.41
3	89.16	88.6	**100**	81.27	89.76
4	80.08	79.01	81.27	**100**	85.09
1,2	**97**.5	96.52	89.61	81.44	91.27
1,3	92.62	89.6	**96.54**	82.87	90.41
1,4	83.3	81.18	82.63	**96.78**	85.97

2,3	90.37	91.84	**96.75**	82.69	90.41
2,4	82.2	82.28	**96.74**	**96.74**	89.49
3,4	80.07	79.12	85.86	**95.41**	85.12
1,2,3	**97.29**	96.73	91.87	80.41	**91.58**
1,2,4	**97.55**	96.48	89.74	82.53	**91.58**
2,3,4	91.86	93.17	**95.43**	85.84	**91.58**
1,2,3,4	93.61	92.66	**94.11**	85.93	**91.58**

Table 30 shows the classifier prediction results for each training set when validated against each dataset. The only noticeable trend is that the less similar the devices are to what was used in training, the lower the prediction accuracy. This is confirmed by the first four rows of the table as predictions of 100% are achievable on the dataset used for training and then decrease as the devices become less similar. Based on the expectations identified in the previous section, this performance is what was expected. Fortunately, the performances seen are still excellent overall. MCU4 is different enough to incur degraded prediction accuracy, but all devices still have a 79% or higher prediction accuracy regardless of what was used as the training input. The overall best combination for training would be the MCU12 training set, as it required the least amount of data and produced the highest relative prediction accuracies for each of the four datasets. The training sets, which contained three to four sets of data, only showed a 0.3% improvement over the MCU12 dataset.

Table 31 presents the first four rows and columns of Table 30 as these rows show an interesting outcome not expected. This matrix of data is representative of a symmetric matrix, which suggests that there is reciprocity between the datasets used in training and the predictions to be made. According to Pozar, a matrix like this would be akin to a

passive, lossy, reciprocal network with s-parameter terms as the values. [16] In a generic sense, the attributes of a passive, lossy, reciprocal network would suggest that the network – in this case machine learning method's performance - is stable and not biased by the inputs to favor one device over another. Moreover, this suggests that the results are not a fluke of a single well-trained classifier, as the matrix would not be reciprocal if the method were inconsistent or biased towards specific datasets.

An advantage of this reciprocity is that when collecting additional data on new devices, only a portion of the analysis is needed to fully understand the training inputs to prediction outputs when not combining the datasets. This is known because either the upper or lower halves of the matrix can be inferred with knowledge of the other. For more extreme problems, larger datasets, or greater numbers of datasets, this could result in computational or time savings.

Table 31: Prediction Accuracy of Each Device as the Training Input with 100% of Dataset

Training Data (MCU)	**Prediction Accuracy (%) MCU1 // MCU2 // MCU3 // MCU4**				**Mean**
1	**100**	94.03	89.16	80.08	**90.82**
2	94.03	**100**	88.6	79.01	90.41
3	89.16	88.6	**100**	81.27	89.76
4	80.08	79.01	81.27	**100**	85.09

6. Impact of Sparsed Training Datasets on Prediction Making

The same training method was repeated with edited datasets to understand if this reciprocal result was a fluke or not. The four datasets were edited by randomly sparsing data from the original datasets. Sparsing is the term used in Machine Learning to describe

selecting a subset of data from a larger dataset. Since this analysis has focused on training with the full dataset of each microcontroller, it was imperative to determine how prediction accuracy changed when fewer data points were used as the input. Consequently, the prediction accuracy results for datasets that contain only 10% (3240 points) and 1% (303 points) of the original data were checked – these datasets are referred to onward as the 100%, 10%, and 1% datasets. Since it has been shown that data selection can influence the results, the entire experiment was repeated ten times, and the overall results were averaged. For clarification, a specific run-through of the methodology executed would be as follows: the MCU1-4 data sets containing 30240 data points were 'sparsed' down with a 1% factor to randomly select 303 data points. The 303 points from the MCU1 dataset were then passed to the machine learning codes, which then applied a 25% hold-out of the data for validation. This training and validation process was repeated 125 times to establish the median-performing classifier. Then the median classifier was used to predict upset on the 'sparsed' MCU1, MCU2, MCU3, & MCU4 datasets, and prediction accuracy was recorded. This was repeated nine more times to account for a different 25% hold-out of data. The average prediction accuracy results for 1% and 10% can be seen in Table 32 and Table 33.

Table 32: Prediction Accuracy using 1% of MCU Dataset

Training Data (MCU)	**Prediction Accuracy (%) MCU1 // MCU2 // MCU3 // MCU4**				**Mean**
1	**100**	83.96	82.70	79.40	86.51
2	85.34	**100**	82.50	78.81	86.66
3	83.56	82.11	**100**	81.58	86.81
4	76.43	74.78	77.49	**100**	82.17

Table 33: Prediction Accuracy using 10% of MCU Dataset

Training Data (MCU)	***Prediction Accuracy (%) MCU1 // MCU2 // MCU3 // MCU4***				***Mean***
1	**100**	91.46	87.13	80.57	89.79
2	91.29	**100**	86.50	78.98	89.19
3	87.44	86.79	**100**	81.34	88.89
4	80.35	78.66	80.83	**100**	84.96

Taken holistically, Table 32, Table 33, and Table 31 suggest that more data results in a more reciprocal result and a more accurate prediction outcome. Since the reciprocal nature is based on the amount of data used, this observed result is coincidental because no one correct answer exists for how much data should be collected. Moreover, the change in classifier accuracy when using less data is a far more exciting result. The difference in classifier accuracy for the 1% and 10% data compared to the 100% data can be seen in Table 34 and Table 35. Surprisingly, the change in prediction accuracy is not as significant as anticipated. The 10% dataset showed between a 0.13% and 1.02% decrease in accuracy based on which MCU device was used as the training data set; and the 1% dataset showed between a 2.9% and 4.3% decrease in prediction accuracy.

Table 34: 1% dataset's change in accuracy from the 100% dataset

Training Data (MCU)	**Prediction Accuracy (%) MCU1 // MCU2 // MCU3 // MCU4**				**Mean**
1	0	-10.07	-6.46	-0.68	-4.30

2	-8.69	0	-6.1	-0.2	-3.74
3	-5.6	-6.49	0	0.31	-2.94
4	-3.65	-4.23	-3.78	0	-2.91

Table 35: 10% dataset's change in accuracy from the 100% dataset

Training Data (MCU)	**Prediction Accuracy (%) MCU1 // MCU2 // MCU3 // MCU4**				**Mean**
1	0	-2.57	-2.03	0.49	-1.02
2	-2.74	0	-2.1	-0.03	-1.21
3	-1.72	-1.81	0	0.07	-0.86
4	0.27	-0.35	-0.44	0	-0.13

The slightness of the decrease in accuracy is encouraging because it suggests that big data is not required to make meaningful predictions. However, it should be understood that more data is better, but only when the data is valid. As communicated beforehand, data selection is vital because prediction accuracy can vary significantly based on the data used in training – bad data in, bad classifier out.

Overall, a 1% dataset still maintains the general accuracy performance trends as the 100% dataset. The trend is that the more different the predicting DUT is from the training DUT, the lower the accuracy. Comparison of the mean performance of the three datasets results in Table 36 and summarizes these trends. These trends and the raw values seen in Table 31, Table 32, and Table 33 are necessary to digest because they suggest that the amount of data used in training may lead researchers to believe that two DUTs are more different than they are. For example, recall from the experimental setup description

that in data collection, the only known difference between the MCU1 and MCU2 is the actual instance of chip – The test criteria, breakout board, and cabling were all identical. However, in the 100% dataset, there is an approximately 6% difference in the predictions made between MCU1/2 and MCU2/1. In contrast, there is an asymmetrical 10% and 8.7% difference in the predictions on the 1% training set used. It is up to the researcher whether the symmetry and loss of ~3% accuracy are meaningful, but the result is noteworthy.

Table 36: Comparison of Spared Datasets' Prediction Accuracy

Training Data (MCU)	**Mean Prediction Accuracy (1%)**	**Mean Prediction Accuracy (10%)**	**Mean Prediction Accuracy (100%)**
1	86.51	89.79	**90.82**
2	86.66	89.19	90.41
3	86.81	88.89	89.76
4	82.17	84.96	85.09

The last meaningful aspects to note from this data are: MCU1 and MCU2 are the same devices, but the change in prediction accuracy between them using the 100% data is 5.97%; The difference between MCU1 and MCU3, which are different chips, with different pinouts, and consequently different breakout boards, is 10.84%; Finally, the difference between MCU1 and MCU4, which are different chips, boards, and device architectures, is 19.92%.

Chapter 9: Conclusions and Interpretation of Results

The chapter will be broken into five sections, one for each of the four research questions mentioned in Chapter 1 and a final section that provides an overall conclusion for this work. Each of the first four sections will restate the research question and discuss the results that lead to answering that question. Additionally, a discussion will be provided for what the results may suggest in the larger sense of the work as a whole.

1. Traditional and Existing Upset Trends (Chapter 5)

Research Question 1*:* *Are there upset trends that humans can discuss and interpret?*

Yes, the waveform characteristics - frequency, pulse width, power, and injection time - showed trends that humans could interpret – and generally align with known mathematical functions. Example graphs of these functions can be found below: Frequency – Log-normal distribution function; Pulse width – Logarithmic function; Power – Linear function; Injection time – Sawtooth wave function. Moreover, the trends' shapes were very similar for all four devices.

With respect to frequency, upset is most likely (>0.8 Probability of Upset) at 20, 50, 100 MHz, while upset is less likely (< 0.2 PoU) at 200 & 400 MHz, and 0.0 for 800 MHz. This set of thresholds suggest that higher frequencies do not correspond to higher PoU and that the devices share a sort of "resonance" for IEMI upset between 20 and 100 MHz. Therefore, the devices appear to couple IEMI signals (or RF signals, if speaking generally) best at those frequencies.

The mathematical function that best relates to this trend would be a log-normal function. However, additional frequency data would be needed to fill this trend's graph in more detail because the data currently held is a sparse representation of a narrow frequency space. Moreover, it is well understood from microwave engineering that a slight change in frequency can significantly change circuit coupling. Therefore, it is not realistic to assume that all IEMI signals with frequencies from 20 MHz to 1000 MHz will effectively produce an upset response or that the coupling effectiveness even follows the continuous nature of a log-normal function over the frequency range.

Frequency was found to influence all the trends of the other features by acting as a damping or amplifying function based on the frequency selected. Frequencies that were presumed to couple best result in an amplified, or prominent, trend, while frequencies that coupled poorly resulted in a damped, or nonexistent, trend. For these devices, 50 MHz and 100 MHz resulted in the most prominent trends, while 800 MHz resulted in no trend. The damping and amplifying prominence of the trends is most easily observed with the pulse width trends by frequency.

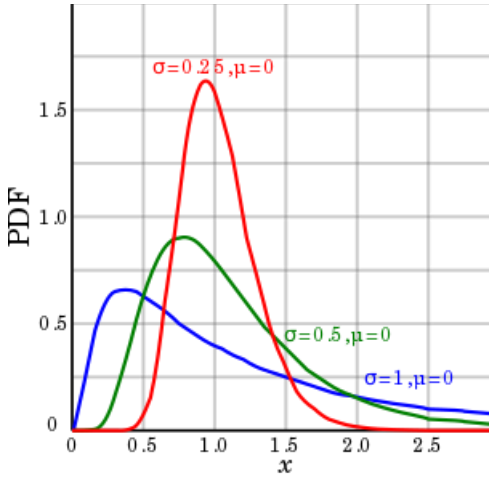

Figure 44: Example of a 'Log-Normal Distribution Function' from Wikipedia

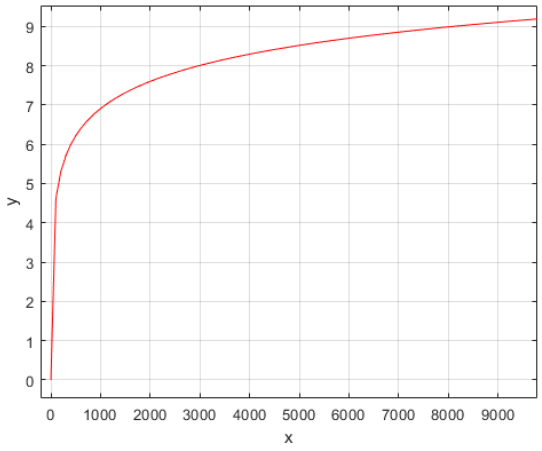

Figure 45: Example of a ‘Logarithm Function’ from MATLAB

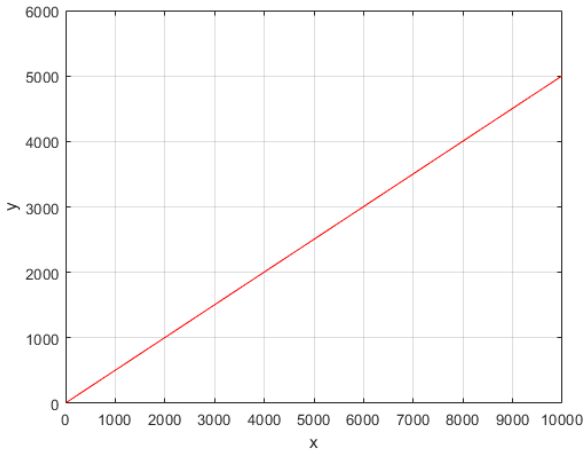

Figure 46: Example of a Linear function from MATLAB

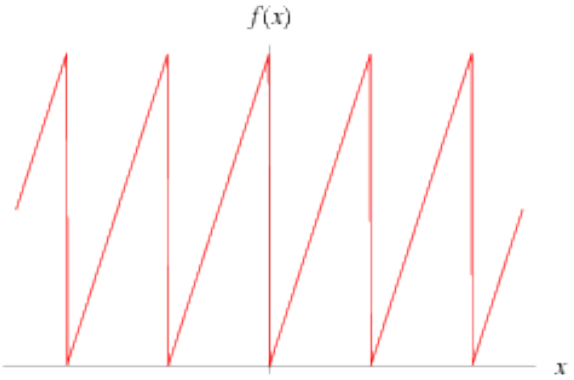

Figure 47: Example of a ‘Sawtooth Wave Function’ from Wolfram

A logarithmic function can approximate the general upset trend shape for pulse width. Consequently, the area under the curve for each frequency’s pulse width trend was largest for the best coupling frequency and decreased significantly as the frequency changed and coupled less well.

Pulse widths with greater than 100 ns suggested a strong diminishing return in the 20 MHz, 200 MHz, and 400 MHz frequencies while the 50 MHz and 100 MHz diminished beyond 400 ns. These values suggest potential upset thresholds for pulse width – meaning that pulse widths beyond these points do not significantly improve the PoU for a large change in value. However, pulse widths that exceed a duration of more than one clock cycle have not been tested. The author believes that IEMI signals with high amplitudes and pulse widths exceeding one clock cycle would entirely overwhelm the expected clock bitstream. This is because the IEMI signal(s) would be larger than or on the same voltage scale as the clock signal and, therefore, with a long pulse width, could overlap multiple cycles. It is well understood that microcontrollers rely on the changing state of the clock to initiate data hand-offs, acknowledgment of bits, and register changes. Consequently, erroneous signals with sufficient pulse width on the clock line would be

expected to result in upset-like outcomes such as execution of register changes, instructions, and software malfunction.

Regarding input power, more power roughly translates to a higher upset probability. However, it was determined that when a power of 8+ watts was present, a PoU of 0.8 or greater was achieved in frequencies between 20 MHz and 400 MHz. This trend suggests a clear power threshold not based on frequency. However, at 800 MHz, the PoU was 0.0, which suggests that RF needs to couple to the circuit to cause an upset. This power threshold strengthens the suggestion that the upset mechanism is tied to the clock being corrupted. However, electron avalanches in transistors have long been posited as the potential upset mechanism in analog devices, such as logic gates and amplifiers. Nevertheless, it is the author's opinion that electron avalanche is **not** the cause of upset in these experiments because 1) the IEMI event is taking place solely on the clock line. 2) The clock line does not directly influence (or connect to) the output registers. 3) The IEMI signal would need to cross-talk through the device to reach the output line transistors. 4) Cross-talk signals typically incur significant attenuation to amplitude, making them less likely to have the strength required to cause transistor avalanche. Finally, 5) the repeatability of the experimental results from Guillette and Clarke's 2018 work suggests that minimal degradation occurred in the electronic device. The rationale originating from this paper is that if internal transistors were being forced to change state (experiencing avalanche), there would likely be a weakening of the PN, or NP, junctions and the devices would upset more easily at lower power levels in repeated testing. This was not observed. Consequently, Occam's Razor would suggest that it is more likely that the device is acting abnormally because it is not receiving the clock signal it needs to

operate nominally. Therefore, the clock signal is being corrupted by the presence of the IEMI signal.

Injection time has the highest upset probability during clock rise and fall states – for MCS-51 devices, while MCU4 bucks this trend and is highest during clock high times. At first glance, this makes sense because data is latched by the MCU device at the rise and fall time for the MCS-51 architecture and the clock HIGH and LOW times for the PIC architecture. [66] [48] These timing methods are generally referred to as edge trigger or level trigger, respectively. An edge trigger activates during a state change, while a level trigger occurs when a specific voltage is reached. [67] However, the method to capture the clock state may not be the driving factor here because, in the case of the PIC device, the external clock is transformed internally into a differently timed clock managed by the device. Specifically, the instruction cycle time is internally changed to be four times the clock input time-period which contrasts with the MCS-51. Furthermore, the MCS-51 has a 1:1 timing cycle to instruction execution cadence, while the timing for the PIC device is further complicated by the fact that it implements multi-instruction fetching and execution – also known as pipelining. This allows a device to use one complete clock cycle to initiate/execute multiple instructions within a single cycle instead of waiting for the first instruction to complete. Regardless of what is causing the difference, the data suggests a meaningful difference that, at the very least, can be attributed to phenomena known to occur by researchers. Further study to determine upset mechanisms may illuminate and help reconcile the differences seen here.

The overarching takeaway from this trends analysis comparison is that despite the differences in architecture, software, execution, and timing, all four devices have similar

trends responses for each of the four core waveform characteristics. It was concluded that MCU4 was observed to have a consistently lower upset trend but did still follow the same trend shape seen in MCU1-3. MCU4's lower upset trend suggests that it is more resistant to IEMI signals. This is not entirely surprising given that it is the newest and most advanced device of the four. However, MCU4's trend shapes were similar to MCU1-3 for all features, but the clock injection feature suggests that even though the architectures, code, breakout boards, and chips are different, the upset response is similar and consequently, so too, is the upset mechanism. Furthermore, the clock injection feature trends are different because the timing and execution of instructions occur with different timing intervals between the two architectures. Regardless, the upset mechanism is believed to be the same across the four devices because the same type of upset effects are produced when IEMI signals of sufficient pulse width, power, and frequency are present.

The common upset mechanism is believed to be clock line disruption caused by IEMI signals which muddle the clock signal to have additional clock states at times not expected by the MCU. These additional clock states, when interpreted, cause timing, register, or instruction mistakes that cascade forward through the device and manifest into alternative outputs described as recoverable upsets like the "Shift type" upset or unrecoverable upsets like the "Latch type" upset.

2. Survey and Selection of Machine Learning Methods (Chapter 6)

Research Questions 2: *To what extent can fast-to-train machine learning methods be used to predict upset?*

k-Nearest Neighbors and Decision Trees performed as well as, or better than, the more complex and longer to train Support Vector Machine method. Weighted *k*-NNs were the highest performers, with a performance of 94.3%.

A study was performed to determine how performance changed when the number of neighbors, distance method, and weight metrics were changed for the *k*-NN. Changing the Distance Metric from Euclidean to City Block resulted in higher, more stable performance across a broader spectrum of Number of Neighbors. The performance was increased to 94.7%. This minimal change in improvement is not considered a major result, as past researchers have concluded that some algorithms are better suited to data than others. Instead, the City Block Distance metric was found to have more consistent performance trends as the number of neighbors changed. This choice is meaningful, as it impacts experiments like those performed in Chapters 7 and 8, where the data was sparsed such that it could have resulted in a massive decrease in performance. Furthermore, using a classifier that is inherently more resistant to variation - with respect to the types of adjustments that can be made in training - pushes any experimentation with data to be more in line with the scientific method's core tenant of only changing one variable at a time.

The primary takeaway from this chapter is that machine learning, even with the most basic methods, shows itself to be well matched to IEMI electromagnetics problems. Moreover, this work showed that the simpler methods could perform at equal or better prediction accuracies than more complex methods such as GPML, SVM, and ANN.

3. Prediction Accuracy by Selection of Features (Chapter 7)

Research Question 3: *What features are needed to make high-quality predictions?*

Generically, this research suggests that IEMI upset predictions of 90% or better accuracy can be made with as few as two features. However, certain combinations of these features are better than others. From the trends analysis performed in Chapter 5, it was made clear that the four core features – frequency, pulse width, power, and inject time – had observable trends which suggested thresholds for where upset would likely take place. Moreover, frequency was seen to be a feature that consistently influenced the trends of the other features. The analysis done in Chapter 7 is consistent with these findings as combinations of data that contained the core four features routinely produced a prediction accuracy of around 90%.

It is recommended that researchers prioritize collecting the frequency and pulse width waveform characteristics while keeping in mind that other features can also contribute to high-performing classifiers. In a practical sense, this work provides statistical evidence that the core features are strongly tied to high accuracy predictions, especially frequency, pulse width, and measured power.

It is important to note that the single feature prediction accuracy does not contradict these findings. Instead, it proves that the features depend upon one another, which is consistent with what was seen in the traditional trends analysis, as none of the features on their own produced a high-performing classifier (P >75%).

Although the best performance classifiers indeed contained six to seven features with a 94% performance, the two and three feature classifiers using only frequency and E_{max},

or frequency, pulse width, and measured power, resulted in 90-91% accuracy. A 3-4% improvement in accuracy is meaningful. However, it does not strongly justify the requirement of the extra data. This result suggests that should researchers need to forgo collection of, or training with, the other features, they can still achieve good results.

In combination with this k-NN study, the trends analysis goes a long way towards reducing the black-box nature of machine learning because it provides human interpreted context to the outcomes based on the data. For example, researchers can see how the data may be separated into clumps, and the highest accuracy classifiers are made when using the data that is believed to be most descriptive of upset. This results in confidence that the classifier can pick up on the combined trends of the core four features and synthesize them in a way that makes valuable predictions that humans believe.

4. Comparison of Prediction Accuracy by Selection of Training Data (Chapter 8)

Research Question 4: *To what extent can a chip, architecture, and device-level classifier be made to predict upset in microcontrollers?*

The significant result is that a k-NN machine learning classifier trained on IEMI upset data from one microcontroller is highly accurate in predicting upset in microcontrollers exposed to the same IEMI of a different architecture or serial number. This suggests the broad applicability of a trained machine learning agent in predicting IEMI upset on microcontrollers as a device class.

Comparison of the classifier feature performance trends of the four MCU devices showed that the same architecture devices were within an average of 1% of each other. In contrast, the fourth device, which was a different architecture, had the same trends shape but deviated by more than 5%. In cross-validation – also known as prediction making – it was determined that MCU4 was different enough to incur degraded prediction accuracy but was similar enough that all devices still had a nearly 80% or higher prediction accuracy regardless of what data was used as the input training set for the predicting classifier. The overall best combination for training was the MCU12 training set. It used the least amount of data and had the highest average prediction accuracy of 91.27% when tested against the four microcontrollers' datasets. The training sets, which contained three to four sets of data, showed a 0.3% improvement over the MCU12 dataset with an average prediction accuracy of 91.58% across all four devices. This suggests that training with data containing the difference between two instances of the same device may be more important than training with data containing the difference between multiple devices – discussed more in the following paragraphs.

It was also noted that the prediction accuracy, based on the trained model, converges towards a reciprocal result as more data is used in training. For example, a classifier trained with MCU1 data making predictions on MCU3 will have the same accuracy as a classifier trained with MCU3 data making predictions on MCU1. This result suggests that time and computational saving can be realized by only computing the upper or lower half of the matrix and reflecting it about the diagonal. However, it does require a significant amount of data to achieve the reciprocal result. Datasets that contained less than 90% of the 30240 samples did not have "strongly" reciprocal outcomes. Consequently, this result

is not considered significant and is instead considered coincidental but interesting. Furthermore, at the very least, it suggests that the machine learning algorithm and repeatability of these results are self-consistent.

Furthermore, it was discovered that there was only a 1% decrease in average prediction accuracy when only 10% of the entire dataset was used as the training input – and a nearly 4% decrease in performance when using 1% of the dataset. This is an exciting and highly encouraging result because it further strengthens an argument for making meaningful predictions across multiple architectures, even with small amounts of upset data.

The change in prediction accuracy was compared to quantify the percent-based difference between the DUTs. Recall that MCU1 and MCU2 are the same devices, so the expectation is that they should have very similar prediction outcomes - when using 100% of the data. However, the prediction difference between them is 5.97%. MCU1 and MCU3, which are different chips, with different pinouts, breakout boards, and the same architecture, have a prediction difference of 10.84%. Moreover, the prediction difference between MCU1 and MCU4 - which are different chips, boards, and chip architectures - is 19.92%.

The prediction difference between MCU1 and MCU2 was larger than expected. It, therefore, was presumed to be more representative of systematic error from instrumentation drift or manufacturing variation between chips. Therefore, this 5.97% difference would be inherently present when cross predicting between any two devices. Consequently, the best classifier prediction accuracy possible would be 100 minus this value which is 94.03%, which is in line with the prediction values presented here.

In the grand scheme of things, the data suggests that these microcontrollers are fundamentally far more similar than they are different. This is likely why classification accuracies of 80% or greater are possible. However, it is the author's opinion that these results only go as far as the scope of what they investigated. Similar devices performing wildly different tasks or implemented in unique ways may respond differently, and therefore these classifiers likely would not be appropriate. Regardless, this body of work does show that it is possible to train multi-device classifiers that perform with greater than 80% accuracy depending on the fundamental differences between the training device and the predicting device.

5. General Conclusions

As stated in Chapter 1, the primary motivation of this work was to evaluate the extent to which machine learning could be applied to build a predictive model for electromagnetic induced upset. The approach collected basic information about the device and the IEMI waveform. By carefully selecting and testing the devices - four microcontrollers spanning three different circuit boards and two different chip architectures - empirical experimentation captured the impact of the waveform, device, board, and architecture on IEMI upset. Then, a machine learning algorithm was trained with nominal and upset operation examples when IEMI was present. The resulting machine learning classifiers were then validated against unseen data to determine the prediction accuracy.

A key challenge in this work, and similar work in this field, is that perfect information rarely exists to build a model that may be used in IEMI upset modeling and simulation (M & M&S). Consequently, the problem of predicting IEMI upset on a microcontroller is, by default, a black box type problem.

The application of machine learning presented and proven in this dissertation fills the M&S void by acting as the model by which simulation could then be performed. However, machine learning has gotten a reputation for abstracting its operation and outcomes such that end-users do not understand how the outcome was reached. To counter this issue, three of the most elementary machine learning algorithms, support vector machines, decision trees, and k- nearest neighbors, were used to determine their ability to predict upset on a single microcontroller. A comparison between methods found that the k-Nearest Neighbors algorithm was the simplest and highest-performing method. However, all of the methods produced 90% or greater accurate predictions.

Furthermore, it was resolved that collecting basic waveform data on a few different microcontrollers was sufficient to build a predictive model capable of correctly identifying IEMI upset in microcontrollers spanning multiple architectures more than 80% of the time. Classifiers trained using two or three of the four microcontrollers were found to perform almost equally as well as the one that contained all four – though certain combinations were better than others. The best-of-class training set contained all four devices and achieved an average accuracy of 91.58% when validated against all four devices. In contrast, the second-place training set only contained MCU 1 and 2 data and achieved an average accuracy of 91.27%. It was concluded from the trends analysis and machine learning validation analysis that the devices were more similar than they were

different and that training with data on even a single microcontroller could produce meaningful classifiers even against a device of a different architecture.

Fundamentally, this work is consistent with the notion that machine learning is only as good as the data it trains on. Furthermore, it suggests that limited data can be used to make excellent predictions regardless of their make, model, or architecture, given that the devices are implemented similarly and are executing a similar task. Consequently, follow-on work to prove these results hold when the problem is broadened to free-field testing and complex device implementations are important. Specific experiments and suggested future work can be found in the next chapter.

Chapter 10: Future Work

This dissertation assessed the extent to which the k-Nearest Neighbors Machine Learning method could predict upset on Microcontrollers with excellent results. However, additional research can still be performed. This chapter suggests follow-on work that could complement this dissertation.

1. Upset Mechanism Investigation

Additional experimentation could be performed to distort microcontroller clock signals by adding long duration, constant, or sporadic noise with increasing amplitude to determine the point at which similar upset outcomes occur. The goal of this study would not be solely focused on IEMI type waveforms to cause the noise but instead, approach it from a generic standpoint. Additionally, a contrived clock signal using an arbitrary waveform generator to produce signals which have additional clock states would go a long way towards proving the upset mechanism suggested in this research. However, specifically regarding IEMI waveforms, complementary research would be a repetition rate study and multi-clock-cycle-pulse-width study.

Multi-cycle-pulse-width study: Further experimentation could be performed to evaluate the impact of multi-cycle duration pulse widths. It has been suggested that longer-duration pulse widths may produce a different type of upset or reveal more about the upset mechanism.

Repetition Rate Investigation: It has been suggested by several external reviewers that a methodical study into repetition rate would be interesting to complement the trends study presented here.

The above experiments were not pursued in this dissertation, primarily because they would require a meaningful change to the experimental set-up in addition to a change in the scope for this work. The intent of this work was not to identify and prove an upset mechanism, nor was it to exhaustively define or prove traditional upset trends on microcontrollers. Instead, the scope of research focused on the application of machine learning which required data collection. The data collected enabled meaningful discussion and evaluation of trends and was therefore included as a bonus.

2. Application to Complex Devices Containing MCUs

An obvious next step would be to continue this research on untested processors such as ARM, INTEL, and FPGAs. From there, the k-NN algorithm could be applied to upset data on complex devices containing one of the chips previously tested to determine the extent to which the classifier's prediction accuracy decreases. Taking this approach would continue to methodically broaden the scope of the work while allowing for reach back into this dissertation or related work. After proving viability at this level, it would be important to explore data sets that contain free-field data to account for the coupling aspects of the overarching problem. Completion of free-field experiments may go a long way to enabling a generic IEMI upset classifier for digital devices in general.

This dissertation did not pursue other microchips, complex devices, and free-field coupling experiments because a crawl-walk-run approach was needed. As discussed in

the introduction, free-field coupling and complex devices increase the scope of the problem beyond what would be reasonable for a single dissertation. More importantly, the research foundation for this work has not been established to justify starting at such a complexity level. The devices tested in this work were selected to provide a limited but distinct set of differences between devices (and subsequent data). Additional devices could have been pursued, but from a practical standpoint, a line had to be drawn to ensure that the work did not become never-ending - Four devices spanning two architectures and three serial numbers was that line.

3. Other Machine Learning Methods

This work, and Bilalic's, have shown that k-NN, DT, SVM, GPML, and ANN all are up to the task of predicting IEMI induced upset. However, countless other machine learning methods may also be worth pursuing. It may be worthwhile to perform a more expansive shootout of machine learning methods (beyond what was done in Chapter 5) to determine which methods are more appropriate for different devices classes, as one method may work better on microcontrollers than say, diodes.

4. Smart IEMI Data Collection Using AI

Applied machine learning could also be used for data collection. Using a similar automated data collection apparatus in concert with an Artificial Intelligence algorithm that could determine the type and quantity of data to collect would be interesting. Conceivably, the AI would be able to minimize the amount of data required to train

classifiers optimally. This effort used a human-defined, constant interval type of approach. However, other approaches could be surveyed, compared, and then tested against the best an AI could do.

5. Automated Free-Field Experiment Setup

The experimental apparatus built for this dissertation only supports direct power injection type experimentation. However, the development of equivalent capability using more powerful, or more traditional sources such as magnetrons, in an anechoic chamber to perform experiments of similar scope but using free-field radiated RF signals would be excellent. This type of apparatus would require considerable work to accommodate necessary safety checks as there are genuine hazards associated with performing free-field radiation experiments against electronic devices. An obvious one would be fire detection and suppression. Many electronic devices contain lithium-ion batteries, which, when tampered with, are known to burst into flame. Moreover, microcontrollers can be easily reset and controlled for automated testing. However, it is unclear how “resets” could be performed on a drone or cellphone without human intervention. Consequently, this type of experimentation would likely require significant financial investment and support from instrumentation and control experts as well as potentially robotics to effectively “remove the man from the loop.”

In general, the sky is the limit when considering where this research could, or should, go in the future. Therefore, the main challenge is securing funding and researchers to accomplish it.

References

[1] F. Caselli and W. J. Coleman, "Cross-Country Technology Diffusion The Case of Computers," *AEA Papers and Proceedings on Technology, Education, and Economic Growth*, pp. 328-335, 2001.

[2] S. Mohanty, U. Choppali and E. Kougianos, "Everything You Want to Know About Smart Cities: The Internet of Things is the Backbone," *IEEE Consumer Electronics Magazine, Volume: 5, Issue: 3,* pp. 60-70, July 2016.

[3] M. Spencer and F. Ulaby, "Spectrum Issues Faced by Active Remote Sensing: Radio Frequency Interference and Operational Restrictions Technical Committees," *IEEE Geoscience and Remote Sensing Magazine, vol. 4, no. 1,* pp. 40-45, March 2016.

[4] C. L. Longmire, "On the Electromagnetic Pulse Produced by Nuclear Explosions," *IEEE Transactions on Electromagnetic Compatibility, vol. EMC-20, no. 1,* 1978.

[5] J. Benford, J. A. Swegle and E. Schamiloglu, High Power Microwaves, 3rd Edition, Boca Raton: CRC Press, 2016.

[6] ABC News Associated Press, "Energy Chief Cites Risk of Cyberattacks Crippling Power Grid," 6 June 2021. [Online]. Available: https://abcnews.go.com/Business/wireStory/energy-chief-cites-risk-cyberattacks-crippling-power-grid-78117554.

[7] K. Zetter, "Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid," *Wired Magazine,* pp. https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/, 3 March 2016.

[8] M. Webber, "The Texas Power Crisis Didn't Have to Happen," 15 June 2021. [Online]. Available: https://www.asme.org/topics-resources/content/the-texas-power-crisis-didn-t-have-to-happen.

[9] W. Radasky and E. Savage, "Intentional Electromagnetic Interference and its Impact on the U.S. Power Grid," Metatech Corp. on behalf of Oak Ridge National Laboratory, Goleta, CA, 2010.

[10] W. A. Radasky, C. E. Baum and M. W. Wik, "Introduction to the Special Issue on High-Power Electromagnetics (HPEM) and Intentional Electromagnetic Interference (IEMI)," *IEEE Transactions on Electromagnetic Compatibility, Vol. 46, No. 3,* pp. 314-321, 2004.

[11] D. Wunsch and R. Bell, "Modeling and Testing of Immunity of Computerized Equipment to Fast Electrical Transients," *IEEE Transactions on Electromagnetic Compatibility Vol. 41, No. 4,* pp. 452-459, 1999.

[12] R. Gardner, "Electromagnetic Terrorism: A Real Danger," in *Proceedings of the XIth Symposium on Electromagnetic Compatibility*, Wroclaw, Poland, June 1998.

[13] W. Radasky and M. Backstrom, "Brief Historical Review and Bibliogprhy for Intentional Electromagnetic Interference (IEMI)," in *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*, Beijing, China, 2014.

[14] M. Wik, W. Radasky and R. Gardner, "Intentional Electromagnetic Interference (EMI) -- What is the Threat and What Can We Do About It," in *Fifteenth International Wroclaw Symposium on EMC*, Wroclaw, 2000.

[15] T. J. Clarke, "Modeling of High Power Electromagnetic Effects on Digital Electronics," Air Force Office of Scientific Research, Arlington, 2013.

[16] D. M. Pozar, Microwave Engineering, 4th edition, Hoboken: John Wiley & Sons, Inc, 2012.

[17] M. K. Parai, B. Das and G. Das, "An Overview of Microcontroller Unit: From Proper Selection to Specific Application," *International Journal of Soft Computing and Engineering (IJSCE) Vol 2, Issue 6*, pp. 228-231, 2013.

[18] Ethridge, C. Dwayne, et al. Scientific Calculating Peripheral, Compcon Fall 79. Proceedings, 1979, pp. 467-471, DOI: 10.1109/CMPCON.1979.729154

[19] S. Soderbergh, Director, *Ocean's Eleven.* [Film]. 2001.

[20] S. Soderbergh, Director, *Ocean's Thirteen.* [Film]. 2007.

[21] International Electrotechnical Commission, "Measurements of Electromagnetic Immunity 150 kHz to 1 GHz—Part 4: Direct RF Power Injection Method, Standard IEC 62132-4," 21 February 2006. [Online]. Available: https://webstore.iec.ch/publication/6510#additionalinfo.

[22] V. Ceperic and A. Baric, "Modelling of Electromagnetic Immunity of Integrated Circuits by Artifical Neural Networks," in *20th International Zurich Symposium on Electromagnetic Compatibility*, Zurich, 2009.

[23] R. E. Richardson, V. G. Puglielli and R. A. Amadori, "Microwave Interference Effect in Bipolar Transistors," *IEEE Transactions on Electromagnetic Compatibility, Vol. EMC-17,* pp. 216-219, 1975.

[24] R. Vick and E. Habiger, "The Dependence of the Immunity of Digital Equipment on the Hardware and Software Structure," *Proceedings of International Symposium on Electromagnetic Compatibility,* pp. 383-386, 1997.

[25] D. Guillette and T. Clarke, "IEMI on a Microcontroller: Review of Upset Variation with

Respect to Chip Instance," Air Force Research Laboratory, Kirtland AFB, NM, 2019.

[26] D. Guillette, T. Clarke, and C. Christodoulou, "IEMI Microcontroller Effects: An Overview of Recent Results," in *AMEREM*, Oxnard, CA, 2018.

[27] W. Radasky, C. Baum and M. Wik, "Introduction to the Special Issue on High-Power Electromagnetics (HPEM) and Intentional Electromagnetic Interference (IEMI)," *IEEE Transactions on Electromagnetic Compatibility, vol. 46, no. 3,* pp. 314-321, 2004.

[28] R. Bilalic, "Dissertation - A Novel Application of Machine Learning for Prediction of Upset in Microcontrollers," University of New Mexico, Albuquerque, 2017.

[29] S. Marsland, Machine Learning: An Algorithm Perspective, Second Edition, Boca Raton: Chapman & Hall/ CRC Publishing, 2015.

[30] R. E. Richardson, "Modeling of Low-Level Rectification RFI in Bipolar Circuitry," *IEEE Transactions on Electromagnetic Compatibility, Vol. 21,* pp. 307-311, 1979.

[31] W. A. Radasky, C. E. Baum and M. W. Wik, "Introduction to the Special Issue on High Power Electromagnetics (HPEM) and Intentional Electromagnetic Interference (IEMI)," *IEEE Transactions on Electromagnetic Compatibility, Vol. 46, No. 3,* pp. 322-328, 2004.

[32] J. A. Swegle, J. Benford and E. Schamiloglu, High Power Microwaves, 3rd Edition., Boca Raton: CRC Press, 2016.

[33] M. G. Backstrom and K. G. Lovstrand, "Susceptibility of Electronic Systems to High-Power Microwaves: Summary of Test Experience," *IEEE Transactions of Electromagnetic Compatibility, Vol 46, No.3,* pp. 396-403, 2004.

[34] D. Giri and F. Tesche, "Classification of Intentional Electromagnetic," *IEEE TRANSACTIONS ON ELECTROMAGNETIC COMPATIBILITY, VOL. 46, NO. 3,* pp. 322-328, 2004.

[35] M. Camp, H. Gerth, H. Garbe, and H. Haase, "Predicting the Breakdown Behavior of Microcontrollers Under EMP/UWB Impact Using a Statistical Analysis," *IEEE Transactions on Electromagnetic Compatibility vol. 46, no. 3,* pp. 368-379, 2004.

[36] Georgiopoulos and Christodoulou, Neural Network Applications in Electromagnetics, Artech House, 2001.

[37] S. Caorsi and G. Cevini, "A Neural Network Approach for Electromagnetic Diagnostic Application," *PIERS Online,* vol. Vol. 2, no. No. 4, pp. 385-389, 2006.

[38] D. Micu, L. Czumbil, G. Christoforidis, and A. Ceclan, "Layer Recurrent Neural Network Solution for an Electromagnetic Interference Problem," *IEEE Transactions on*

Electromagnetics, vol. 47, no. 5, pp. 1410-1413, 2011.

[39] X. Li, Y. Yu, Q. Wang, and Y. Li, "Prediction of Electromagnetic Compatibility Programs Based on Artificial Neural Networks," in *2008 World Automation Congress*, Waikoloa, 2008.

[40] G. Oliveri, P. Rocca, and A. Massa, "SVM for Electromagnetics: State-of-art, Potentialities, and Trends," *IEEE Antennas and Propagation Society, AP-S International Symposium (Digest)*, pp. 7-8, 2012.

[41] Y. Wu, Z. Tang, B. Zhang, and Y. Xu, "Using Support Vector Machine Regression for Measuring Electromagnetic Parameters of Magnetic Materials," in *International Symposium on Microwave, Antenna, Propagation, and EMC Technologies for Wireless Communication*, Chengdu, 2007.

[42] J. Villain, V. Deniau, A. Fleury, E. Simon, C. Gransart and R. Kousri, "EM Monitoring and Classification of IEMI and Protocol-Based Attacks on IEEE 802.11n Communication Networks," *IEEE Transactions on Electromagnetic Compatibility Vol. 61 No.6*, vol. 61, no. 6, pp. 1771-1781, 2019.

[43] V. Ceperic, G. Gielen and A. Baric, "Black-box Modeling of Conducted Electromagnetic Immunity by Support Vector Machines," in *IEEE International Symposium on Electromagnetic Compatibility*, Pittsburgh, 2012.

[44] V. Devabhaktuni, C. Bunting, D. Green, D. Kvale, L. Mareddy, and V. Rajamani, "A New ANN-based Modeling Approach for Rapid EMI/EMC Analysis of PCB and Shielding Enclosures," *IEEE Transactions on Electromagnetic Compatibility*, vol. 55, no. 2, pp. 385-394, 2013.

[45] I. Chahine, M. Kadi, E. Gaboriaud, A. Louis and B. Mazari, "Using Neural Networks for Predicting the Integrated Circuits Susceptibility to Conducted Electromagnetic Disturbances," in *18th International Zurich Symposium on Electromagnetic Compatibility*, Zurich, 2007.

[46] R. Bilalic, "A Novel Application of Machine Learning Methods to Model Microcontroller Upset Due to Intentional Electromagnetic Interference," University of New Mexico: School of Engineering, Albuquerque, 2017.

[47] J. G. M. R. D. M. Muhammad Ali Mazidi, The 8051 Microcontroller, and Embedded Systems Using Assembly and C Second Edition, Columbus: Pearson Prentice Hall, 2006.

[48] Microchip Technology Inc., "PIC18 Datasheet," Microchip Technology Inc., Chandler, AZ, 2017.

[49] E. Abbott, Flatland: A Romance of Many Dimensions., New York: Dover Publications, 1838.

[50] P. Flach, Machine Learning: The Art and Science of Algorithms That Make Sense of Data., Cambridge: Cambridge University Press, 2012.

[51] J. Adair, The Art of Creative Thinking: How to be Innovative And Develop Great Ideas, London: Kogan Page Publishers, 2009.

[52] N. Matloff, Statistical Regression and Classification: From Linear Models to Machine Learning, Boca Raton: Chapman and Hall/CRC, 2017.

[53] e. a. Michael W. Berry, Supervised and Unsupervised Learning for Data Science, New York City: Springer Publishing, 2020.

[54] J. Q. A. Benjamin, Proofs that Really Count: The Art of Combinatorial Proof, Washington D.C.: Mathematical Associate of America, 2003.

[55] R. Kohavi, "A Study of Cross-Validation and BootStrap For Accuracy Estimation and Model Selection," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, San Mateo, CA, 1995.

[56] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, 1995.

[57] W. Liu, P. Pokharel and J. Principe, "The Kernel Least-Mean-Square Algorithm," *IEEE Transactions on Signal Processing,* vol. 56, pp. 543-554, 2008.

[58] M. S. A. K. V. K. P. Tan, Introduction to Data Mining, New York City: Pearson, 2018.

[59] L. Breiman, J. Freidman, R. Olshen and C. Stone, Classification and Regression Trees, Boca Raton, FL: Chapman & Hall, 1984.

[60] S. Doblias, "CART: Classification and Regression Trees for Clean but Powerful Models," 2021 30 Jan. [Online]. Available: https://towardsdatascience.com/cart-classification-and-regression-trees-for-clean-but-powerful-models-cc89e60b7a85. [Accessed 18 Oct 2021].

[61] P. H. T. Cover, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, 1967.

[62] V. M. S. Raschka, Python Machine Learning 3rd Edition, Birmingham: Packt Publishing Ltd., 2019.

[63] J. Ballantine and A. Jerbert, "Distance from a line or plane to a point," *American Mathematical Monthly*, vol. 59, no. 4, pp. 242-243, 1954.

[64] A. Bradley, "The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms," *Pattern Recognition,* vol. 30, no. 7, p. 1145–1159, 1997.

[65] Industry Council on ESD Target Levels, "White Paper 4 - Understanding Electrical Overstress - EOS," 25 August 2016. [Online]. Available: https://www.esdindustrycouncil.org/ic/en/documents/40-white-paper-4-elecrical-overstress. [Accessed 12 January 2022].

[66] Atmel Corporation, "8-bit Microcontroller with 2/4-Kbyte Flash AT89LP20/40/52 Datasheet," Atmel Corporation, San Jose, CA, 2009.

[67] All About Circuits, "Edge Trigger Latches," EE Tech, Boise, ID, 2021.

[68] M. Martinez-Ramon and C. Christodoulou, Support Vector Machines for Antenna Array Processing and Electromagnetics, Williston, VT: Morgan and Claypool Publishers, 2006.

Appendix

A collection of MATLAB scripts and codes used in this dissertation can be obtained upon request from the author. Below are a few select codes which have specific relevance to interested parties.

1. Complete Code for MCU4 PIC DUT

```
;*******************************************************************************
;                                                                              *
;   This file is a basic code template for code generation on the              *
;   PIC18F26K20. This file contains the basic code building blocks to build    *
;   upon.                                                                      *
;                                                                              *
;   Refer to the MPASM User's Guide for additional information on features     *
;   of the assembler.                                                          *
;                                                                              *
;   Refer to the respective data sheet for additional information on the       *
;   instruction set.                                                           *
;                                                                              *
;*******************************************************************************
;                                                                              *
;    Filename:        xxx.asm                                                  *
;    Date:                                                                     *
;    File Version:                                                             *
;    Author:                                                                   *
;    Company:                                                                  *
;                                                                              *
;*******************************************************************************
;                                                                              *
;    Files required: P18F26K20.INC                                             *
;                                                                              *
;*******************************************************************************
;                                                                              *
;    Features of the 18F26K20:                                                 *
;                                                                              *
; Self-programmable under software control                                     *
;    Extended watchdog timer programmable from 4 ms to 131 s                   *
;    Single supply 3 V ICSP via two pins                                       *
;    Operating range 1.8 to 3.6 V                                              *
;    Interrupt on high/low voltage detection (HLVD)                            *
;    Programmable brownout with software enable option                         *
;    Four crystal modes, up to 64 MHz                                          *
;    4X phase lock loop                                                        *
;    Programmable on-chip voltage reference                                    *
;    Dual analog comparators                                                   *
;    10 bit, 14 channel ADC                                                    *
;    Enhanced USART, supports RS-485, RS-232, and LIN 2.0                      *
;    MSSP module supporting SPI, and I2C with M/S modes with address mask      *
;    Programmable slew rate                                                    *
;                                                                              *
;*******************************************************************************
;                                                                              *
;    Notes:                                                                    *
;                                                                              *
;                                                                              *
;*******************************************************************************
```

```
;                                                                       *
;   Revision History:                                                   *
;                                                                       *
;***********************************************************************

; PIC18F26K42 Configuration Bit Settings

; Assembly source line config statements

#include "p18f26k42.inc"

; CONFIG1L
  CONFIG  FEXTOSC = ECM         ; External Oscillator Selection (XT (crystal oscillator) above 100 kHz, below 8
MHz; PFM set to medium power)
  CONFIG  RSTOSC = EXTOSC  ; Reset Oscillator Selection (EXTOSC operating per FEXTOSC bits)
; CONFIG1H
  CONFIG  CLKOUTEN = OFF       ; Clock out Enable bit (CLKOUT function is disabled)
  CONFIG  PR1WAY = ON          ; PRLOCKED One-Way Set Enable bit (PRLOCK bit can be cleared and set only
once)
  CONFIG  CSWEN = ON         ; Clock Switch Enable bit (Writing to NOSC and NDIV is allowed)
  CONFIG  FCMEN = ON           ; Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor enabled)

; CONFIG2L
  CONFIG  MCLRE = EXTMCLR       ; MCLR Enable bit (If LVP = 0, MCLR pin is MCLR; If LVP = 1, RE3 pin
function is MCLR )
  CONFIG  PWRTS = PWRT_OFF      ; Power-up timer selection bits (PWRT is disabled)
  CONFIG  MVECEN = ON          ; Multi-vector enable bit (Multi-vector enabled, Vector table used for interrupts)
  CONFIG  IVT1WAY = ON         ; IVTLOCK bit One-way set enable bit (IVTLOCK bit can be cleared and set only
once)
  CONFIG  LPBOREN = OFF        ; Low Power BOR Enable bit (ULPBOR disabled)
  CONFIG  BOREN = SBORDIS       ; Brown-out Reset Enable bits (Brown-out Reset enabled , SBOREN bit is
ignored)

; CONFIG2H
  CONFIG  BORV = VBOR_2P45      ; Brown-out Reset Voltage Selection bits (Brown-out Reset Voltage (VBOR) set
to 2.45V)
  CONFIG  ZCD = OFF            ; ZCD Disable bit (ZCD disabled. ZCD can be enabled by setting the ZCDSEN bit of
ZCDCON)
  CONFIG  PPS1WAY = ON         ; PPSLOCK bit One-Way Set Enable bit (PPSLOCK bit can be cleared and set only
once; PPS registers remain locked after one clear/set cycle)
  CONFIG  STVREN = ON          ; Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
  CONFIG  DEBUG = OFF          ; Debugger Enable bit (Background debugger disabled)
  CONFIG  XINST = OFF          ; Extended Instruction Set Enable bit (Extended Instruction Set and Indexed
Addressing Mode disabled)

; CONFIG3L
  CONFIG  WDTCPS = WDTCPS_31    ; WDT Period selection bits (Divider ratio 1:65536; software control of
WDTPS)
  CONFIG  WDTE = OFF            ; WDT operating mode (WDT enabled regardless of sleep)

; CONFIG3H
  CONFIG  WDTCWS = WDTCWS_7     ; WDT Window Select bits (window always open (100%); software control;
keyed access not required)
  CONFIG  WDTCCS = SC          ; WDT input clock selector (Software Control)

; CONFIG4L
  CONFIG  BBSIZE = BBSIZE_512   ; Boot Block Size selection bits (Boot Block size is 512 words)
  CONFIG  BBEN = OFF           ; Boot Block enable bit (Boot block disabled)
  CONFIG  SAFEN = OFF           ; Storage Area Flash enable bit (SAF disabled)
  CONFIG  WRTAPP = OFF          ; Application Block write protection bit (Application Block not write protected)

; CONFIG4H
```

```
  CONFIG  WRTB = OFF          ; Configuration Register Write Protection bit (Configuration registers (300000-
30000Bh) not write-protected)
  CONFIG  WRTC = OFF          ; Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-
protected)
  CONFIG  WRTD = OFF          ; Data EEPROM Write Protection bit (Data EEPROM not write-protected)
  CONFIG  WRTSAF = OFF        ; SAF Write protection bit (SAF not Write Protected)
  CONFIG  LVP = ON            ; Low Voltage Programming Enable bit (Low voltage programming enabled.
MCLR/VPP pin function is MCLR. MCLRE configuration bit is ignored)

; CONFIG5L
  CONFIG  CP = OFF            ; PFM and Data EEPROM Code Protection bit (PFM and Data EEPROM code
protection disabled)

;-------------------------------------------------------------------------------
;  delay variables
;-------------------------------------------------------------------------------
;sGPIO    res 1 ; shadow copy of GPIO
COUNT1EQU 08H
COUNT2EQU 09H

;-------------------------------------------------------------------------------
; SET RC CALIBRATION
;-------------------------------------------------------------------------------
;ROCCAL             CODE     0xF0
;            res 1

;-------------------------------------------------------------------------------
; RESET VECTOR
;-------------------------------------------------------------------------------
;RES_VECT    CODE    0x0000          ; processor reset vector
;               GOTO    START           ; go to beginning of program

;-------------------------------------------------------------------------------
; MAIN PROGRAM
;-------------------------------------------------------------------------------
MAIN_PROG   CODE                    ; let linker place main program

Start
;**** Setup the port ****
            CLRF    LATA
            CLRF    TRISA
            CLRF    PORTA

Main
;****Start of the delay loop 1****
;Loop1 decfsz COUNT1,1 ;Subtract 1 from 255
;            goto Loop1 ;If COUNT is zero, carry on.
;            decfsz COUNT2,1 ;Subtract 1 from 255
;            goto Loop1 ;Go back to the start of our loop.
            ;This delay counts down from
            ;255 to zero, 255 times

;****Delay finished, now turn the LED off****
            movff PORTA,WREG
            addlw 01h ;Turn the LED off by first putting
            movwf LATA ;it into the w register and then on
            ;the port

;****Add another delay****
;Loop2 decfsz COUNT1,1 ;This second loop keeps the
;            goto Loop2 ;LED turned off long enough for
;            decfsz COUNT2,1 ;us to see it turned off
```

```
;               goto Loop2 ;


;****Now go back to the start of the program
                goto Main ;go back to Start and turn LED
                ;on again
;****End of the program****
   end ;Needed by some compilers,
                ;and also just in case we miss
                ;the goto instruction.
```

2. User Input File Generator Code

```
%% Housekeeping
set(groot,'ShowHiddenHandles','on')
c = get(groot,'Children');
delete(c)

clear all
clc
disp('************')
disp('Script Start')
disp('************')

%% User inputs to experiment
num_samples = 1;
freq = [20e6,50e6,100e6,200e6,400e6,800e6];
pulse_width = [25e-9,50e-9,100e-9,200e-9,400e-9,800e-9,1000e-9];
amp = [-27,-24,-21,-18,-15,-12,-9,-6,-3];
injection_time = [linspace(6e-6,8e-6,80)];
filename = sprintf('ExperimentSetupSequences-2NOV2020');

aref = 'T0';
bref = 'T0';
cref = 'T0';
dref = 'T0';
aref2 = 'A';
bref2 = 'C';
cref2 = 'E';
dref2 = 'G';

%% Fake Sequences
disp('Building Sequences')
gg = 2;
num_seq = size(freq,2)*size(pulse_width,2)*size(amp,2)*size(injection_time,2)*num_samples;
numfreq = size(freq,2);
numpw = size(pulse_width,2);
numamp = size(amp,2);
numit = size(injection_time,2);
file_details = sprintf('samp%dfreq%dpulwid%damp%dtime%d',num_samples,numfreq,numpw,numamp,numit);

row(1,:) = {'Sequence #','# Shots','Frequency Hz','Power','A Ref','A Time',...
    'B Ref','B Time','C Ref','C Time','D Ref','D Time','E Ref','E Time',...
    'F Ref','F Time','G Ref','G Time','H Ref','H Time'};

for s = 1: num_samples
fprintf('Seq. %d/%d\n',gg,num_seq);
for a = 1:size(freq,2)
for b = 1:size(pulse_width,2)
for c = 1:size(amp,2)
```

```
for d = 1:size(injection_time,2)
row(gg,:) = {gg-1;num_samples;freq(a);amp(c);...
    aref;injection_time(d);aref2;pulse_width(b);...
    bref;9.3e-5;bref2;2.0e-6;...
    cref;0;cref2;0;...
    dref;0;dref2;9.5e-5};
gg = gg+1
end
end
end
end
end
disp('> Complete')

%% Export
disp('Writing file: ExperimentFile-xxxx.xlsx')
file1 = sprintf('%s-%s.xlsx',filename,file_details);
writecell(row,file1);

disp('> Complete')

disp('************')
disp('Script End')
disp('************')
```

3. Upset Detection Code

```
for ivar = 1:1:size(TargetINFO(:,1),1)
    % CREATE TARGET VARIABLES
    TargetTime_Temp = eval(sprintf('Target%i',ivar)); % TIME LOCATION OF TARGET_ivar
    PinTemp0 = sprintf('Pin0_value_T%i)',ivar);
    PinTemp1 = sprintf('Pin1_value_T%i)',ivar);
    PinTemp2 = sprintf('Pin2_value_T%i)',ivar);
    TargetLocIndex_Temp = sprintf('TargetLocation%i',ivar); % INDEX ASSOCIATED WITH THE TIME
LOCATION TARGET_ivar

    % PULL OUT TARGET LOCATION VOLTAGES ON EACH PIN
    TargetLocIndex_Temp = round((TargetTime_Temp-CurrentFile(1,1))/(CurrentFile(2,1)-CurrentFile(1,1))); %
DEFINES INDEX AT TARGET TIME LOCATION
    PinTemp0 = CurrentFile(TargetLocIndex_Temp,3); % Channel 2
    PinTemp1 = CurrentFile(TargetLocIndex_Temp,4); % Channel 3
    PinTemp2 = CurrentFile(TargetLocIndex_Temp,5); % Channel 4

    % DEFINE THRESHOLD FOR EACH PIN
    V_Threshold_P0 = (max(CurrentFile(:,3))-min(CurrentFile(:,3)))/3; % this voltage value is the point used to
determine whether the pin value is 1 or 0
    V_Threshold_P1 = (max(CurrentFile(:,4))-min(CurrentFile(:,4)))/3; % this voltage value is the point used to
determine whether the pin value is 1 or 0
    V_Threshold_P2 = (max(CurrentFile(:,5))-min(CurrentFile(:,5)))/3; % this voltage value is the point used to
determine whether the pin value is 1 or 0

    % DETERMINE BINARY VALUE FROM VOLTAGE SIGNAL
    % If Actual Pin0 value greater than threshold set to 1 otherwise set to 0
    if (PinTemp0 >= V_Threshold_P0)
        eval(sprintf('Bit0_value_T%i = 1;',ivar));
    else
        eval(sprintf('Bit0_value_T%i = 0;',ivar));
    end
    % If Actual Pin1 value greater than threshold set to 1 otherwise set to 0
```

```
        if (PinTemp1 >= V_Threshold_P1)
            eval(sprintf('Bit1_value_T%i = 1;',ivar));
        else
            eval(sprintf('Bit1_value_T%i = 0;',ivar));
        end
        % If Actual Pin2 value greater than threshold set to 1 otherwise set to 0
        if (PinTemp2 >= V_Threshold_P2)
            eval(sprintf('Bit2_value_T%i = 1;',ivar));
        else
            eval(sprintf('Bit2_value_T%i = 0;',ivar));
        end

        % EVALUATE ACTUAL_BINARY VALUE FOR TARGET AND SAVE FOR LATER
        eval(sprintf('Actual_binary_T%i =
    (Bit0_value_T%i*1)+(Bit1_value_T%i*2)+(Bit2_value_T%i*4);',ivar,ivar,ivar,ivar));
        ActBinTemp = eval(sprintf('Actual_binary_T%i',ivar));

        % DEFINE EXPECTED BINARY VALUE
        ExpBinTemp = eval(sprintf('Expected_binary_T%i',ivar));

        % DETERMINE IF CURRENT TARGET%i LOCATION IS UPSET OR NOT.
        if (ExpBinTemp == ActBinTemp)
            Upset_temp(ivar,1) = 0;
        else
            Upset_temp(ivar,1) = 1;
        end
    end % end of target location check

    % CHECK FOR UPSET: IF UPSET, IDENTIFY TYPE
    % IF, ONE OR MORE TARGET LOCATIONS ARE NOT CORRECT SET VALUE TO 1
    Expected_array = [Expected_binary_T1,Expected_binary_T2,Expected_binary_T3,...
        Expected_binary_T4,Expected_binary_T5,Expected_binary_T6,...
        Expected_binary_T7,Expected_binary_T8,Expected_binary_T9];
    Actual_array = [Actual_binary_T1,Actual_binary_T2,Actual_binary_T3,...
        Actual_binary_T4,Actual_binary_T5,Actual_binary_T6,...
        Actual_binary_T7,Actual_binary_T8,Actual_binary_T9];

    if(sum(Upset_temp) ~= 0)
        [Latch,Shift,Miscount] = SALVO_PROCESS_latchcheck(Expected_array,Actual_array);
    else
        Latch = 0;
        Shift = 0;
        Miscount = 0;
    end

    if sum(Upset_temp) ~= 0
        Upset = 1;
    else
        Upset = 0;
    end

    clear tempSum


function [Latch,Shift,Miscount] = SALVO_PROCESS_latchcheck(Expected_array,Actual_array)
    % INITIALIZE OUTPUT VARIABLES
    Latch = 0;
    Miscount = 0;
    Shift = 0;
    Shift_bit = 0;
```

```
    % DOUBLE CHCECK THAT A NON-UPSET Actual_array WAS NOT PASSED
    if (Actual_array == Expected_array)
        Latch = 0;
        Shift = 0;
        Miscount = 0;
        return
    end % END NON-UPSET CHECK

    % CHECK FOR SPECIAL 'BROKEN' CASES NOT COVERED BY THE BELOW CODE
    % Added: 4/17/2017
    if (Actual_array == [7,0,1,2,3,4,5,6,7]);
        Shift = 1;
        Latch = 0;
        Miscount = 0;
        return
    end

    % CHECK FOR LATCHED CASE
    if (size(find(Actual_array == Actual_array(1,9)),2) > 1)
        if (size(find(Actual_array == 0),2) == 2)
            Miscount = 1;
        else
            Latch = 1;
        end
    else
        Latch = 0;
    end % END IF LATCH

    % CHECK FOR SHIFT IF LATCH AND MISCOUNT EQUAL ZERO
    if ((Latch == 0) && (Miscount ~=1))
        input = abs(Expected_array - Actual_array); % TAKE DIFFERENCE OF EXPECTED AND ACTUAL ARRAY
        Shift_amount = 8 - input(size(input,2));
            if Shift_amount > 0
                Shift_bit = Shift_amount;
                ideal_shift_array = circshift(Expected_array,Shift_amount,2);
                ideal_shift_array(1:Shift_amount+1) = zeros(1,Shift_amount+1);
                Shift_true = ideal_shift_array - Actual_array;
                if(sum(Shift_true) == 0)
                    Shift = 1;
                    Miscount = 0;
                    Latch = 0;
                else
                    Miscount = 1;
                    Shift = 0;
                    Latch = 0;
                end
            end
    end % END IF SHIFT
end % END FUNCTION
```

4. Classifier Permutation Generator and Feature Selector Code

```
%% variable list
Features = {'Frequency';'PulseWidth';'InputPower';'MeasuredPower';...
    'InjectTime';'Energy';'Vmax';'Vmin';'Vrms';'Vp2p'};

%% make permutation column list of each by row.
List1 = nchoosek(Features,1);
List2 = nchoosek(Features,2);
```

```
List3 = nchoosek(Features,3);
List4 = nchoosek(Features,4);
List5 = nchoosek(Features,5);
List6 = nchoosek(Features,6);
List7 = nchoosek(Features,7);
List8 = nchoosek(Features,8);
List9 = nchoosek(Features,9);

Combos = [Features;List1;List2;List3;List4;List5]
```

5. Train KNN Classifiers Code

```
function [Classifier_Accuracy] = MakeKNNFiles(SelectedData,fileName,trainFeatures)

numFeatures = size(trainFeatures,2);
numCombinations = size(trainFeatures,1);

%- Train, Validate, Export classifier
for i = 1:numCombinations
    predictorNames = trainFeatures(i,:);
    modelName = sprintf('KNN_f%dc%d_m%d_%s',numFeatures,numCombinations,i,fileName);
    modelAccuracy = sprintf('KNN%d_acc',i);
    fprintf ('Train Classifier %s',modelName)
    [modelKNN,modelAccuracy]= trainKNNClassifierADV(SelectedData,predictorNames);
    disp('> Complete')
    Accuracy(i) = modelAccuracy;
    Classifier_Accuracy(i,:) = table({modelName},modelAccuracy,predictorNames);
end

%% Outputs
%- Excel document
fprintf('Writing file: Classifier_Trends_Features_%d_Combins_%d.xlsx\n',numFeatures,numCombinations)
fileA = sprintf('%s_Classifier_Trends_f%dc%d.xlsx',fileName,numFeatures,numCombinations);
writetable(Classifier_Accuracy,fileA);
disp('> Complete')

end

function [trainedClassifier, validationAccuracy] = trainKNNClassifierADV(trainingData,predictorNames)

inputTable = trainingData;

predictors = inputTable(:, predictorNames); %so the problem is that this call wants

response = inputTable.Upset;

if size(predictorNames,2) == 1
    isCategoricalPredictor = [false];
elseif size(predictorNames,2) == 2
    isCategoricalPredictor = [false, false];
elseif size(predictorNames,2) == 3
    isCategoricalPredictor = [false, false, false];
elseif size(predictorNames,2) == 4
    isCategoricalPredictor = [false, false, false, false];
elseif size(predictorNames,2) == 5
    isCategoricalPredictor = [false, false, false, false, false];
elseif size(predictorNames,2) == 6
    isCategoricalPredictor = [false, false, false, false, false, false];
elseif size(predictorNames,2) == 7
```

```
    isCategoricalPredictor = [false, false, false, false, false, false, false];
elseif size(predictorNames,2) == 8
    isCategoricalPredictor = [false, false, false, false, false, false, false, false];
elseif size(predictorNames,2) == 9
    isCategoricalPredictor = [false, false, false, false, false, false, false, false, false];
elseif size(predictorNames,2) == 10
    isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false];
end

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Cityblock', ...
    'Exponent', [], ...
    'NumNeighbors', 10, ...
    'DistanceWeight', 'SquaredInverse', ...
    'Standardize', true, ...
    'ClassNames', [0; 1]);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x) knnPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = predictorNames;

trainedClassifier.ClassificationKNN = classificationKNN;
trainedClassifier.About = 'This struct is a trained model exported from Classification Learner R2019b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n  yfit = c.predictFcn(T)
\nreplacing "c" with the name of the variable that is this struct, e.g. "trainedModel". \n \nThe table, T, must contain the
variables returned by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must match the original
training data. \nAdditional variables are ignored. \n \nFor more information, see <a
href="matlab:helpview(fullfile(docroot, "stats", "stats.map"), "appclassification_exportmodeltoworkspace")">How to
predict using an exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
% inputTable = trainingData;

% Set up holdout validation
cvp = cvpartition(response, 'Holdout', 0.25);
trainingPredictors = predictors(cvp.training, :);
trainingResponse = response(cvp.training, :);
trainingIsCategoricalPredictor = isCategoricalPredictor;

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationKNN = fitcknn(...
    trainingPredictors, ...
    trainingResponse, ...
    'Distance', 'Cityblock', ...
    'Exponent', [], ...
    'NumNeighbors', 10, ...
    'DistanceWeight', 'SquaredInverse', ...
    'Standardize', true, ...
    'ClassNames', [0; 1]);
```

```
% Create the result struct with predict function
knnPredictFcn = @(x) predict(classificationKNN, x);
validationPredictFcn = @(x) knnPredictFcn(x);

% Add additional fields to the result struct

% Compute validation predictions
validationPredictors = predictors(cvp.test, :);
validationResponse = response(cvp.test, :);
[validationPredictions, validationScores] = validationPredictFcn(validationPredictors);

% Compute validation accuracy
correctPredictions = (validationPredictions == validationResponse);
isMissing = isnan(validationResponse);
correctPredictions = correctPredictions(~isMissing);
validationAccuracy = sum(correctPredictions)/length(correctPredictions);
```

6. Repeat Train and Cross Prediction Code

```
% Train Weighted KNN sq inv, city block.

%% Housekeeping
close all;
clear all;
clc;
disp('...script start...')

%% Inputs
%- Open data
pathname1 = 'C:\Users\xxxx\Documents\PhD\Ready Data Files';
addpath(pathname1)

disp ('Read in Dataset - MCU1')
file1 = 'ReadyData-MCU1.xlsx';
% Data_MCU1 = readmatrix(file1); %cant use cell as index
% Data_MCU1 = readcell(file1); % cant use cell as index.
Data_MCU1 = readtable(file1); % cant use variable names
disp('> Complete')

disp ('Read in Dataset - MCU2')
file2 = 'ReadyData-MCU2.xlsx';
Data_MCU2 = readtable(file2); % cant use variable names
disp('> Complete')

disp ('Read in Dataset - MCU3')
file3 = 'ReadyData-MCU3.xlsx';
Data_MCU3 = readtable(file3); % cant use variable names
disp('> Complete')

disp ('Read in Dataset - MCU4')
file4 = 'ReadyData-MCU4.xlsx';
Data_MCU4 = readtable(file4); % cant use variable names
disp('> Complete')

disp ('Read in Dataset - MCU23')
file23 = 'ReadyData-MCU23.xlsx';
Data_MCU23 = readtable(file23); % cant use variable names
disp('> Complete')
```

```
disp ('Read in Dataset - MCU234')
file234 = 'ReadyData-MCU234.xlsx';
Data_MCU234 = readtable(file234); % cant use variable names
disp('> Complete')

%% Train Classifier
%- Set classifier inputs
trainFeatures = {'FreqHz','InjectTime','PulseWidth','Vrms','Vmax','Emax'};
numFeatures = size(trainFeatures,2);
numCombinations = size(trainFeatures,1);

%- Train kNN with data1
% -- repeat n times to identify stability
k = 1;
nmax = 125;
for n = 1:nmax
   for i = 1:numCombinations
      predictorNames = trainFeatures(i,:);
      modelName = sprintf('KNN-f%dc%d-m%d',numFeatures,numCombinations,k);
      modelAccuracy = sprintf('KNN%d_acc',k);
      [modelKNN,modelAccuracy]= trainKNNClassifierADV(Data_MCU1,predictorNames);
      Accuracy(k) = modelAccuracy;
      Models(k) = modelKNN;
      Classifier_Accuracy(k,:) = table({modelName},modelAccuracy,predictorNames);
      min_acc = 100*min(Accuracy);
      max_acc = 100*max(Accuracy);
      mean_acc = 100*mean(Accuracy);
      median_acc = 100*median(Accuracy);
      std_acc = 100*std(Accuracy);
      fprintf('Train Classifier %s > Progress: %d/%d > Mean: %2.4f > Med:
%2.4f\n',modelName,k,nmax,mean_acc,median_acc)
      k = k+1;
   end
end

figure(1)
plot(1:nmax,Accuracy)
xlabel('n')
ylabel('Classifier Accuracy')
title(sprintf('%s Classifier Repeatability',modelName))

%% Select Median classifier
Accuracy = 100*Accuracy;
for q = 1:125
   if Accuracy(q) == median_acc
      bestkNN = Models(q);
      break
   end
end

%% Predict
%- Cross predict with data1
inputTable1 = Data_MCU1(:,1:10); % column 11 is the upset data
predictors = inputTable1(:, trainFeatures); %so the problem is that this call wants inputTable(:,#int)
response1 = Data_MCU1{:,11};
k2 = 1;
jmin = 1;
jmax = 30240;
correct_predict1 = 0;
for j = jmin:jmax
```

```
    NewData = inputTable1(j,:);
    predicted_label1(k2) = bestkNN.predictFcn(NewData);
    actual_label(k2) = response1(j);
    fprintf('Prediction Progress: %d/%d\n',j,(jmax-jmin)+1)
    if predicted_label1(k2) == actual_label(k2)
        correct_predict1 = correct_predict1 + 1;
    end
    k2 =k2+1;
end
%- determine number of accurate predictions
fprintf('Predicted: %d of %d\n',correct_predict1,jmax)
prediction_accuracy1 = 100*(correct_predict1/jmax);
fprintf('Prediction MCU1 Accuracy: %f percent\n',prediction_accuracy1)

%- Cross predict with data2
inputTable2 = Data_MCU2(:,1:10); % column 11 is the upset data
predictors = inputTable2(:, trainFeatures); %so the problem is that this call wants inputTable(:,#int)
response = Data_MCU2{:,11};
k2 = 1;
jmin = 1;
jmax = 30240;
correct_predict2 = 0;
for j = jmin:jmax
    NewData = inputTable2(j,:);
    predicted_label2(k2) = bestkNN.predictFcn(NewData);
    actual_label(k2) = response(j);
    fprintf('Prediction Progress: %d/%d\n',j,(jmax-jmin)+1)
    if predicted_label2(k2) == actual_label(k2)
        correct_predict2 = correct_predict2 + 1;
    end
    k2 =k2+1;
end
%- determine number of accurate predictions
fprintf('Predicted: %d of %d\n',correct_predict2,jmax)
prediction_accuracy2 = 100*(correct_predict2/jmax);
fprintf('Prediction MCU2 Accuracy: %f percent\n',prediction_accuracy2)

%- Cross predict with data3
inputTable3 = Data_MCU3(:,1:10); % column 11 is the upset data
predictors = inputTable3(:, trainFeatures); %so the problem is that this call wants inputTable(:,#int)
response3 = Data_MCU3{:,11};
k2 = 1;
jmin = 1;
jmax = 30240;
correct_predict3 = 0;
for j = jmin:jmax
    NewData = inputTable3(j,:);
    predicted_label3(k2) = bestkNN.predictFcn(NewData);
    actual_label(k2) = response3(j);
    fprintf('Prediction Progress: %d/%d\n',j,(jmax-jmin)+1)
    if predicted_label3(k2) == actual_label(k2)
        correct_predict3 = correct_predict3 + 1;
    end
    k2 =k2+1;
end
%- determine number of accurate predictions
fprintf('Predicted: %d of %d\n',correct_predict3,jmax)
prediction_accuracy3 = 100*(correct_predict3/jmax);
fprintf('Prediction MCU3 Accuracy: %f percent\n',prediction_accuracy3)

%- Cross predict with data4
```

```
inputTable4 = Data_MCU4(:,1:10); % column 11 is the upset data
predictors = inputTable4(:, trainFeatures); %so the problem is that this call wants inputTable(:,#int)
response4 = Data_MCU4{:,11};
k2 = 1;
jmin = 1;
jmax = 30240;
correct_predict4 = 0;
for j = jmin:jmax
   NewData = inputTable4(j,:);
   predicted_label4(k2) = bestkNN.predictFcn(NewData);
   actual_label(k2) = response4(j);
   fprintf('Prediction Progress: %d/%d\n',j,(jmax-jmin)+1)
   if predicted_label4(k2) == actual_label(k2)
      correct_predict4 = correct_predict4 + 1;
   end
   k2 =k2+1;
end
%- determine number of accurate predictions
fprintf('Predicted: %d of %d\n',correct_predict4,jmax)
prediction_accuracy4 = 100*(correct_predict4/jmax);
fprintf('Prediction MCU4 Accuracy: %f percent\n',prediction_accuracy4)

%- Cross predict with data23
inputTable23 = Data_MCU23(:,1:10); % column 11 is the upset data
predictors = inputTable23(:, trainFeatures); %so the problem is that this call wants inputTable(:,#int)
response23 = Data_MCU23{:,11};
k2 = 1;
jmin = 1;
jmax = 60480;
correct_predict23 = 0;
for j = jmin:jmax
   NewData = inputTable23(j,:);
   predicted_label23(k2) = bestkNN.predictFcn(NewData);
   actual_label(k2) = response23(j);
   fprintf('Prediction Progress: %d/%d\n',j,(jmax-jmin)+1)
   if predicted_label23(k2) == actual_label(k2)
      correct_predict23 = correct_predict23 + 1;
   end
   k2 =k2+1;
end
%- determine number of accurate predictions
fprintf('Predicted: %d of %d\n',correct_predict23,jmax)
prediction_accuracy23 = 100*(correct_predict23/jmax);
fprintf('Prediction MCU23 Accuracy: %f percent\n',prediction_accuracy23)

%- Cross predict with data234
inputTable234 = Data_MCU234(:,1:10); % column 11 is the upset data
predictors = inputTable234(:, trainFeatures); %so the problem is that this call wants inputTable(:,#int)
response234 = Data_MCU234{:,11};
k2 = 1;
jmin = 1;
jmax = 90720;
correct_predict234 = 0;
for j = jmin:jmax
   NewData = inputTable234(j,:);
   predicted_label234(k2) = bestkNN.predictFcn(NewData);
   actual_label(k2) = response234(j);
   fprintf('Prediction Progress: %d/%d\n',j,(jmax-jmin)+1)
   if predicted_label234(k2) == actual_label(k2)
      correct_predict234 = correct_predict234 + 1;
   end
   k2 =k2+1;
```

```
end
%- determine number of accurate predictions
fprintf('Predicted: %d of %d\n',correct_predict234,jmax)
prediction_accuracy234 = 100*(correct_predict234/jmax);
fprintf('Prediction MCU23 Accuracy: %f percent\n',prediction_accuracy234)

disp ('... script end...')
```