CYBER-PHYSICAL SYSTEM INTRUSION: A CASE STUDY OF AUTOMOBILE IDENTIFICATION VULNERABILITIES AND AUTOMATED APPROACHES FOR INTRUSION DETECTION

THESIS

David R. Crow, Second Lieutenant, USAF

AFIT-ENG-MS-20-M-012

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-20-M-012

CYBER-PHYSICAL SYSTEM INTRUSION: A CASE STUDY OF AUTOMOBILE IDENTIFICATION VULNERABILITIES AND AUTOMATED APPROACHES FOR INTRUSION DETECTION

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

David R. Crow, B.S.
Second Lieutenant, USAF

March 2020

AFIT-ENG-MS-20-M-012

CYBER-PHYSICAL SYSTEM INTRUSION: A CASE STUDY OF AUTOMOBILE IDENTIFICATION VULNERABILITIES AND AUTOMATED APPROACHES FOR INTRUSION DETECTION

THESIS

David R. Crow, B.S.
Second Lieutenant, USAF

Committee Membership:

Scott R. Graham, Ph.D.
Chair

Brett J. Borghetti, Ph.D.
Member

Lt. Col. Patrick J. Sweeney, Ph.D.
Member

AFIT-ENG-MS-20-M-012

Abstract

Today's vehicle manufacturers do not tend to publish proprietary packet formats for the controller area network (CAN), a network protocol regularly used in automobiles and manufacturing. This is a form of security through obscurity—it makes reverse engineering efforts more difficult for would-be intruders—but obfuscating the CAN data in this way does not adequately hide the vehicle's unique signature, even if these data are unprocessed or limited in scope. To prove this, we train two distinct deep learning models on data from 11 different vehicles. Our results clearly indicate that one can determine which vehicle generated a given sample of CAN data. This erodes consumer safety: a sophisticated attacker who establishes a presence on an unknown vehicle can use similar techniques to identify the vehicle and better format attacks.

To protect critical cyber-physical systems (CPSs) against attacks like those enabled by this CAN vulnerability, system administrators often develop and employ intrusion detection systems (IDSs). Before developing an IDS, one requires an understanding of the behavior of the CPS and of the causality of its constituent parts. Such an understanding allows one to characterize normal behavior and, in turn, identify and report anomalous behavior. This research explores two different time series analysis techniques, Granger causality and empirical dynamic modeling (EDM), which may contribute to this understanding of a system. Our findings indicate that Granger causality is not a suitable approach to IDS development but that EDM may enable the understanding of a system required of an IDS architect. We thus encourage further research into EDM applications to IDSs for CPSs.

AFIT-ENG-MS-20-M-012

This work is dedicated to my wife and to my dogs.

I am grateful for their love and companionship.

Acknowledgements

I would like to thank Dr. Scott Graham for his continuous support and expert advice during the research process. He allowed this thesis to be my own work while providing direction whenever he thought I needed it. I would also like to acknowledge Dr. Brett Borghetti and Lt. Col. Patrick Sweeney for their helpful lessons and thorough reviews of the work I produced. Finally, I must express my profound gratitude to my wife for providing me with unfailing support and unending desserts. This accomplishment would not have been possible without her.

David R. Crow

Table of Contents

Page

List of Figures

List of Tables

List of Abbreviations

AFRL	Air Force Research Lab
ArbID	arbitration ID
ARIMA	autoregressive integrated moving average
AVAS	Avionics Vulnerability Assessment System
CAN	controller area network
CAN FD	flexible data rate
CCM	convergent cross mapping
CNN	convolutional neural network
CPS	cyber-physical system
CSV	comma-separated values
ECU	electronic control unit
EDM	empirical dynamic modeling
IDS	intrusion detection system
MLP	multilayer perceptron
NIST	National Institute of Science and Technology
OBD	on-board diagnostics
OEM	original equipment manufacturer
RMSE	root-mean-square error
RPM	revolutions per minute
S-map	sequential locally weighted global linear map
SNN	Siamese neural network

CYBER-PHYSICAL SYSTEM INTRUSION:

A CASE STUDY OF AUTOMOBILE IDENTIFICATION VULNERABILITIES

AND AUTOMATED APPROACHES FOR INTRUSION DETECTION

I. Introduction

1.1 Motivation

The controller area network (CAN), a cyber-physical system (CPS) which connects a large number of the devices in a modern vehicle (including vital systems like the brakes, steering wheel, and transmission), is vulnerable in several ways. In the extreme, it is vulnerable to cyberattacks capable of altering, preventing, or otherwise modifying the operator's desired behavior. This research presents a more obscure vulnerability: the packets that broadcast on a vehicle's CAN bus can uniquely identify the vehicle. This means that an attacker can construct a database of known CAN packet formats and, using this database and one of the tools presented in Chapter III, identify a new vehicle. This allows the attacker to strengthen the attack and thus places the operator and passengers at greater risk.

We illustrate this vulnerability by building, tuning, and evaluating two deep learning models. We train the models by giving as input CAN data samples and each sample's generating vehicle, and we test these models by giving new samples as input and comparing the predicted vehicle to the actual vehicle. This research seeks to determine which vehicle generated each test sample. Data comes from one primary source and consists of 230 megabytes of raw CAN data captured from 11 distinct vehicles. By formatting and partitioning the available data, we generate two disparate datasets. The first is composed of all available 1,024-byte CAN samples; the second contains samples from the first such that the classes are evenly represented. Each sample's label is the ID of the vehicle that generated it.

We then feed these samples into the deep learning models. The first, a multilayer perceptron (MLP), is simple enough that an attacker only somewhat familiar with deep learning tools can easily implement it and use it to classify CAN samples with some level of success. The second, a more complex convolutional neural network (CNN), achieves much better classification performance than does the MLP. Overall, results indicate that one can use either the MLP or the CNN to determine which vehicle generated a given CAN data sample. This compromises operator and passenger privacy. It also risks safety because it gives bad cyber actors another tool to correctly structure malicious CAN packets.

Intrusion detection systems (IDSs) are a common method to defend against CPS attacks, including those enabled by the aforementioned CAN vulnerability. These systems monitor computer networks such as CAN and report malicious activity to system administrators. In the CPS domain, an IDS can prevent attackers from successfully modifying or misrepresenting physical processes. Consider an automobile's CPSs. If an attacker intends to, say, cause the driver to speed and thus receive a speeding ticket, the attacker may choose to inject packets detailing a lower speed, which would in turn cause the speedometer to display incorrect information. In this case, an effective IDS will notice that the data for speed does not conform to the expected behavior indicated by the data for the related physical processes (e.g., engine and wheel rotational velocities, throttle position, fuel efficiency). In other words, the IDS will notice that the speed readings are anomalous. As another example, if an IDS knows that a substantial increase in an automobile's brake pressure likely precedes a relative decrease in velocity, the IDS can assert that no change, a small change, or an increase in velocity (after significant brake pressure) is anomalous. Of course, this requires an IDS capable of determining expected behavior and identifying anomalies. To design such an IDS for a vulnerable CPS, IDS architects require the following:

1. Insight into the dynamics or patterns of a CPS, to include an understanding of the way in which some current system state or behavior enables predictions concerning a future state;

2. An ample quantity of data obtained under normal operating conditions to establish normal behavior;

3. A process to determine whether new traffic conforms to normal behavior; and

4. An alert system to report to the administrator the traffic that does not conform.

IDS architects can achieve (1) by obtaining either significant understanding of a CPS or sufficiently powerful computational resources. Often, the latter is infeasible: many CPSs are computationally limited by available hardware or by standards and regulations. Modern automobiles, for example, utilize small packets and fairly simple hardware. For this reason, the former is often more attainable. A solid understanding of a system's dynamics, like how one signal affects another or how some current state predicts some future state—causality, in a word—allows an architect to develop a mechanism to identify anomalous traffic.[1] This research thus examines two different techniques to contribute to an architect's understanding of a system. The first, Granger causality, is a well-known, simple method for evaluating the causality between two time series. The second, empirical dynamic modeling (EDM), is an emerging field capable of more sophisticated time series analysis. Our research attempts to demonstrate a potential ability to use one or both techniques as the first step towards an IDS.

[1]Assuming the architect has an ample quantity of normal data.

To do so, we apply Granger causality and EDM to two distinct datasets. We generate the first dataset using a simplistic model of the relationship between an automobile's steering wheel and the relative velocity of its two front wheels. Additionally, the Air Force Research Lab (AFRL) maintains a flight simulator, the Avionics Vulnerability Assessment System (AVAS), which generates the second dataset used in this research. The simulation computes various metrics, like airspeed, angle of attack, position, heading, and wind angle; this research concerns the airplane's airspeed, altitude, and pitch. The simulator logs all values as they change over time. The steering and AVAS datasets represent a linear system and a nonlinear one, respectively.

To evaluate the available analysis tools, we compute Granger causality for every pair of variables in both systems. In doing so, we determine if the simple technique affords insights sophisticated enough to develop an effective IDS. We then apply every EDM tool to the two datasets to establish whether or not its more complex analysis better enables an IDS than does that of Granger causality. The results support Clive Granger's original claim: Granger causality tests do not apply to nonlinear systems in a meaningful way [1]. The results further indicate that, even for linear systems, a Granger causality analysis is likely insufficient for IDS design. However, the results of the EDM analyses imply a possibility of using the technique to develop sophisticated IDSs for nonlinear systems. For linear systems, EDM does not appear to reveal any previously unknown, important dynamics (including those identified by Granger causality or by other analysis techniques).

1.2 Contributions

The primary contributions of this research are as follows:

- A previously unseen demonstration of the unique signature of a vehicle's raw CAN data and a discussion of the associated risks;

- An in-depth presentation of the advantages and disadvantages of Granger causality and EDM, as applied to both linear and nonlinear data;
- The first known argument for applying EDM to cybersecurity and, more specifically, to IDS development; and
- A complete library of available EDM functions for use with any set of time series data of a specific format.

1.3 Organization

The remainder of this thesis reviews the research in detail. Chapter II presents necessary background information before discussing some of the related work in current literature and explaining why this work is insufficient for the research at hand. Chapter III describes the CAN fingerprinting experiments, including the data used, the deep learning methodology, and the results obtained. Chapter IV examines whether Granger causality or EDM could serve as the first step toward an effective IDS. Specifically, the chapter details the data, the analysis techniques, and the results of the analyses. Finally, Chapter V considers the implications of the results found in Chapters III and IV and suggests possible opportunities for future research.

II. Background and Related Work

Each section in this chapter focuses on a specific field of research or technology. The chapter first defines cyber-physical systems (CPSs) and time series data before introducing the controller area network (CAN) protocol and its use in the automotive industry. Next, it details the need for intrusion detection systems (IDSs) in critical systems, including those vehicles that utilize the CAN protocol. It follows by describing causality and explaining Granger causality and empirical dynamic modeling (EDM), two different techniques for causality analysis. The chapter then discusses neural networks and deep learning approaches to classification. Finally, it examines some of the current literature in these fields and highlights relevant gaps in recent research.

2.1 Cyber-Physical Systems & Time Series Data

The Association for Computing Machinery Transactions on Cyber-Physical Systems defines CPSs as follows:

> "Cyber-Physical Systems ... has emerged as a unifying name for systems where the cyber parts, i.e., the computing and communication parts, and the physical parts are tightly integrated, both at the design time and during operation. Such systems use computations and communication deeply embedded in and interacting with physical processes to add new capabilities to physical systems ... There is an emerging consensus that new methodologies and tools need to be developed to support cyber-physical systems." [2]

A CPS can be represented by a model, but this model is typically difficult to understand or replicate.[2] For this reason, one must analyze the CPS's output. Often, the output of CPS monitoring is time series data representing the value of some process (or processes) over time. One example of a time series in an aircraft is the propeller's

[2]Granger causality and EDM attempt to give insight into a CPS's model.

instantaneous revolutions per minute (RPM) over time, as measured by the aircraft's sensors. The National Institute of Science and Technology (NIST) says the following of time series analysis: "Time series analysis accounts for the fact that data points taken over time may have an internal structure (such as autocorrelation, trend or seasonal variation) that should be accounted for" [3]. Kotu and Deshpande contrast time series analysis and forecasting: "Time series *analysis* is the process of extracting meaningful non-trivial information and patterns from time series. Time series *forecasting* is the process of predicting the future value of time series data based on past observations and other inputs" [4]. Most techniques for both analysis and forecasting require data stationarity for the time series in question. Says NIST: "A stationary process has the property that the mean, variance and autocorrelation structure do not change over time ... a flat looking series, without trend, constant variance over time, a constant autocorrelation structure over time and no periodic fluctuations (seasonality)" [3].

Figure 1 presents examples of time series plots; panels (b) and (g) represent stationary time series. Although these are arbitrary plots, they succinctly represent a wide array of potential time series. Many of the time series generated by the CPSs in an automobile or aircraft are non-stationary, so analysis and forecasting techniques that require stationarity (e.g., Granger causality) are likely not viable for these data. Other techniques, like EDM, allow for non-stationary time series analysis and forecasting. Section 2.4 explores causality analysis using two different techniques.

2.2 The Controller Area Network Protocol

The CAN protocol is a link-layer message broadcast system developed for automobile applications by Bosch in the early 1980s [6, 7]. It is commonly used in the automotive, manufacturing, and healthcare industries due to its low cost, low weight, and architectural simplicity. As Figure 2 shows for a set of four devices controlled by

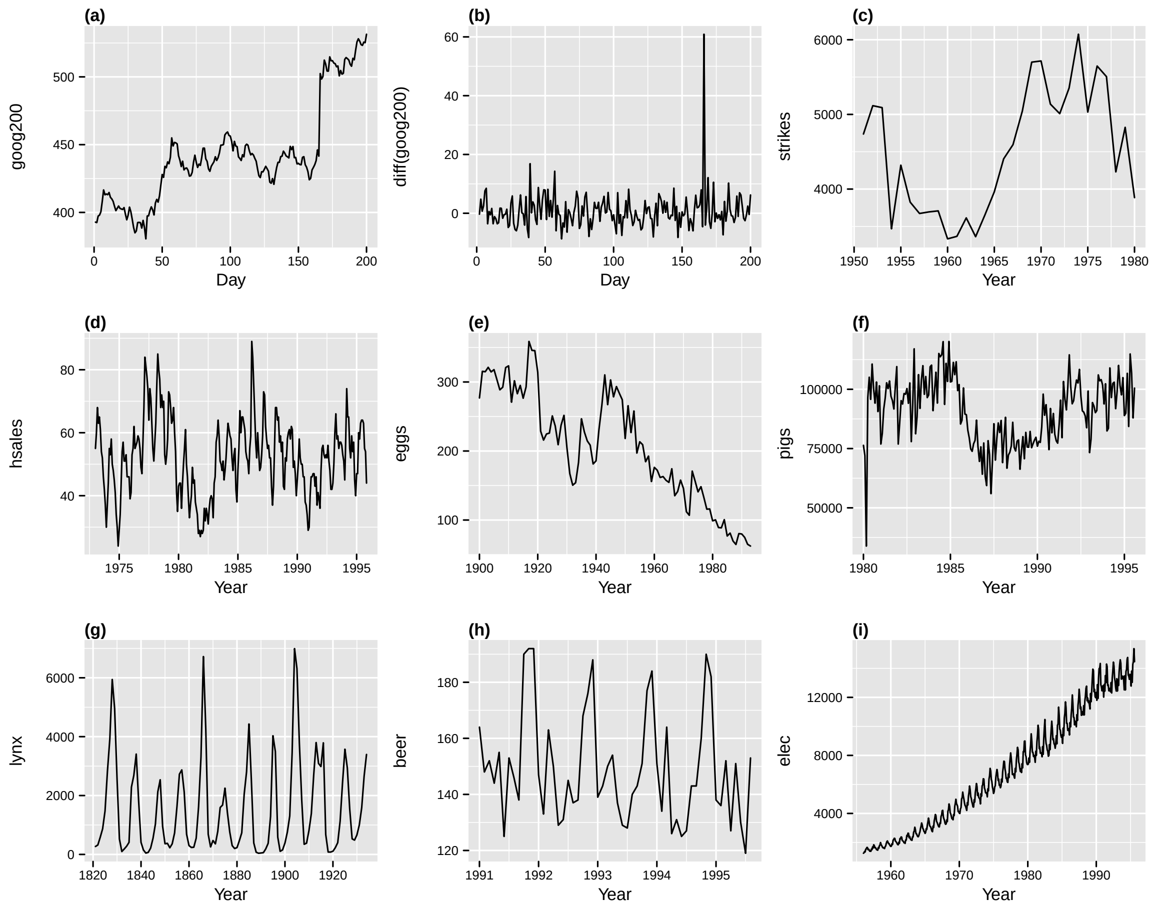


Figure 1. Examples of time series plots. "(*a*) Google stock price for 200 consecutive days; (*b*) Daily change in the Google stock price for 200 consecutive days; (*c*) Annual number of [labor] strikes in the US; (*d*) Monthly sales of new one-family houses sold in the US; (*e*) Annual price of a dozen eggs in the US (constant dollars); (*f*) Monthly total of pigs slaughtered in Victoria, Australia; (*g*) Annual total of lynx trapped in the McKenzie River district of north-west Canada; (*h*) Monthly Australian beer production; (*i*) Monthly Australian electricity production." [5]

an electronic control unit (ECU), the network significantly reduces required wiring within CPSs. Systems that implement the CAN protocol typically use one or more packet frame formats. Figures 3 to 5 display the standard format, the extended format, and the flexible data rate (CAN FD) format, respectively. These formats are backwards compatible. The extended format enables longer message IDs than does the standard format. The CAN FD format enables larger payload sizes and a potential transmission rate increase [8, 9]. For all three formats, only the arbitration ID (ArbID) and payload are variable; the remaining fields—flags and error-checking bits—are set deterministically according to the ArbID and payload. An implementation of the CAN protocol, which links sensing and actuating devices to a computer network, is a classic example of a CPS.

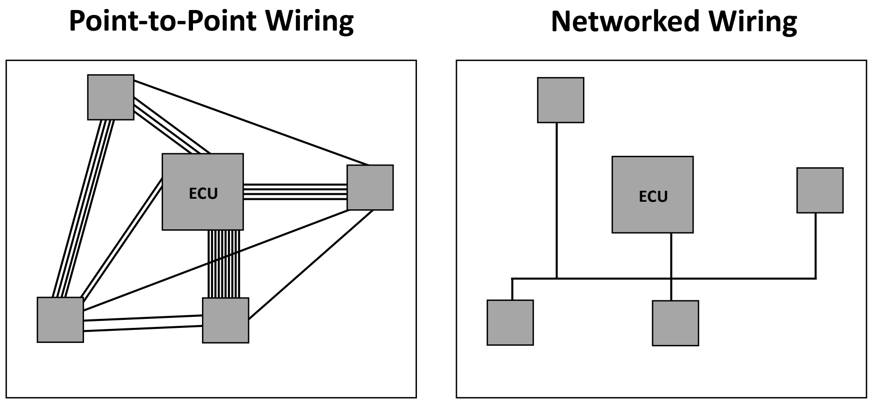


Figure 2. Vehicle network wiring before and after the CAN protocol. [10]

In automobiles, the devices on the network broadcast short messages detailing the vehicle's functionality (e.g., wheel speed, brake pedal pressure, engine RPM); this ensures data consistency throughout the vehicle. United States federal regulations require that all passenger vehicles manufactured after 2007 provide an on-board diagnostics (OBD)-II interface connected to the CAN bus. Consumers, mechanics, and original equipment manufacturers (OEMs) can utilize this interface to monitor some of the car's communications [11]. Recent CAN data collection efforts found that most vehicles utilize the standard or extended frame formats specified in ISO 11898-1:2015, so CAN frames in today's automobiles typically contain no more than eight data bytes [12]. This limits the complexity of CAN data analysis.

The components along the CAN bus are cost-sensitive, so on-bus computational capabilities are limited. For these reasons, automobile manufacturers typically employ a policy of *security through obscurity*, in which the exact data contents are obfuscated

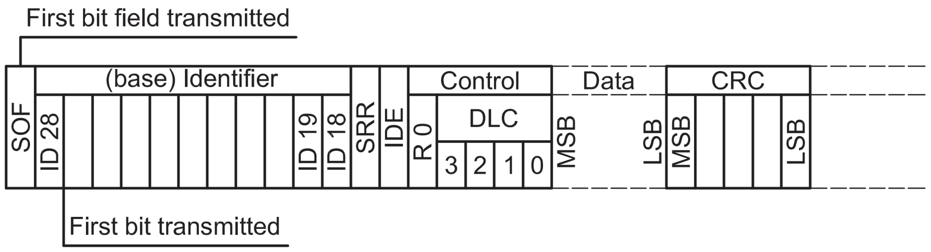


Figure 3. ISO 11898-1 CAN standard message format. [11]

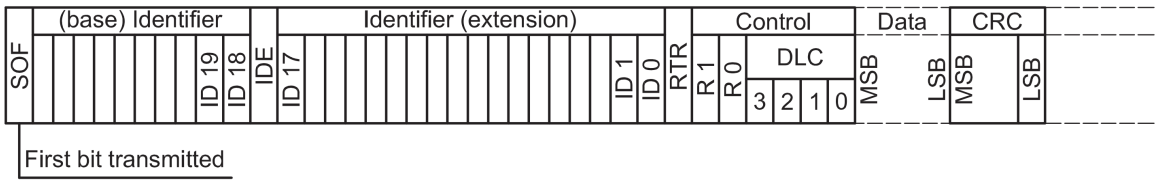


Figure 4. ISO 11898-1 CAN extended message format. [11]

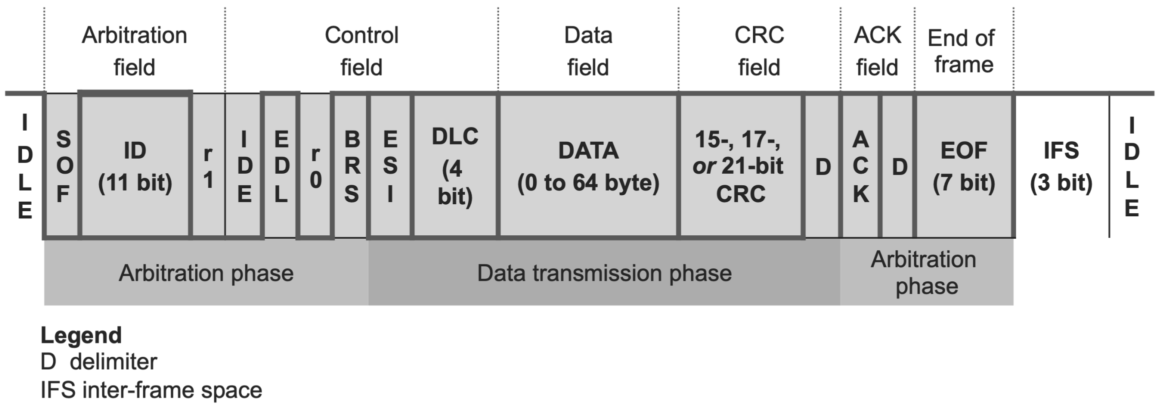


Figure 5. ISO 11898-1 CAN flexible data rate message format. [8]

to prevent significant analysis. For this reason, it is difficult for a non-insider to learn what information is contained in a vehicle's CAN data [13]. However, Stone et al. showed that CAN data collected via an OBD-II port can be automatically reverse-engineered, processed, and converted into time series data. Additionally, the researchers hypothesized that one can quantify the causal relationships present in these time series with EDM [12].

2.3 Intrusion Detection Systems

Much like a home security system, which alerts the police in the event of a burglary, an IDS monitors a computer network and alerts the system administrator in the event of an intrusion. Accomplished cybersecurity researcher Dorothy Denning said the following of IDSs:

> "The development of a real-time intrusion-detection system is motivated by four factors: 1) most existing systems have security flaws that render them susceptible to intrusions, penetrations, and other forms of abuse; finding and fixing all these deficiencies is not feasible for technical and economic reasons; 2) existing systems with known flaws are not easily replaced by systems that are more secure—mainly because the systems have attractive features that are missing in the more-secure systems, or else they cannot be replaced for economic reasons; 3) developing systems that are absolutely secure is extremely difficult, if not generally impossible; and 4) even the most secure systems are vulnerable to abuses by insiders who misuse their privileges." [14]

As the availability of cyber devices has increased over the past three decades, so too has the need for effective IDSs. There are two primary types of IDS: misuse detection and anomaly detection [15]. Misuse detection systems compare network traffic to predefined attack signatures stored in large databases; a match indicates an attack on the network. Anomaly detection systems, on the other hand, compare traffic to a predetermined baseline to identify anomalies. Detecting and preventing intrusions on a vehicle's CAN bus limits the possibility of attacks like the 2015 Jeep hack [16, 17]. Such an attack is made possible by the CAN's prime vulnerabilities: "1) weak access control; 2) no encryption; 3) and no authentication" [18]. Tyree et al. discuss three IDS approaches specific to the CAN domain [19]:

1. Rule-based IDSs: "Early works involve rule-based detectors, an analogue of signature-based detection in enterprise security. These can detect simple signal-injection attacks, (sophisticated but very limited) bus-off attacks ... and potentially ECU reprogramming".

2. Frequency-based IDSs: These systems aim "to exploit the regular frequency of important CAN messages. Frequency anomalies have been explored to detect and prevent signal-injection attacks and potentially ECU reprogramming".

3. ECU-fingerprinting IDSs: Such detectors consider "automatic ECU identification as a stepping stone to IDS ... [by using] data-driven techniques to classify which ECU sent each message by exploiting timing or voltage signatures".

Recently, various approaches in deep learning have allowed for the development of effective IDSs in all three categories. Kang and Kang present one such example [20]. See Liao et al. and Kwon et al. for discussions on further examples [21, 22].

2.4 Causality & Time Series Analysis

Causality is the relationship between cause and effect. For two processes a and b that exhibit a highly causal relationship, one can expect that some set of parameters for a predicts some other set of parameters for b. If a and b are not causal, modifying a is not likely to elicit a predictable response in b. In passenger vehicles, causality is ever-present. The engine's RPM, for example, directly affects the vehicle's speed. Brake position indirectly affects engine RPM and thus the vehicle's speed. Causal relationships also hold in the reverse direction. Some perceived value for vehicle speed implies that the vehicle must have had an engine RPM in some identifiable range at some previous point in time. Pearl, one of the preeminent researchers in the field, says the following of causality:

> "We say, for example, 'reckless driving causes accidents' or 'you will fail the course because of your laziness' (Suppes 1970), knowing quite well that the antecedents merely tend to make the consequences more likely, not absolutely certain. Any theory of causality that aims at accommodating such utterances must therefore be cast in a language that distinguishes various shades of likelihood—namely, the language of probabilities. Connected with this observation, we note that probability theory is currently the official mathematical language of most disciplines that use causal modeling, ... In these disciplines, investigators are concerned not merely with the presence or absence of causal connections but also with the relative strengths of those connections and with ways of inferring those connections from noisy observations. Probability theory ... provides both the principles and the means of coping with—and drawing inferences from—such observations." [23]

In fewer words, Pearl asserts that causality is difficult to measure. Researchers who seek to analyze causality in a system must approximate it using methods in statistical probability. Granger causality presents one such method. EDM provides several statistical analysis techniques, and some of these techniques also seek to estimate the causality in a system. Understanding this causality could enable an effective IDS.

2.4.1 Granger Causality

Pearl further claims that Granger causality is a "statistical rather than causal [method]" meaning that it does not directly quantify causality [23]. Still, Granger's technique is likely the most prominent and widely used method for approximating causality in a system, probably because it "does not rely on the specification of a scientific model and thus is particularly suited for empirical investigations of cause-effect relationships" [24]. If the following properties hold for the variables X and Y, then X is said to Granger-cause Y [1, 24]:

1. Y does not precede X in time.
2. X contains information that improves the prediction of Y.
3. This information is not contained in any other series.

It is difficult to deterministically confirm (3). For this reason, one mathematically identifies Granger causality by predicting Y; if the variance of the prediction error is lower when X is included in the prediction set than when it is not, then X Granger-causes Y [1]. Note that the accuracy of the causality estimation is improved if both X and Y are stationary. Granger causality does have one other major caveat: because it essentially measures association between two variables, it can lead to illegitimate causalities if one does not consider important relevant variables [24]. Overall, though, this concept serves as a key tool for time series analysis, and this research shows that it allows one to draw limited conclusions about time series data generated by some CPSs. Figure 6 presents an example of a time series X that Granger-causes another time series Y.

2.4.2 Empirical Dynamic Modeling

Floris Takens introduced his delay embedding theorem in 1981 [26]. Takens's Theorem concerns mathematical attractors, where "an attractor is the value, or set of values, that a system settles toward over time" [27]. EDM is an application of Takens's Theorem. In Sugihara et al.'s words, the field "is based on the mathematical theory of reconstructing system attractors from time series data" [28]. In practice, it allows one to model nonlinear dynamic systems with observational time series data. Figure 7 provides a summarized visual explanation of the main ideas in Takens's

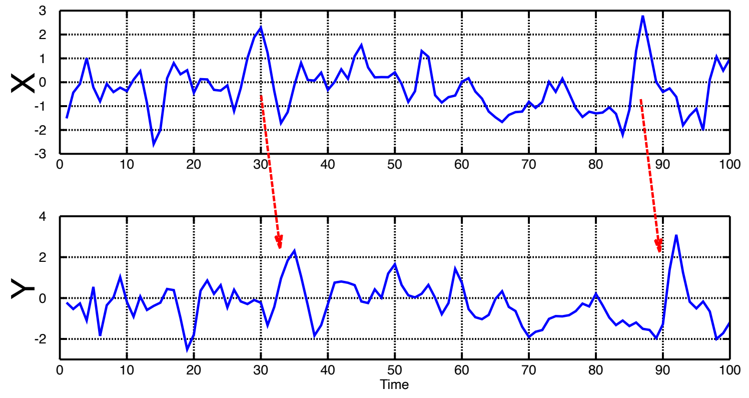


Figure 6. Visualization of Granger causality. Here, X Granger-causes Y. [25]

Theorem and in EDM. Specifically, panel (A) depicts a Lorenz attractor[3] as a model of a dynamic system. As the image shows, one can identify a time series for a given dimension by recording that dimension's observations over time. Panel (B) shows that a univariate time series can be converted to a higher dimensional representation by using time-lagged versions of itself as additional dimensions. EDM calls the resulting manifold a shadow manifold. Takens showed that the shadow manifold is diffeomorphic (maps one-to-one) to its original attractor manifold M [26].

Sugihara et al. demonstrated that this diffeomorphic property between M and its shadow manifolds—one for each dimension—implies that the shadow manifolds are diffeomorphic with respect to each other. The opposite is also true. Thus, if two shadow manifolds are shown to be diffeomorphic with respect to each other, one can assume they belong to the same dynamic system. One can then use convergent cross mapping (CCM), a mathematical technique recently developed by Sugihara et al., to identify the presence of and quantify the causality between the two original time series [31]. In short, CCM seeks to determine whether an arbitrary point and its nearest neighbors in one shadow manifold can accurately predict a point and its neighbors in

[3]The Lorenz attractor is a set of solutions to the Lorenz system, a system of ordinary differential equations first studied by Edward Lorenz in 1963 [29].

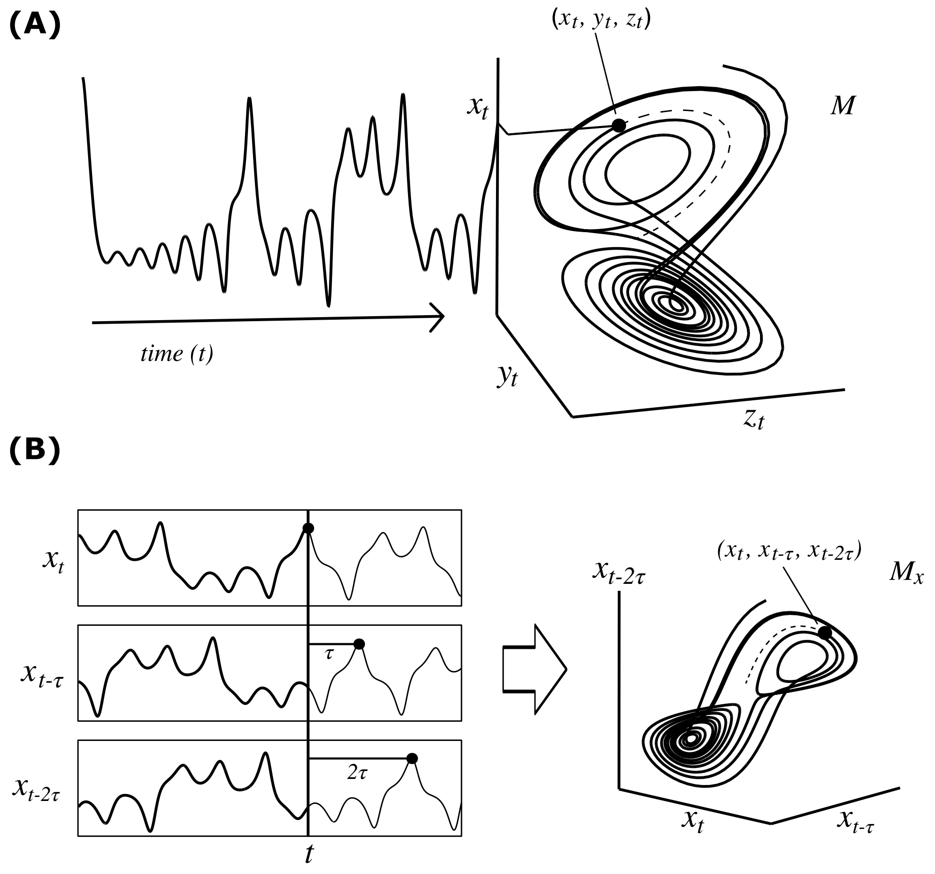


Figure 7. "Empirical dynamic modeling: *(A)* Example Lorenz system. The attractor manifold M is the set of states that the system progresses through time. Projection of the system state from M to the coordinate axis X generates a time series. *(B)* Lags of the time series X are used as coordinate axes to construct the shadow manifold M_X, which is diffeomorphic (maps 1:1) to the original manifold M. The visual similarity between M_X and M is apparent." [28, 30]

another shadow manifold. Figure 8 summarizes this concept. Sugihara et al. showed that increasing the sample sizes for the shadow manifolds improves CCM's predictive power, but they also showed that this predictive power converges to some maximum as the sample sizes increase to infinity [26, 31].

2.4.3 Autoregressive Integrated Moving Average

There exists another, much more well-known method for forecasting time series: autoregressive integrated moving average (ARIMA) models. Implementations of ARIMA first difference a time series to ensure data stationarity before regressing past

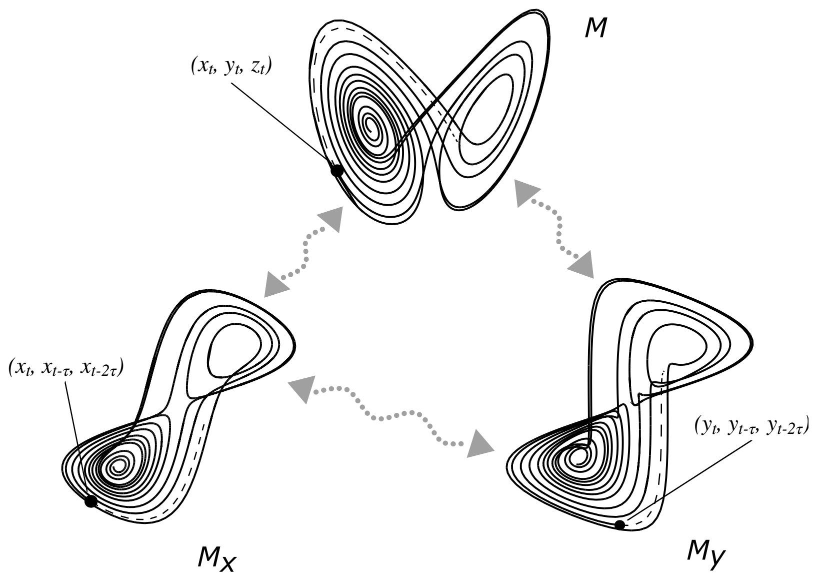

M
(x_t, y_t, z_t)
$(x_t, x_{t-\tau}, x_{t-2\tau})$
$(y_t, y_{t-\tau}, y_{t-2\tau})$
M_X
M_Y

Figure 8. "Convergent cross mapping (CCM) tests for correspondence between shadow manifolds. This example based on the canonical Lorenz system (a coupled system in X, Y, and Z ...) shows the attractor manifold for the original system (M) and two shadow manifolds, M_X and M_Y, constructed using lagged-coordinate embeddings of X and Y, respectively ($lag = \tau$). Because X and Y are dynamically coupled, points that are nearby on M_X ... will correspond temporally to points that are nearby on M_Y ... This enables us to estimate states across manifolds using Y to estimate the state of X and vice versa using nearest neighbors ... With longer time series, the shadow manifolds become denser and the neighborhoods (ellipses of nearest neighbors) shrink, allowing more precise cross-map estimates" [30, 31]. The arrows between the manifolds represent the diffeomorphic properties of the attractors.

values, or lags, of the time series against the original to make predictions. The method uses a moving window approach, which means it computes forecast error using the errors for past slices of the time series [4, 32]. This technique could prove useful in IDS design. However, because it forecasts a time series using only the data from that series, it is not likely to contribute significantly to the deep understanding of a system's dynamics required by an IDS. However, some researchers have published promising work in ARIMA applications to cybersecurity. Yaacob et al., for example, obtained notable IDS results for a specific attack with a set of test data, but the researchers imply a high likelihood of false alarms and thus recommend more research efforts [33]. Similarly, Shirani, Azgomi, and Alrabaee compute confidence intervals for future network traffic using ARIMA. Although the proposed methodology can identify attacks with specific data patterns, their research indicates that ARIMA cannot enable a general-purpose IDS for CPSs [15]. For this reason, we consider techniques—like EDM—which afford IDS architects a deeper understanding of a system.

2.5 Deep Learning

Deep learning, a subfield of machine learning, is motivated by the desire to solve problems which standard computing approaches either fail to solve or solve suboptimally. Although the field has a long history of developments, much of its notable success is recent. As pioneering scientist Terrence Sejnowski claims, “What made it possible for deep learning to make big breakthroughs on some of the most difficult problems in artificial intelligence was persistence, big data, and a lot more computer power” [34]. Until the 2000s, researchers simply lacked the hardware, software, and data necessary for effective applications of deep learning techniques. Recently, such techniques have achieved success and garnered attention in the fields of computer vision, audio processing, game-playing, and time series analysis (among others). In the context of this research, deep neural networks enable pattern detection in CAN data, both for classification and prediction purposes.

2.5.1 Neural Networks

The fundamental deep learning tool is the feedforward neural network or multilayer perceptron (MLP). Effectively, an MLP aims to loosely approximate the human brain by utilizing a set of neurons or nodes. In practice, researchers use MLPs to approximate a function f^*; the network “defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation” [35]. Such a network is called “feedforward” because information flows strictly from the input layer, through the computations that define f, and to the output layer.[4] The interior layers in a neural network (if such layers exist) are called hidden layers because the outputs of these layers are not freely inspectable. A neural network is a *deep* neural network if it contains at least two hidden layers.[5] One can often use an MLP to approximate a nonlinear function using few hidden layers [37]. Figure 9 presents such an MLP; this one is capable of emulating the *exclusive or* operation.

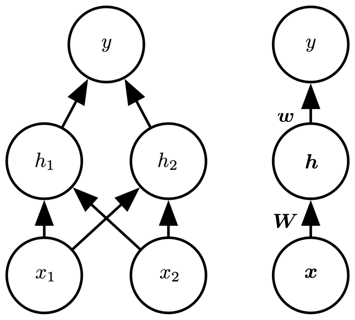


Figure 9. "An example of a feedforward network, drawn in two different styles ... It has a single hidden layer containing two units. (*Left*) In this style, we draw every unit as a node in the graph. This style is explicit and unambiguous, but for networks larger than this example, it can consume too much space. (*Right*) In this style, we draw a node in the graph for each entire vector representing a layer's activations. This style is much more compact. Sometimes we annotate the edges in this graph with the name of the parameters that describe the relationship between two layers. Here, we indicate that a matrix W describes the mapping from x to h, and a vector w describes the mapping from h to y. We typically omit the intercept parameters associated with each layer when labeling this kind of drawing." [35]

In deep learning, convolutional neural networks (CNNs) "are a specialized kind of neural network for processing data that has a known grid-like topology"; because time series are effectively one-dimensional grids, these networks are well-suited for time series data [32, 35]. Where standard neural networks use general matrix multiplication operations to approximate f^*, CNNs utilize convolution, a special mathematical operation in which a user-defined kernel convolves with an input to generate an output.

[4]Neural networks that use feedback connections, where information flows back to previous layers, are called recurrent neural networks.

[5]Definitions for *deep* networks differ. Two or more hidden layers is a common definition [36].

The kernel, input, and output are represented by multidimensional arrays.[6] Figure 10 depicts a simple convolution operation.

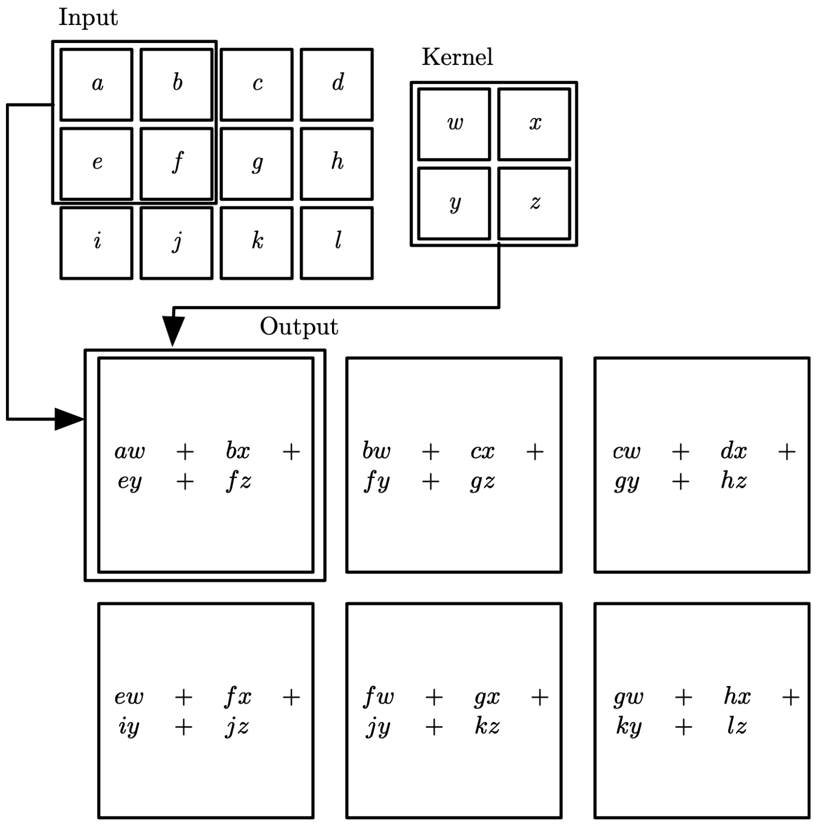


Figure 10. "An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called 'valid' convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor." [35]

2.5.2 Classification

Classification, one of the primary applications of deep learning, concerns the task of predicting an input's class (or category). Goodfellow, Bengio, and Courville summarize classification in the following way:

[6]In the context of time series or CAN data, each array is of shape $n \times 1$.

> "In this type of task, the computer program is asked to specify which of k categories some input belongs to. To solve this task, the learning algorithm is usually asked to produce a function $f : \mathbb{R}^n \rightarrow \{1, ..., k\}$. When $y = f(x)$, the model assigns an input described by vector x to a category identified by numeric code y. There are other variants of the classification task, for example, where f outputs a probability distribution over classes." [35]

In the automotive domain, neural network classifiers enable insight into CAN data structure. As this research demonstrates, a well-trained deep learning model can correctly determine, with 99% accuracy, which of n vehicles generated a given segment of CAN data. This illustrates a vulnerability in the ubiquitous protocol: although OEMs obfuscate message payload structure, the generating vehicle is still uniquely identifiable, and thus the passengers are at risk of targeted attacks.

2.5.3 Siamese Neural Networks

Siamese neural networks (SNNs) are a different type of neural network to determine whether two inputs represent the same entity or different entities. An SNN can be implemented as any type of network, including MLPs and CNNs. One may imagine an SNN as consisting of two networks in tandem, where the networks have identical architectures and use identical weights. The SNN receives two matrices as input, encodes each as a feature vector, and compares the two vectors [38].[7] To determine whether two inputs represent the same entity, the SNN uses one of two primary loss functions: (1) contrastive loss and (2) triplet loss. The former computes the absolute difference between the vectors and reports *same* if the difference surpasses some threshold and *not same* otherwise. The latter requires a third input—an anchor—and updates the network's weights such that the anchor input and the *same* input encode to similar vectors, while the anchor input and the *not same* input encode to dissimilar vectors [37, 38].

[7] In practice, one implements a single network to reduce space requirements; the network receives and encodes each half of the input pair before comparing the encoded vectors [37].

This functionality may enable a more robust vehicle classifier. An attacker with a well-trained SNN need not collect data from every possible vehicle. Instead, the attacker only needs enough data to train the SNN to determine whether the same vehicle generated two different samples of CAN data. At that point, the attacker can effectively classify any new vehicle by comparing its data samples to those of the previously seen vehicles.

2.6 Related Work

Most researchers concerned with CAN security attempt to predict future values in a time series. For example, Marchetti et al. devise an IDS by analyzing sequential CAN IDs, and Tyree et al. create an IDS that learns relationships in different time series signals [19, 39]. However, some researchers in recent years have used deep learning to classify CAN data. Kang and Kang train a deep neural network to classify CAN data packets as either normal or abnormal [20]. Kwon et al.'s literature survey details more examples of deep learning as applied to IDSs and to CAN security [22]. Recent research also shows that one can certainly distinguish CAN data. Enev et al. show that machine learning can discriminate between different drivers using just the reverse-engineered CAN data collected from the vehicle [40]. However, their research does not claim to distinguish the raw CAN data generated by different vehicles. In summary, previous research efforts have demonstrated the ability to (a) predict CAN data (but not classify it); (b) classify CAN signals as good or bad; or (c) classify vehicles by analyzing reverse-engineered CAN data. None of these efforts depict an ability to classify vehicles using raw CAN data. This is an important gap in current research, and our thesis seeks to address this gap.

Concerning applications of time series analysis to IDSs, Qin and Lee explored a tangential application of Granger causality to cybersecurity efforts. Specifically, the authors used Granger causality to correlate and aggregate security alerts to identify high-priority alerts [41]. Cabrera et al. (2001) applied Granger causality to the traffic preceding a network attack to determine which variables best predict the attack. They then showed that observation of these important variables can assist in determining signatures of network attacks [42]. Finally, Cabrera et al. (2002) conducted limited, controlled experiments in a very similar domain. Their work uses Granger causality to discover relevant variables prior to distributed denial of service attacks [43]. Our work differs from previous work in that it compares EDM to Granger causality as a time series analysis technique for IDS development. Additionally, past research does not examine Granger causality applications to general-purpose IDSs; in each case, the researchers attempt to predict very specific attacks.

Finally, our investigation of the current literature revealed no research into EDM applications to automobile- or aircraft-generated time series or to cybersecurity as a whole. Most applications of the techniques concern economics or natural sciences; for example, the Sugihara Laboratory, from which EDM originated, primarily applies the techniques to ecology. We foresee EDM as a useful contribution to the cybersecurity domain, and our research thus seeks to link the two in a way not yet found in the literature.

III. Fingerprinting Vehicles with CAN Bus Data Samples

This chapter addresses the controller area network (CAN) vulnerability mentioned in Chapter I. Specifically, this chapter shows definitively that an original equipment manufacturer (OEM)'s obfuscation processes do not mask a vehicle's identity because CAN data from one vehicle is uniquely identifiable. Two deep learning models, one a multilayer perceptron (MLP) and the other a convolutional neural network (CNN), are trained for each of three distinct data formatting processes to determine how well a model can distinguish between CAN data from disparate vehicles. Input to the models consists of the formatted CAN data; each model then predicts the vehicle ID and compares its output to the true vehicle ID. As the results of the experiments indicate, CAN data can be uniquely attributed to its generating vehicle regardless of the data formatting process. The remainder of this chapter discusses the nature and origins of the data. It then discusses each of the experiments in turn, to include the data formatting process, the deep learning models employed, and the results.

3.1 Data

This research uses 230 megabytes of Stone et al.'s CAN data [12]. The researchers conducted one capture on each of 11 different vehicles. Table 1 displays metadata for the vehicles in the dataset.

A large comma-separated values (CSV) file stores the data. Each of the 4,161,755 rows in this file represents one CAN message, and each row contains:

- A `capture_id` and a `vehicle_id`, which identify the message's origin;
- A `timestamp` relative to the start of the capture;
- An `arbitration_id`, which identifies the subject of the message's contents;

Table 1. Make, Model, and Year for Each Vehicle in the Dataset

Vehicle	Make	Model	Year
1	Chevrolet	Cobalt	2009
2	Chevrolet	Silverado	2011
3	Dodge	1500	2014
4	Ford	F-150	2017
5	Ford	Focus	2010
6	Honda	Accord	2012
7	Honda	Accord	2015
8	Nissan	370Z	2015
9	Nissan	XTERRA	2010
10	Saab	9-7X	2009
11	Toyota	Corolla	2009

- A `dlc`, or data length code, which indicates the number of payload data bytes;
- The hexadecimal `data` itself; and
- The vehicle's `make`, `model`, and `year`.

3.2 Classifying Ordered CAN Payload Data

The messages in the CSV file are not immediately suitable for deep learning. Each message contains no more than eight data bytes (and many contain fewer), and it is extremely unlikely that so little information can uniquely identify a vehicle. To address this issue, we build a set of data samples, where each sample contains 1,024 bytes of sequential CAN data from one capture. That is, for each capture, we sort the messages by timestamp before splitting all hex data into a list of hex bytes, converting the hex bytes into three-digit integers, and dividing the list into samples of 1,024 integers each. Because each integer represents one byte of CAN data, each sample contains $8 \times 1024 = 8192$ individual data bits. We then write each sample and the associated vehicle ID to a new CSV file. This makes it more likely that each sample contains sufficient information and that the underlying CAN data structure is preserved. Table 2 presents a few examples of these data samples.

Table 2. **Ordered Data — Example Samples**

Vehicle	Ordered Data
3	074 166 111 254 255 240 254 ...
7	003 212 003 209 003 199 003 ...
8	255 248 000 128 015 254 030 ...

The new CSV file contains 28,425 samples from the 11 vehicles. Table 3 illustrates the distributions of samples over all vehicles (the number of samples depends primarily on the length of the data capture). In a balanced dataset, each vehicle would contain about nine percent of all samples. Clearly, this dataset is not balanced. We address this class imbalance during model training.

Table 3. **Ordered Data — Samples Per Vehicle**

Vehicle	Samples	Proportion
1	4115	14.48 %
2	1856	6.53 %
3	1812	6.37 %
4	2221	7.81 %
5	3806	13.39 %
6	1709	6.01 %
7	2220	7.81 %
8	3041	10.70 %
9	2564	9.02 %
10	2980	10.48 %
11	2101	7.39 %

3.2.1 Model Architecture

Using an MLP to classify vehicles is a naive approach, but it is useful to present the naive approach's performance to demonstrate that an attacker does not need complex methods to adequately classify vehicles. Each CAN sample contains 1,024 data bytes, so the model receives 8,192 bits as input and outputs one of 11 classes. To identify the best MLP, we test multiple hyperparameter configurations. The left panel of Figure 11 displays the best configuration. It has one hidden layer of 512 nodes, and it uses a `relu` activation function. The dropout layer sets 20% of its inputs to zero. The output layer contains 11 nodes, one for each vehicle in the dataset, and uses a `softmax` activation function. We compile the model with an Adam optimizer of learning rate $lr = 0.001$ and a categorical cross-entropy loss function. The model has 530,443 parameters, all of which are trainable. In total, we test 27 MLP architectures. Table 4 lists the various options for each hyperparameter used during model construction.

Table 4. Hyperparameter Options for MLP Construction

Hyperparameter	**Options**
Number of Hidden Layers	1, 2, 3
Number of Nodes Per Hidden Layer	128, 256, 512
Optimizer Learning Rate	0.01, 0.001, 0.0001

A more complex approach—a CNN—demonstrates that a sophisticated attacker can achieve superior performance over the naive approach. The CNN utilizes one-dimensional convolutional layers because each sample contains one-dimensional data. In testing, the CNN receives one sample as input and outputs one of the 11 classes. Like with the MLP, we test multiple hyperparameter configurations before selecting the best CNN. The best model utilizes a typical CNN structure consisting of convolutional, pooling, and dropout layers. The right panel of Figure 11 displays the specific architecture. The convolutional layers each use 32 filters with a kernel size of four, the pooling layer pools over four elements at each step, and the dropout layer sets 20% of its inputs to zero. The convolutional layers and the first dense layer all use a `relu` activation function; the final dense layer uses `softmax`. We again compile the model with an Adam optimizer of learning rate $lr = 0.001$ and a categorical cross-entropy loss function. The model has 818,427 parameters, and all but 64 are trainable. In total, we test 18 CNN architectures. Table 5 lists the various options for each hyperparameter used during model construction.

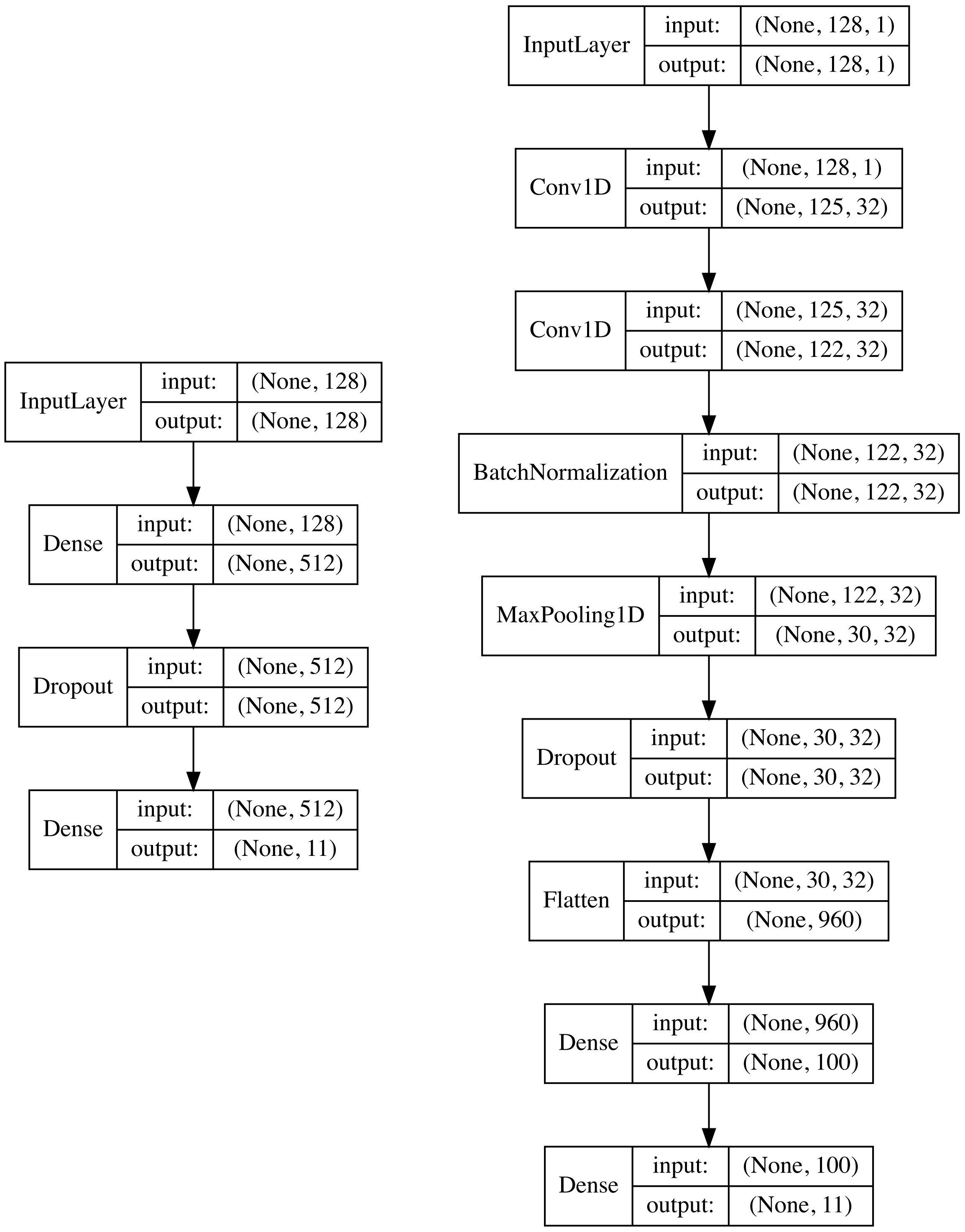


Figure 11. The best architectures identified in iterative model evaluation processes. Input to the models consists of sequential CAN bus data. (*Left*) MLP. (*Right*) CNN.

Table 5. Hyperparameter Options for CNN Construction

Hyperparameter	Options
Filters Per Convolutional Layer	32, 64
Kernel Size Per Filter	2, 3, 4
Pooling Size	2, 3, 4

3.2.2 Model Fitting

Table 3 shows that the classes in the full dataset are somewhat imbalanced. For example, nearly 15% of all samples come from Vehicle 1, but one should expect each vehicle to contain about 9% of the samples. To address this imbalance, we use scikit-learn's `class_weight` module during model training. This tool adjusts weights to account for the frequency of each class, and thus it ensures that both models can use all training data without unnecessarily suffering from class imbalance [44]. Still, we also create a second, balanced dataset by randomly sampling $n = 1{,}695$ samples from every vehicle.

Another important tool during model training is Keras's `EarlyStopping` callback, which limits overfitting [45]. This function monitors validation loss and terminates training if the loss does not improve in 10 epochs. The callback then restores the model to its best version. This tool also limits model training time. To properly train and evaluate the models, we split each dataset into three sets: 60% of the samples are used in training, 20% in validation, and 20% in testing. Scikit-learn's `train_test_split` enables this split by randomly sampling from the dataset [44]. We train each model on the training set, terminate training with the early stopping callback (which monitors the validation set's loss), and evaluate the trained model on the testing set. However, these tools do not guarantee optimality, so we also iteratively improve the models through repeated testing and hyperparameter tuning. A search over a set of possible hyperparameter configurations ensures we can identify the best classifier. In total, we evaluate 27 different MLPs and 18 different CNNs. Tables 4 and 5 depict the hyperparameter options for these models.

Figure 12 shows that the final models possess sufficient capacity for this task. The validation loss and accuracy for each model are nearly as good as the training loss and accuracy—especially for the CNNs—and all four losses and accuracies appear to reach an asymptote. This means that additional training will not benefit the models, and it also means that the models are nearly optimal for this task. Note that even the best MLP struggles to generalize to new data.

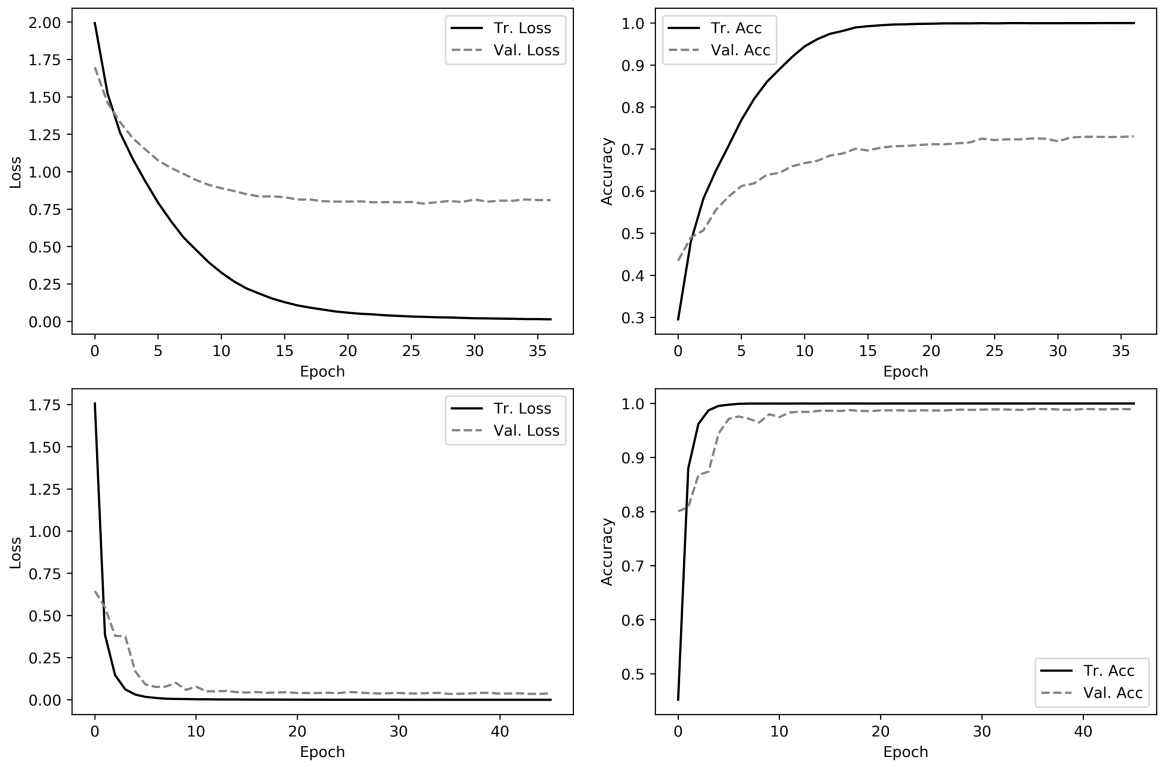


Figure 12. Training and validation loss and accuracy over time for the best models. Input to the models consists of sequential CAN bus data. (*Top left*) The MLP's loss. (*Top right*) The MLP's accuracy. (*Bottom left*) The CNN's loss. (*Bottom right*) The CNN's accuracy.

To evaluate model performance, we compute balanced accuracy for both the balanced and the imbalanced dataset. This metric considers the various class distributions when computing the classification accuracy. Even for the balanced dataset, this is important. Because we randomly sample from the dataset to build the training, validation, and testing sets, each of those three sets could be imbalanced. Additionally, we present confusion matrices to allow for an in-depth analysis of where each model fails. For example, Manufacturer X could use the exact same CAN configuration for different vehicles, so even a well-trained model could fail to distinguish between Manufacturer X's vehicles. These analyses may even imply a similar CAN structure across different manufacturers, so the results of these investigations guide further model modifications and tuning.

3.2.3 Results

The MLP's performance demonstrates that an intruder with a low-level understanding of deep learning could use this field to better format CAN attacks. The CNN's performance shows that a sophisticated intruder can use more advanced techniques to achieve much better results. Specifically, results indicate that both the MLP and the CNN can adequately classify CAN data segments, but the CNN is significantly better than the MLP on both the imbalanced dataset and the balanced dataset. Results also indicate that both models perform better on the larger, imbalanced dataset; this is simply due to the availability of training data in each set. Table 6 presents the overall balanced accuracies.

Table 6. Ordered Data — Balanced Accuracy

Model	**Dataset**	**Accuracy**
Multilayer Perceptron	Imbalanced	94.46 %
Multilayer Perceptron	Balanced	67.48 %
Convolutional Neural Network	Imbalanced	99.64 %
Convolutional Neural Network	Balanced	95.63 %

Table 7 presents class-specific accuracy values for the four trained models. Clearly, the CNN trained on the imbalanced dataset performs better than the other models on every class (except Vehicle 5, for which both the CNN and the MLP achieve 100%

accuracy). Additionally, the CNN trained on the balanced dataset outperforms either version of the MLP on most (10/11) of the classes. This is further evidence of the superior performance of the CNN, and it also indicates that more training data is better, even if the classes in the data are imbalanced.

Table 7. Ordered Data — Class Accuracy

Vehicle	MLP, Imbal.	MLP, Bal.	CNN, Imbal.	CNN, Bal.	Median
1	97.34 %	79.36 %	99.88 %	98.34 %	97.84 %
2	89.90 %	31.07 %	100.00 %	99.30 %	94.60 %
3	91.32 %	56.10 %	100.00 %	93.77 %	92.54 %
4	98.53 %	78.12 %	100.00 %	96.23 %	97.38 %
5	99.87 %	83.10 %	100.00 %	98.13 %	99.00 %
6	91.25 %	77.32 %	99.18 %	98.99 %	95.12 %
7	91.90 %	40.72 %	99.54 %	90.74 %	91.32 %
8	90.15 %	60.70 %	98.68 %	93.92 %	92.03 %
9	94.68 %	75.16 %	100.00 %	99.70 %	97.19 %
10	100.00 %	94.27 %	100.00 %	96.74 %	98.37 %
11	95.27 %	65.22 %	100.00 %	97.48 %	96.38 %

Figure 13 presents the multiclass confusion matrices for both model types on both datasets. Generally, the models tend to correctly classify CAN segments, but some specific misclassifications are certainly common. For example, all four trained models misclassify Vehicle 6 as Vehicle 7 (and vice versa) to some extent; these are different-year Honda Accords. Additionally, the models usually perform similarly on the same vehicle. For example, all models demonstrate strong performance on Vehicle 10—perhaps Saab employed a distinct data structure when designing its 2009 9-7X. Conversely, neither of the MLP models achieve relatively strong performance on Vehicle 2. One can easily discern other trends by further analyzing Figure 13.

3.3 Classifying Ordered CAN ArbIDs

As Section 3.2 demonstrates, deep learning can effectively determine which vehicle in a set generated a given segment of raw CAN data. One potential reason why is that the deep learning models identify the sequential structure of the CAN data—that is, the models learn to fingerprint vehicles by analyzing the order and frequency of

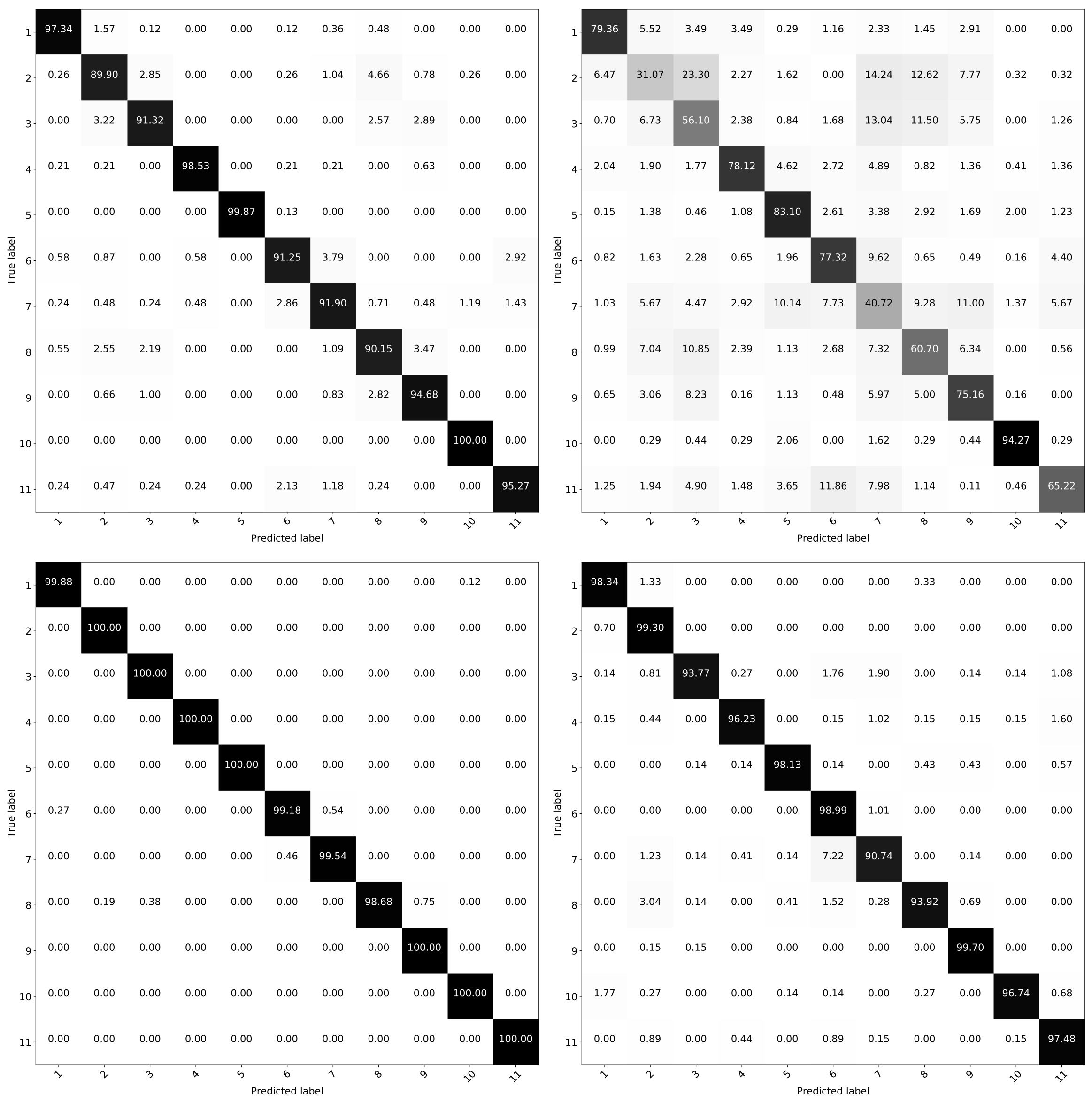


Figure 13. Confusion matrices. Input to the models consists of sequential CAN bus data. (*Top left*) MLP on the imbalanced dataset. (*Top right*) MLP on the balanced dataset. (*Bottom left*) CNN on the imbalanced dataset. (*Bottom right*) CNN on the balanced dataset.

different CAN payloads. In assuming that this hypothesis is correct, it is natural to consider the classification performance of models that receive as input segments of arbitration IDs (ArbIDs) instead of data payloads. Such models could still learn to identify a vehicle's unique sequential packet structure, but both the required size of the training dataset and the time to train the models would decrease. This section details the methodology and results of such an experiment.

Like the experiment described in Section 3.2, the raw CAN messages require preprocessing prior to evaluating the deep learning models. We thus construct a set of ArbID samples; each sample contains 128 ArbIDs in the order they appear on a vehicle's CAN bus. To do so, we again sort the messages in each capture by timestamp before splitting the list of ArbIDs into samples of 128 units; we then write each sample and the associated vehicle ID to a new CSV file. Table 8 depicts three examples of these ArbID samples. The CSV file contains 32,508 distinct ArbID samples from the 11 vehicles. Table 9 lists the percentage of samples belonging to each vehicle. As in Section 3.2's experiment, we address the class imbalance during model training.

Table 8. Ordered ArbIDs — Example Samples

Vehicle	Ordered ArbIDs
3	11C 118 124 140 202 122 1A0 ...
7	1AA 1B0 1D0 1DC 1EA 1ED 320 ...
8	292 245 216 160 180 182 1F9 ...

Table 9. Ordered ArbIDs — Samples Per Vehicle

Vehicle	Samples	Proportion
1	4903	15.08 %
2	2346	7.22 %
3	1936	5.96 %
4	2221	6.83 %
5	3806	11.71 %
6	1947	5.99 %
7	2735	8.41 %
8	3416	10.51 %
9	2874	8.84 %
10	3799	11.69 %
11	2525	7.77 %

3.3.1 Model Architecture

We again evaluate both the naive MLP and the more complex CNN in this experiment. Each model receives 128 ArbIDs as input and outputs one of the 11 classes. As in Section 3.2's experiment, we conduct a search over hyperparameter configurations for both model types; Tables 4 and 5 present the various options tested. The optimal MLP has three hidden layers, each of which has 512 nodes and uses a `relu` activation function. The dropout layer sets 20% of its inputs to zero, and the output layer uses a `softmax` activation function to classify an input as one of the 11 vehicles. The model, which we compile with an Adam optimizer of learning rate $lr = 0.001$ and a categorical cross-entropy loss function, has 597,003 trainable parameters. The left panel of Figure 14 presents the model's exact architecture. The optimal CNN, on the other hand, uses 32 filters, each with a kernel size of two, in both of its one-dimensional convolutional layers, and its pooling layer aggregates two elements at each step. The dropout layer sets 20% of its inputs to zero. The convolutional layers and first dense layer use a `relu` activation function, and the final dense layer uses a `softmax` activation function. We compile the model with an Adam optimizer of learning rate $lr = 0.001$ and a categorical cross-entropy loss function. In total, this CNN has 102,651 trainable parameters. The right panel of Figure 14 depicts the model's architecture.

3.3.2 Model Fitting

As Table 9 shows, the full dataset disproportionately represents the 11 vehicles. We again address this class imbalance by 1) utilizing the `class_weight` module when training on the imbalanced dataset and 2) downsampling to create a second, balanced dataset. Additionally, we again limit overfitting by defining an `EarlyStopping` callback that terminates training after 10 epochs without any improvement in training loss.

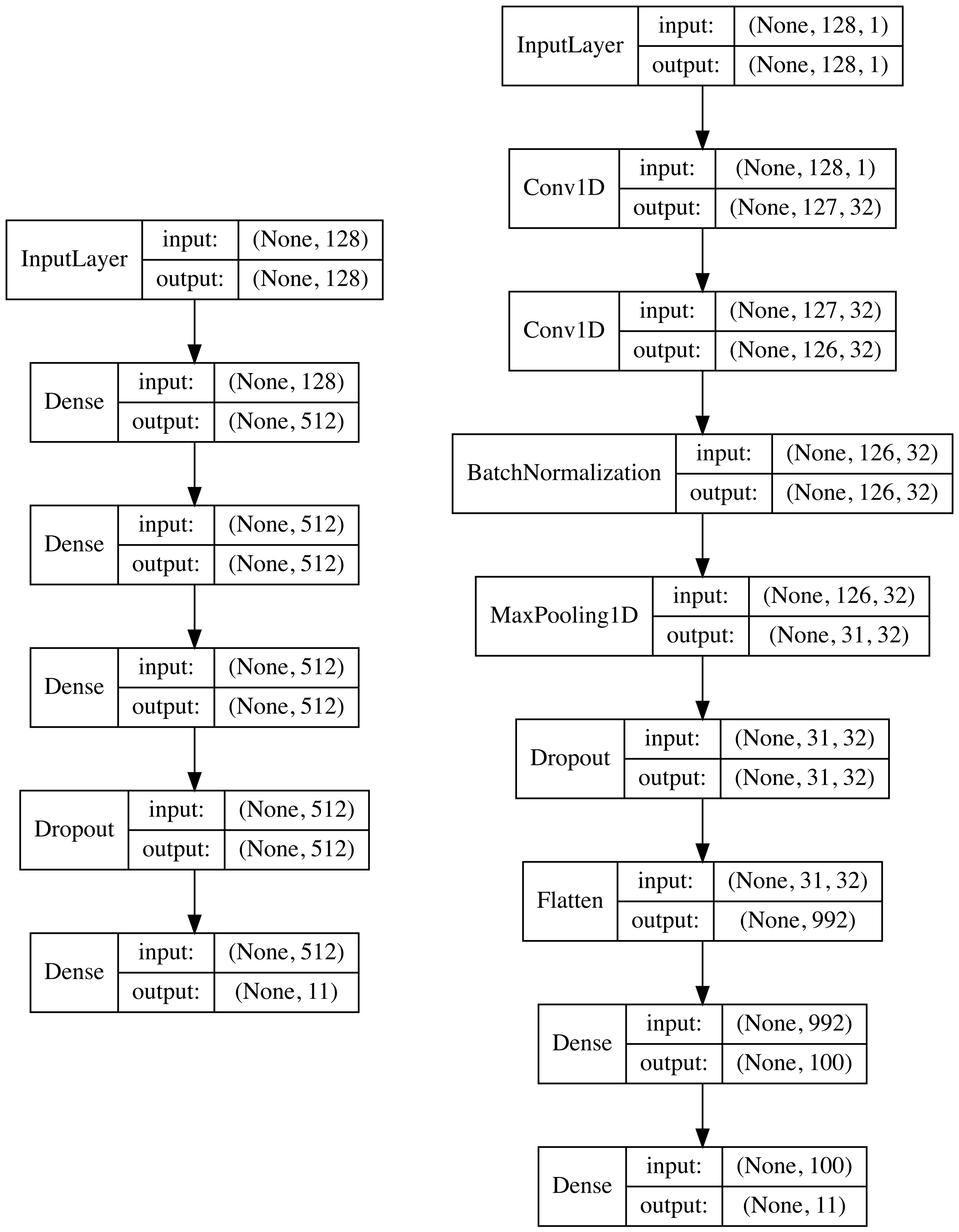

InputLayer input: (None, 128) output: (None, 128)
Dense input: (None, 128) output: (None, 512)
Dense input: (None, 512) output: (None, 512)
Dense input: (None, 512) output: (None, 512)
Dropout input: (None, 512) output: (None, 512)
Dense input: (None, 512) output: (None, 11)
InputLayer input: (None, 128, 1) output: (None, 128, 1)
Conv1D input: (None, 128, 1) output: (None, 127, 32)
Conv1D input: (None, 127, 32) output: (None, 126, 32)
BatchNormalization input: (None, 126, 32) output: (None, 126, 32)
MaxPooling1D input: (None, 126, 32) output: (None, 31, 32)
Dropout input: (None, 31, 32) output: (None, 31, 32)
Flatten input: (None, 31, 32) output: (None, 992)
Dense input: (None, 992) output: (None, 100)
Dense input: (None, 100) output: (None, 11)

Figure 14. The best architectures identified in iterative model evaluation processes. Input to the models consists of sequential CAN bus arbitration IDs. (*Left*) MLP. (*Right*) CNN.

Finally, we employ a 60/20/20 ratio to split each dataset into training, validation, and testing sets. This methodology applies to each version of the MLP and CNN. In total, we evaluate 27 MLPs and 18 CNNs. As Figure 15 shows, the final models possess sufficient capacity for the classification task. Finally, to evaluate the trained models, we present balanced accuracies and multiclass confusion matrices.

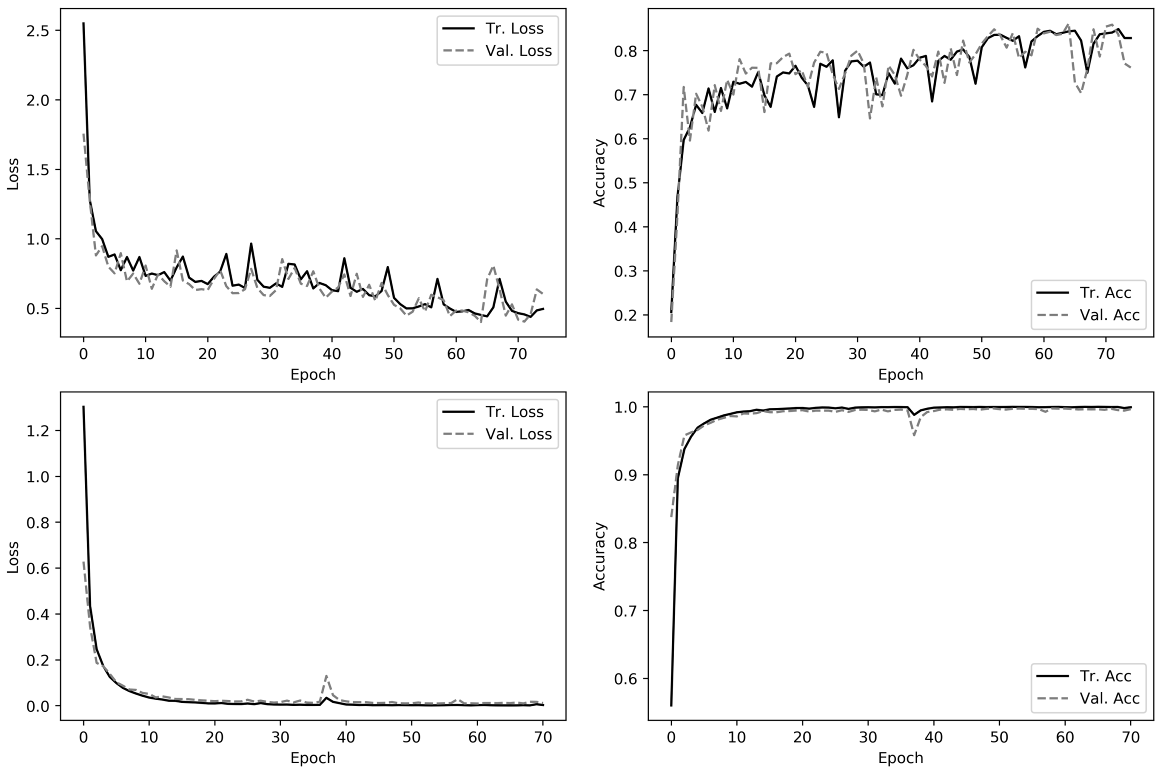


Figure 15. Training and validation loss and accuracy over time for the best models. Input to the models consists of sequential CAN bus arbitration IDs. (*Top left*) The MLP's loss. (*Top right*) The MLP's accuracy. (*Bottom left*) The CNN's loss. (*Bottom right*) The CNN's accuracy.

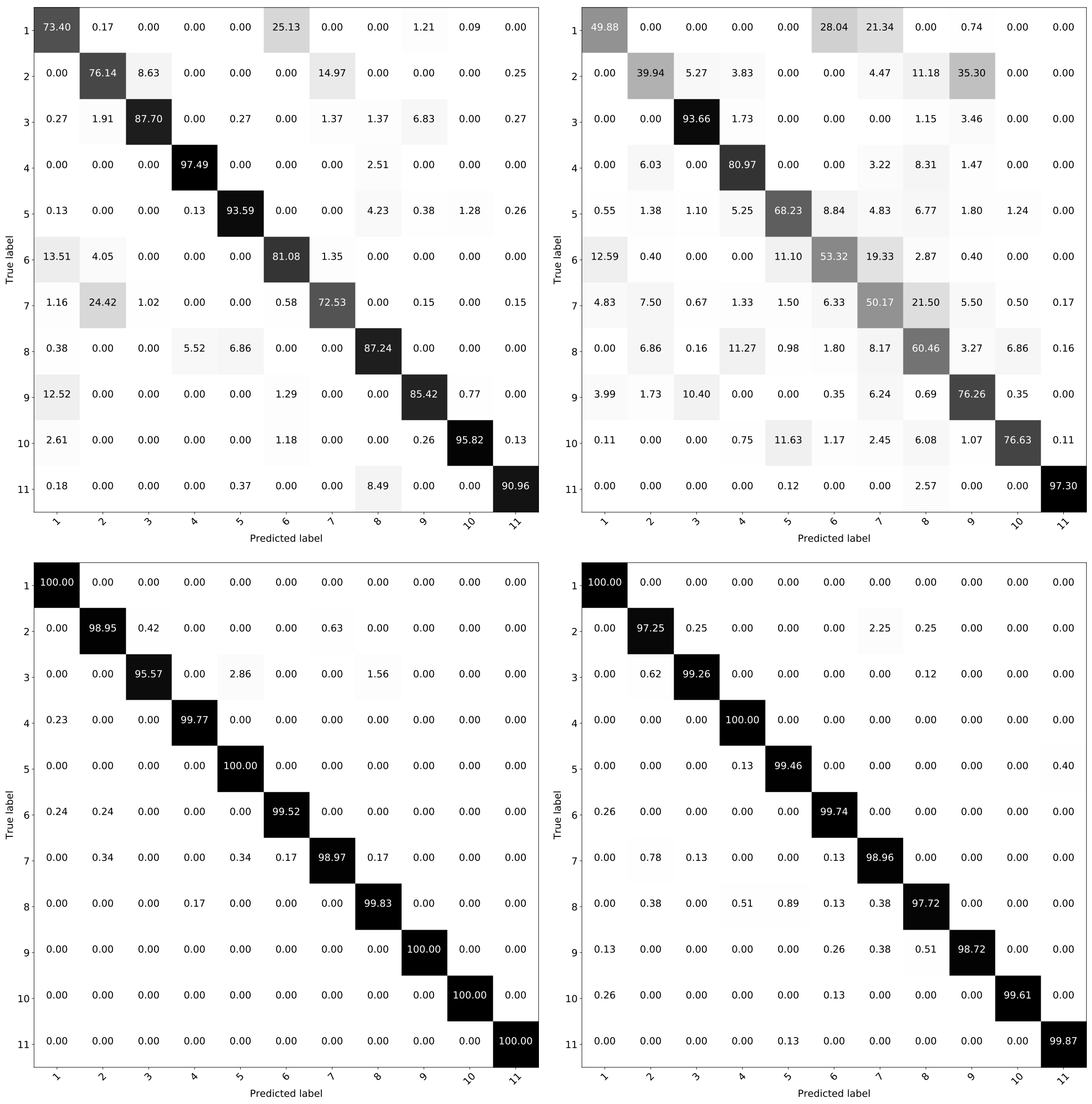
3.3.3 Results

As in the first experiment, results indicate that the MLP is a sufficient classifier for the task. On the imbalanced dataset, the MLP achieves a balanced classification accuracy above 80%. The CNN, however, surpasses 99% on both datasets, so it is once again the superior classification technique. Table 10 depicts the balanced accuracies for the four trained models; the results shown are nearly identical to those presented in Table 6. This supports the hypothesis that the deep learning models learn the structure of a vehicle's CAN data (as opposed to the data's exact nature). In the context of this experiment, this means that a bad cyber actor can feed simply the ArbIDs into a deep learning model, thus allowing for a smaller dataset, a shorter training period, and faster classification.

Table 10. Ordered ArbIDs — Balanced Accuracy

Model	**Dataset**	**Accuracy**
Multilayer Perceptron	Imbalanced	82.08 %
Multilayer Perceptron	Balanced	68.16 %
Convolutional Neural Network	Imbalanced	99.52 %
Convolutional Neural Network	Balanced	99.05 %

Table 11 reveals the class-specific accuracy values for the four trained models. It is clear that the CNNs are superior to the MLPs for all vehicles in both datasets; the MLPs do not achieve 98% accuracy on any class, and yet the CNNs surpass 99% accuracy for nearly every class. Similarly, Figure 16 depicts the four confusion matrices for the models. Interestingly, the MLP has significant performance differences when trained on different datasets. For example, misclassification relationships exist between Vehicles 1 and 7, 2 and 8, and 7 and 8 when the MLP is trained on the balanced dataset. When it is trained on the imbalanced dataset, these same relationships are not evident. In other cases, however, the MLP performs similarly on both datasets. For example, the models often mistake Vehicles 1 and 6 for each other. The same can be said about Vehicles 2 and 7. To a much lesser extent, one can draw the latter conclusion about the CNNs, but further analysis of misclassifications by the CNNs is made difficult by their phenomenal performances.



	1	2	3	4	5	6	7	8	9	10	11
1	73.40	0.17	0.00	0.00	0.00	25.13	0.00	0.00	1.21	0.09	0.00
2	0.00	76.14	8.63	0.00	0.00	0.00	14.97	0.00	0.00	0.00	0.25
3	0.27	1.91	87.70	0.00	0.27	0.00	1.37	1.37	6.83	0.00	0.27
4	0.00	0.00	0.00	97.49	0.00	0.00	0.00	2.51	0.00	0.00	0.00
5	0.13	0.00	0.00	0.13	93.59	0.00	0.00	4.23	0.38	1.28	0.26
6	13.51	4.05	0.00	0.00	0.00	81.08	1.35	0.00	0.00	0.00	0.00
7	1.16	24.42	1.02	0.00	0.00	0.58	72.53	0.00	0.15	0.00	0.15
8	0.38	0.00	0.00	5.52	6.86	0.00	0.00	87.24	0.00	0.00	0.00
9	12.52	0.00	0.00	0.00	0.00	1.29	0.00	0.00	85.42	0.77	0.00
10	2.61	0.00	0.00	0.00	0.00	1.18	0.00	0.00	0.26	95.82	0.13
11	0.18	0.00	0.00	0.00	0.37	0.00	0.00	8.49	0.00	0.00	90.96

	1	2	3	4	5	6	7	8	9	10	11
1	49.88	0.00	0.00	0.00	0.00	28.04	21.34	0.00	0.74	0.00	0.00
2	0.00	39.94	5.27	3.83	0.00	0.00	4.47	11.18	35.30	0.00	0.00
3	0.00	0.00	93.66	1.73	0.00	0.00	0.00	1.15	3.46	0.00	0.00
4	0.00	6.03	0.00	80.97	0.00	0.00	3.22	8.31	1.47	0.00	0.00
5	0.55	1.38	1.10	5.25	68.23	8.84	4.83	6.77	1.80	1.24	0.00
6	12.59	0.40	0.00	0.00	11.10	53.32	19.33	2.87	0.40	0.00	0.00
7	4.83	7.50	0.67	1.33	1.50	6.33	50.17	21.50	5.50	0.50	0.17
8	0.00	6.86	0.16	11.27	0.98	1.80	8.17	60.46	3.27	6.86	0.16
9	3.99	1.73	10.40	0.00	0.00	0.35	6.24	0.69	76.26	0.35	0.00
10	0.11	0.00	0.00	0.75	11.63	1.17	2.45	6.08	1.07	76.63	0.11
11	0.00	0.00	0.00	0.00	0.12	0.00	0.00	2.57	0.00	0.00	97.30

	1	2	3	4	5	6	7	8	9	10	11
1	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	98.95	0.42	0.00	0.00	0.00	0.63	0.00	0.00	0.00	0.00
3	0.00	0.00	95.57	0.00	2.86	0.00	0.00	1.56	0.00	0.00	0.00
4	0.23	0.00	0.00	99.77	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.24	0.24	0.00	0.00	0.00	99.52	0.00	0.00	0.00	0.00	0.00
7	0.00	0.34	0.00	0.00	0.34	0.17	98.97	0.17	0.00	0.00	0.00
8	0.00	0.00	0.00	0.17	0.00	0.00	0.00	99.83	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.00

	1	2	3	4	5	6	7	8	9	10	11
1	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	97.25	0.25	0.00	0.00	0.00	2.25	0.25	0.00	0.00	0.00
3	0.00	0.62	99.26	0.00	0.00	0.00	0.00	0.12	0.00	0.00	0.00
4	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.13	99.46	0.00	0.00	0.00	0.00	0.00	0.40
6	0.26	0.00	0.00	0.00	0.00	99.74	0.00	0.00	0.00	0.00	0.00
7	0.00	0.78	0.13	0.00	0.00	0.13	98.96	0.00	0.00	0.00	0.00
8	0.00	0.38	0.00	0.51	0.89	0.13	0.38	97.72	0.00	0.00	0.00
9	0.13	0.00	0.00	0.00	0.00	0.26	0.38	0.51	98.72	0.00	0.00
10	0.26	0.00	0.00	0.00	0.00	0.13	0.00	0.00	0.00	99.61	0.00
11	0.00	0.00	0.00	0.00	0.13	0.00	0.00	0.00	0.00	0.00	99.87

Figure 16. Confusion matrices. Input to the models consists of sequential CAN bus arbitration IDs. (*Top left*) MLP on the imbalanced dataset. (*Top right*) MLP on the balanced dataset. (*Bottom left*) CNN on the imbalanced dataset. (*Bottom right*) CNN on the balanced dataset.

Table 11. Ordered ArbIDs — Class Accuracy

Vehicle	MLP, Imbal.	MLP, Bal.	CNN, Imbal.	CNN, Bal.	Median
1	73.40 %	49.88 %	99.90 %	99.73 %	86.57 %
2	76.14 %	39.94 %	96.26 %	99.74 %	86.20 %
3	87.70 %	93.66 %	95.54 %	99.48 %	94.60 %
4	97.49 %	80.97 %	99.77 %	99.74 %	98.62 %
5	93.59 %	68.23 %	100.00 %	99.62 %	96.60 %
6	81.08 %	53.32 %	99.00 %	99.87 %	90.04 %
7	72.53 %	50.17 %	97.61 %	99.87 %	85.07 %
8	87.24 %	60.46 %	99.81 %	98.49 %	92.86 %
9	85.42 %	76.26 %	99.85 %	98.81 %	92.12 %
10	95.82 %	76.63 %	100.00 %	100.00 %	97.91 %
11	90.96 %	97.30 %	100.00 %	100.00 %	98.65 %

3.4 Classifying Unordered CAN Payload Data

Section 3.2 shows that deep learning tools can identify which vehicle from a set generated a given segment of ordered CAN data. Section 3.3 demonstrates the same capability but for segments of ordered ArbIDs. These abilities may be because the deep learning models identify the underlying structure of the CAN data—that is, the models identify the payload organization and frequencies unique to each of an OEM's vehicles. This may imply that the *sequential* structure of the CAN segments analyzed in Sections 3.2 and 3.3 is irrelevant. For this reason, this experiment evaluates whether randomized CAN payload data can effectively fingerprint a vehicle. Here, *randomized* means that we simply shuffle each list of hex bytes described in Section 3.2 before splitting the list into samples. Although a bad cyber actor is unlikely to process collected data in such a way, the results presented in Section 3.4.3 illustrate that the deep learning models do seem to identify the CAN structure, and this thus provides valuable insight into the nature of CAN data and design.

Because the quantity of data in each capture does not change, the generated CSV file contains 28,425 samples, the same number as the file generated in the first experiment. Likewise, the sample distribution over the 11 classes for this experiment is identical to the first experiment's distribution. Table 12 depicts this distribution, and Table 13 presents a few examples of the randomized CAN samples. Once again, we address the obvious class imbalance during model training. As expected, it is difficult to discern differences between these samples and those presented in Table 2 because the data formatting process does not alter the data bytes themselves—only the order in which the bytes appear.

Table 12. Unordered Data — Samples Per Vehicle

Vehicle	Samples	Proportion
1	4115	14.48 %
2	1856	6.53 %
3	1812	6.37 %
4	2221	7.81 %
5	3806	13.39 %
6	1709	6.01 %
7	2220	7.81 %
8	3041	10.70 %
9	2564	9.02 %
10	2980	10.48 %
11	2101	7.39 %

Table 13. Unordered Data — Example Samples

Vehicle	Unordered Data
3	255 000 015 007 000 017 000 ...
7	010 254 000 127 249 000 000 ...
8	040 000 123 000 000 000 144 ...

3.4.1 Model Architecture

Like before, we compare the MLP's performance to that of a CNN. As in the first experiment, the models each receive as samples 1,024 three-digit hexadecimal data bytes extracted from CAN messages. The models classify each sample as one of the 11 vehicles described in Section 3.1. We iteratively construct these models using the parameters listed in Tables 4 and 5. The optimal MLP has three 256-node hidden layers that each utilize a `relu` activation function. These layers are followed by a 20% dropout layer and an 11-node `softmax` output layer. It is compiled with an Adam optimizer of learning rate $lr = 0.001$ and a categorical cross-entropy loss function and has 396,811 trainable parameters. The best CNN utilizes two one-dimensional convolutional layers, each of which uses 32 filters of size two; one pooling layer, which pools over two elements at a time; and one dropout layer, which zeroes 20% of its inputs. The output layer uses a `softmax` activation function, and the rest of the layers use a `relu` activation function. The model monitors categorical cross-entropy loss and employs an Adam optimizer of learning rate $lr = 0.001$. It has 1,091,515 parameters, only 64 of which are non-trainable. The left and right panels of Figure 17 depict the architectures for the optimal MLP and CNN, respectively.

3.4.2 Model Fitting

The classes in the dataset are disproportionately represented; Table 12 shows the exact sample counts for each class. We employ the same methods as before—namely, the `class_weight` module and the creation of a second, balanced dataset—to mitigate the effects of this imbalance. We again utilize Keras's `EarlyStopping` callback to reduce overfitting, and we also split the dataset into training, validation, and testing sets before evaluating each of the 27 MLP configurations and 18 CNN configurations using the parameters depicted in Tables 4 and 5, respectively. Figure 18 shows that the final versions of each model type possess sufficient capacity to effectively classify CAN samples.

3.4.3 Results

Once again, the experimental results indicate that both model types can adequately classify CAN samples. However, the performances in this experiment are noticeably worse than in either of the previous experiments. As Table 14 illustrates, the optimal MLP achieves 59.26% balanced classification accuracy, far lower than the 94.46% and 82.08% accuracies achieved in the first two experiments.[8] Similarly, training the models on ordered data or ArbIDs allows for a balanced accuracy of 99.64% or 99.52%,

[8] All accuracy values refer to the given model's performance on the imbalanced dataset.

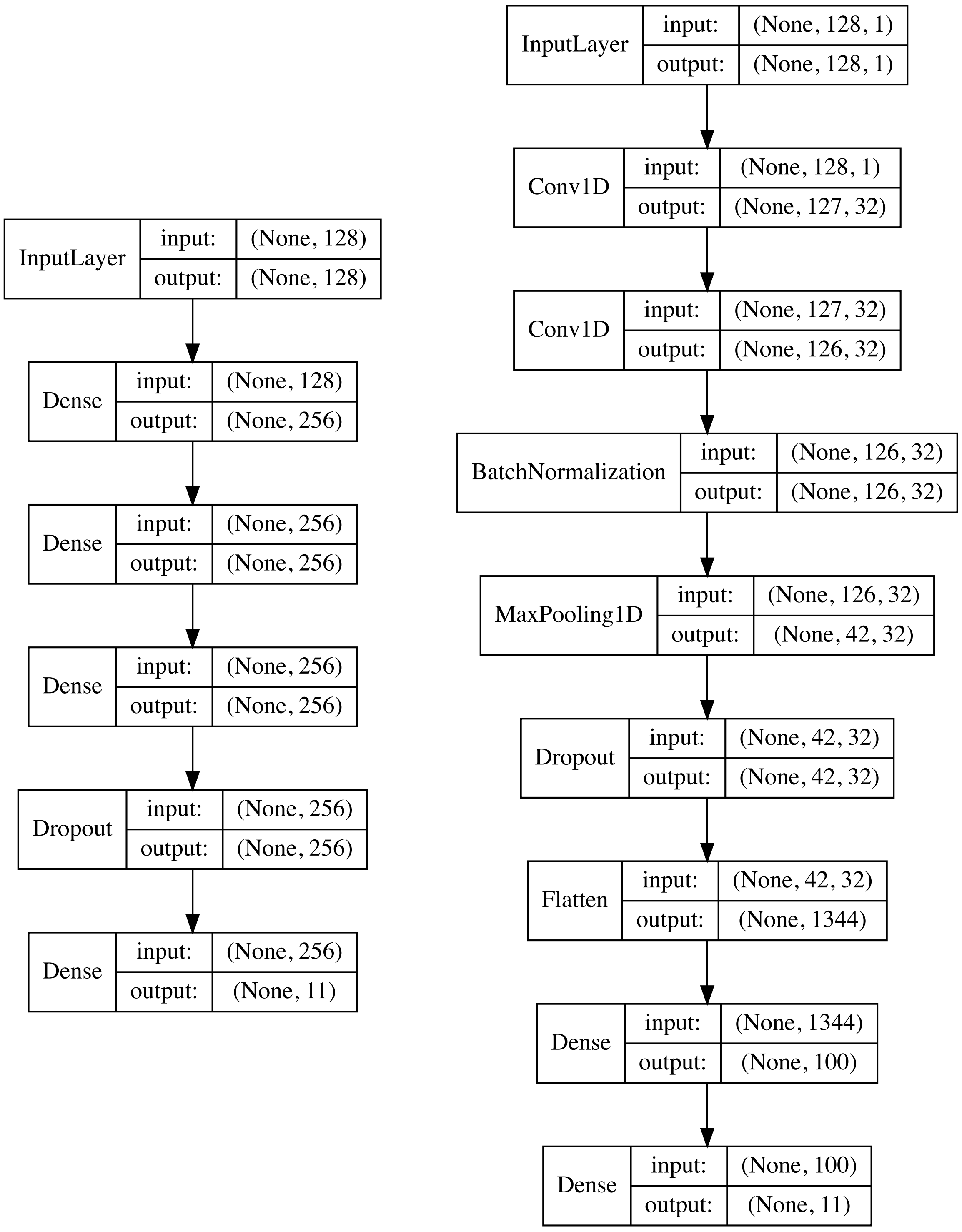


Figure 17. The best architectures identified in iterative model evaluation processes. Input to the models consists of CAN bus data in a random order. (*Left*) MLP. (*Right*) CNN.

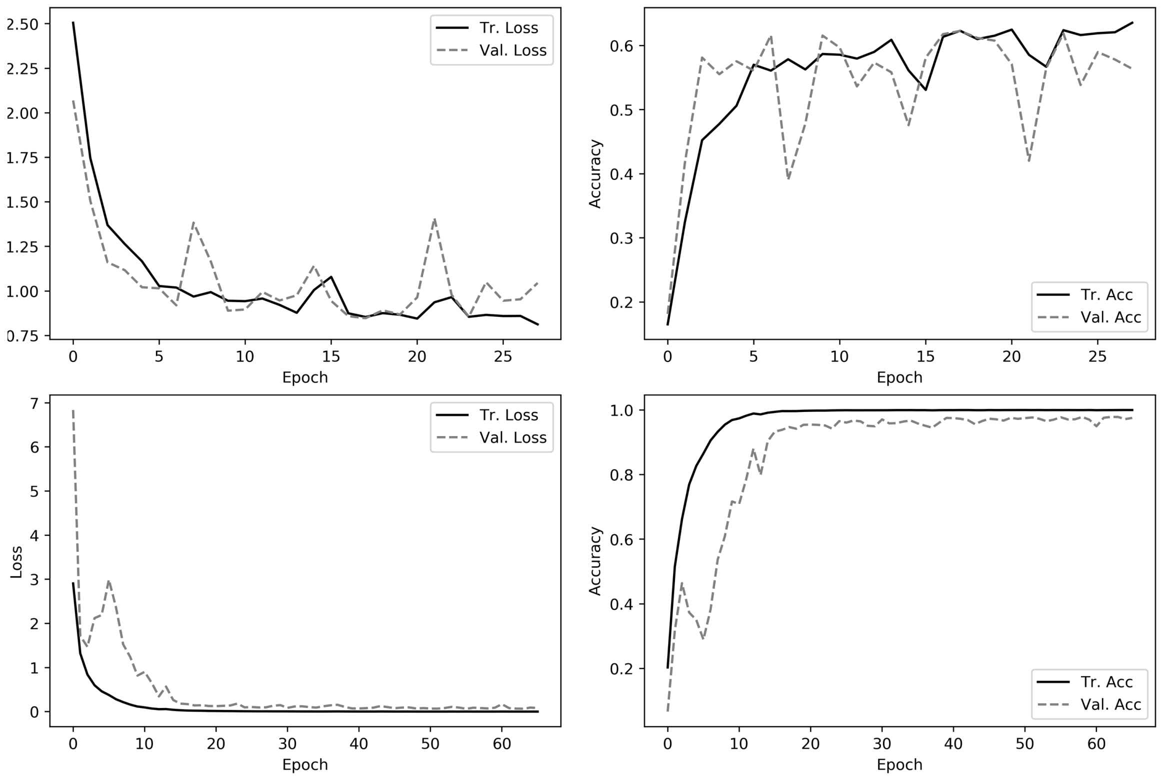


Figure 18. Training and validation loss and accuracy over time for the best models. Input to the models consists of CAN bus data in a random order. (*Top left*) The MLP's loss. (*Top right*) The MLP's accuracy. (*Bottom left*) The CNN's loss. (*Bottom right*) The CNN's accuracy.

respectively, for the CNN; when trained on unordered data, the CNN's performance dips slightly to 98.83%. Table 15 depicts the models' class accuracy for each vehicle. Both versions of the MLP suffer on several classes, but the CNN, when trained on the full dataset, attains 96.98% or better classification accuracy for every vehicle. Finally, Figure 19 contains the confusion matrices for the four models. Clearly, the MLP trained on the full dataset performs well on some vehicles, like 3, 6, and 9, but the other MLP only demonstrates strong performance on Vehicle 3. Comparatively, both MLPs perform relatively poorly on a majority of the vehicles, including 2, 3, 5, 7, 8, and 10.[9] Certainly, both CNNs surpass either MLP. When trained on the balanced

[9]It is important to note that both MLPs still outperform a random guess approach, which should only correctly classify about 9% of the samples from a given dataset.

dataset, the CNN achieves 80% accuracy on four classes. The CNN trained on the full dataset benefits from its increased model capacity and quantity of training data, because it exceeds 96% accuracy on every vehicle.

Table 14. Unordered Data — Balanced Accuracy

Model	**Dataset**	**Accuracy**
Multilayer Perceptron	Imbalanced	59.26 %
Multilayer Perceptron	Balanced	42.04 %
Convolutional Neural Network	Imbalanced	98.83 %
Convolutional Neural Network	Balanced	75.22 %

Table 15. Unordered Data — Class Accuracy

Vehicle	**MLP, Imbal.**	**MLP, Bal.**	**CNN, Imbal.**	**CNN, Bal.**	**Median**
1	75.46 %	29.00 %	96.98 %	87.25 %	81.36 %
2	38.96 %	25.11 %	100.00 %	85.23 %	62.10 %
3	92.80 %	95.69 %	100.00 %	75.67 %	94.25 %
4	0.00 %	33.46 %	100.00 %	74.95 %	54.21 %
5	52.52 %	28.81 %	99.74 %	54.27 %	53.39 %
6	89.14 %	46.27 %	97.94 %	64.80 %	76.97 %
7	46.15 %	50.00 %	98.65 %	70.30 %	60.15 %
8	46.08 %	49.91 %	100.00 %	79.08 %	64.50 %
9	75.39 %	52.72 %	99.18 %	81.04 %	78.21 %
10	49.41 %	36.64 %	98.18 %	73.79 %	61.60 %
11	68.03 %	46.27 %	98.77 %	89.98 %	79.00 %

These results—especially for the CNN—show that deep learning models can effectively classify unordered CAN data. This implies that the models do in fact learn to identify structural characteristics of the data, like the commonly used byte values and their frequencies. As stated at the start of this section, someone hoping to fingerprint vehicles with CAN data is not likely to randomize the bytes as we did. However, in identifying why the models still correctly classify unordered samples, one enables a deeper analysis of an OEM's CAN design choices and tendencies.

3.5 Distinguishing CAN Data with Siamese Neural Networks

We also briefly explored Siamese neural network (SNN) applications to this work. Given enough training data, an SNN can distinguish new cars—that is, those not

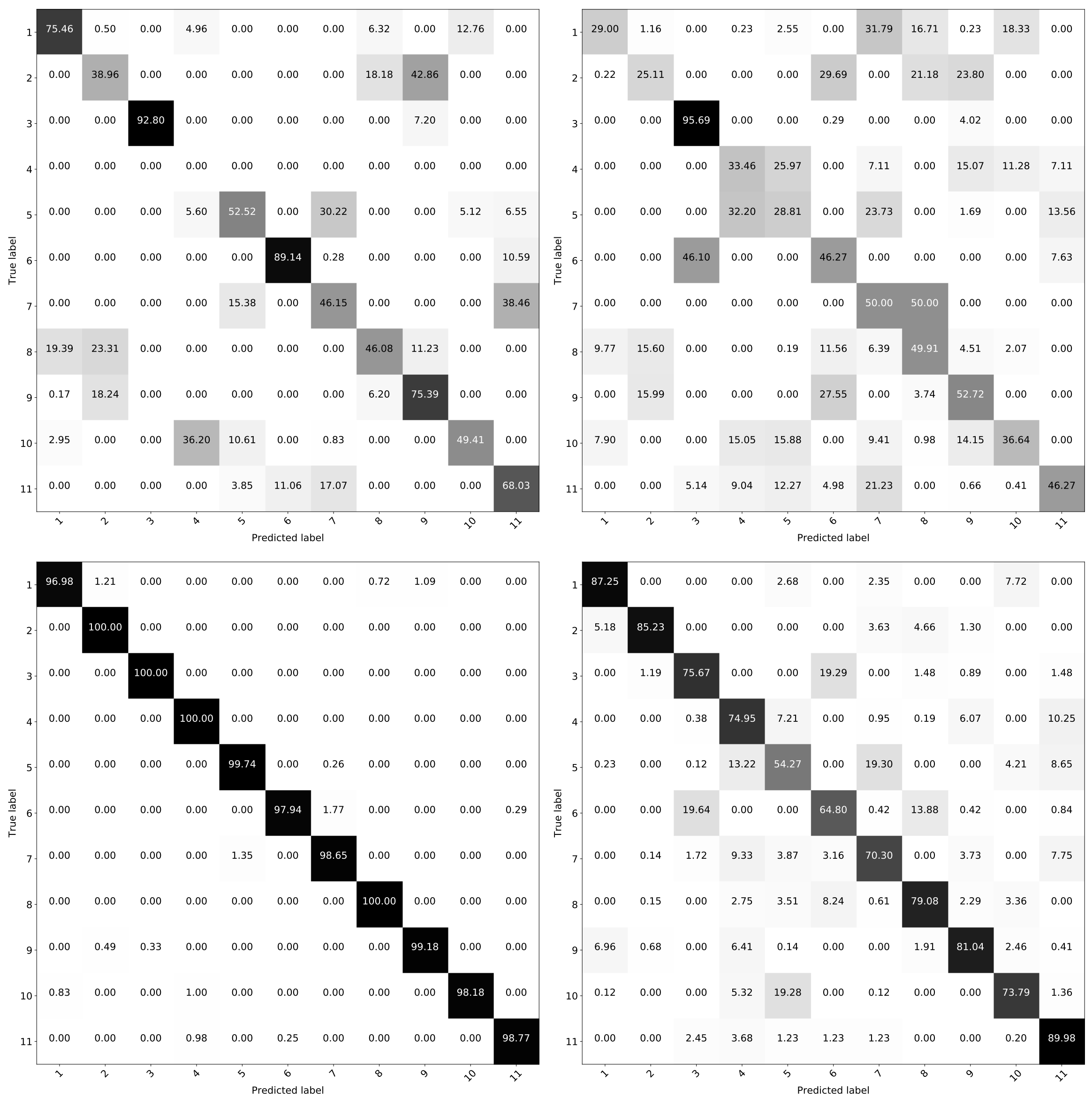


Figure 19. Confusion matrices. Input to the models consists of CAN bus data in a random order. (*Top left*) MLP on the imbalanced dataset. (*Top right*) MLP on the balanced dataset. (*Bottom left*) CNN on the imbalanced dataset. (*Bottom right*) CNN on the balanced dataset.

present in the original training set—because it learns during training whether two observations come from the same source or from different sources. However, our attempts did not prove fruitful. It seems that an SNN with a contrastive loss function does not possess the capacity required to determine *same/not same* for two CAN samples. We contend that a network with a triplet loss function may succeed, but, in our exploratory efforts, the loss consistently centered around the margin α for all network configurations. Although a more in-depth analysis of this topic is outside the scope of this thesis, future researchers may wish to advance these efforts. They should consider the following modifications:

- Utilize a batch-hard triplet generation methodology to ensure the triplets sufficiently train the model (i.e., so the triplets are nontrivial); and
- Implement a weighted loss function to penalize those losses that collapse to α.

Demonstrating that an SNN can effectively fingerprint CAN samples confirms the security and privacy risks to the vehicle's operator and passengers. Specifically, because an SNN is more powerful than the MLPs and CNNs we evaluated, and because this research already illustrates the risks present in the network's design, an SNN that can fingerprint vehicles is simply a more robust tool for the CAN intruder.

3.6 Summary

This chapter discussed a corpus of CAN data and three separate experiments, each of which demonstrated an effective classification approach for these data. Section 3.2 described the first experiment, in which we generated 1,024-byte samples of ordered payload data for each of 11 vehicles. Section 3.3 examined the second experiment, in which we generated samples consisting of 128 ordered ArbIDs. Section 3.4 considered the third experiment, in which we extracted the ordered payload data, as in the first experiment, and then shuffled the data bytes before partitioning them into 1,024-byte samples. In each experiment, we constructed and trained multiple versions of two disparate deep learning classifiers, one an MLP and the other a CNN, before using these classifiers to identify which of the 11 vehicles generated each sample. In all cases, the experimental results indicated that deep learning models can effectively classify the samples. Section 5.1 discusses the real-world implications of these findings.

IV. EDM as a Basis for an Intrusion Detection System

Chapters I and II introduced intrusion detection systems (IDSs), which mitigate cyber-physical system (CPS) vulnerabilities like those discussed in Chapter III. This chapter discusses how two different time series analysis techniques can contribute to IDS development. The first, Granger causality, is a well-known, relatively simple statistical method for quantifying the causality in a system. The second, empirical dynamic modeling (EDM), is a small-but-growing technique for analyzing the dynamics, to include the causality, of a system. The remainder of this chapter discusses the nature and origins of the data. It then describes the computational methodology of Granger causality and EDM before finally analyzing the results of applying these techniques to the data.

4.1 Experimental Data & Analysis

We utilize two statistical analysis tools to develop insights into a system's characteristics, including its nonlinearity, deterministic chaos, and causality. This section describes the experimental data and the techniques used to analyze the data.

4.1.1 Data

This research utilizes data from two simulated CPSs. The first simulated dataset represents the effect of an automobile's steering wheel angle over time on the revolutions per minute (RPM) measurements for the turning wheels. The second dataset consists of captured nonlinear data generated by the Avionics Vulnerability Assessment System (AVAS), an Air Force Research Lab (AFRL)-developed flight simulator that employs real-world physics and flight dynamics for research purposes. The steering dataset is considered to be linear because the relationship between each pair of time series is linear or nearly linear. Specifically, the relationship between the two wheel RPMs is linear, and the relationships between the steering input and each wheel RPM are almost linear. Similarly, the AVAS dataset is nonlinear because the relationships between the time series are nonlinear. The datasets allow us to evaluate the utility of both Granger causality and EDM when used to analyze linear systems and nonlinear systems. Although simpler methods may enable effective analysis of linear systems, many CPSs of interest are nonlinear.

4.1.1.1 Linear Data

To properly assess both Granger causality and EDM, we construct a dataset representative of a linear system. It is expected that the analysis techniques make better predictions and identify stronger relationships when evaluating time series that are linearly linked. The variables describing the steering wheel angle and RPMs of the turning wheels in a passenger vehicle constitute such a system. We generate this time series with Python 3.7 for a vehicle with the following characteristics: a 30-inch wheel radius, including the tire; a 72-inch wheelbase; a 60-inch track; a maximum steering wheel turning angle of 360 degrees; a steering ratio of 8:1 (and thus a maximum wheel angle of 45 degrees); and a constant forward speed of 25 miles per hour.

Under these assumptions, a sum of sines function loosely represents some hypothetical driving scenario. That is, the solid line in Figure 20 serves as a potential steering wheel angle time series. The other two time series, which we directly compute given the instantaneous steering wheel angle, indicate the left and right wheel RPMs. This steering system is rather rudimentary—it doesn't account for the physical properties of a real system, including the effects of other relevant variables—but even its simplistic nature may allow us to draw conclusions concerning EDM's applications to linear CPSs. The empirical results shown in Figure 21 confirm that the time series from the model are fairly linearly related.[10] Because the values of the variables cover significantly different ranges, we standardize all variables with R's `scale` function[11] to ensure each is equally important during analysis.

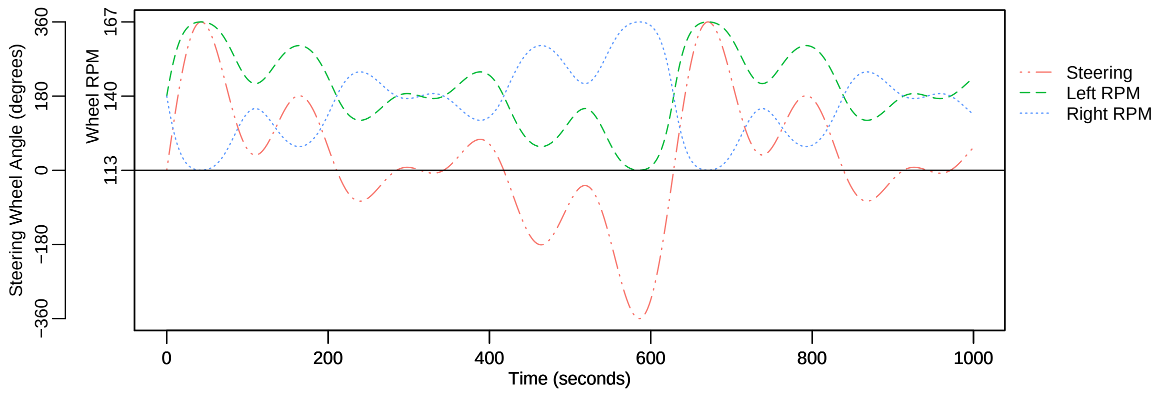


Figure 20. Plots of the steering system time series.

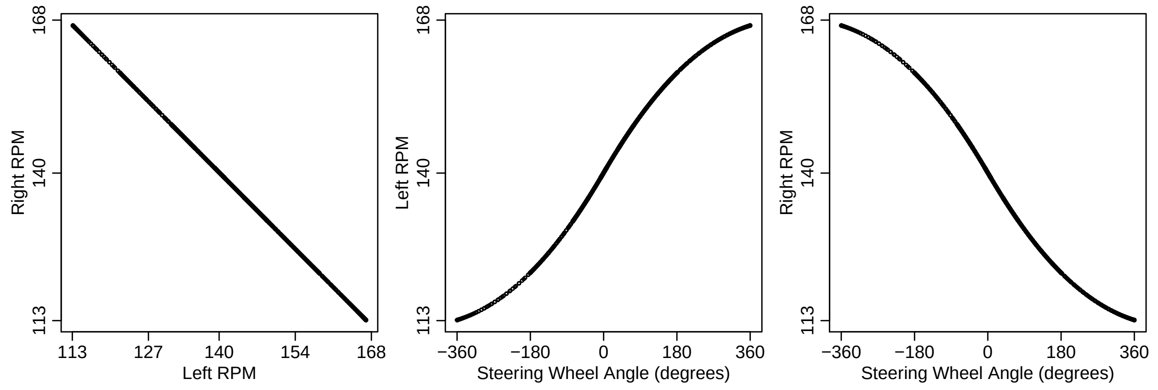


Figure 21. Scatter plots demonstrating the relationship between each pair of variables in the steering system time series.

[10]The somewhat nonlinear behavior between either wheel and the steering wheel is due to the mechanics of a standard automobile's Ackermann steering mechanism.

[11]For some time series X, the function z-scales X by subtracting its mean and then dividing it by its standard deviation.

4.1.1.2 Nonlinear Data

To create the second dataset, we guided an AVAS-simulated aircraft through takeoff, low-altitude cruising, and multiple shallow banked turns. Our data collection yielded 7,582 observations from a 14-minute flight. Each observation includes seven different flying metrics and a timestamp relative to the start of the simulation. The metrics are roll and pitch (each in radians),[12] altitude (in feet), and airspeed and velocity in each of the three coordinate axes (in feet per second). The roll and pitch values range from $-\pi$ to π, the altitude and airspeed are both greater than zero, and the velocities are floating point values.

As with the linear dataset, we z-scale the variables prior to analysis. We then select a subset of the variables—airspeed, altitude, and pitch—before conducting the analyses. Other subsets of the eight variables likely exhibit the desired dynamics, but it is expected that these three variables best demonstrate a tightly coupled system. Figure 22 presents the three time series, prior to scaling, in one plot. It is important to note that scaling the variables does not guarantee the successful application of Granger causality analysis; the technique assumes the variables under analysis exhibit linear dynamics, but Figure 23 clearly illustrates that the system is highly nonlinear.

4.1.2 Granger Causality

StatsModels's `grangercausalitytests`, which evaluates the null hypothesis "X does not Granger-cause Y", quantifies the Granger non-causality present in each system. Input to this function includes the two time series to be evaluated, of course, but it also includes a `maxlag` parameter, which indicates the maximum number of time steps to lag, or offset, the X time series. The function computes the system's Granger non-causality for all lag values from one to `maxlag` (inclusive). Output from

[12] We exclude yaw because, in the AVAS, yaw is simply a measurement of the plane's heading relative to north. In other words, it is not a characteristic of the plane's dynamics.

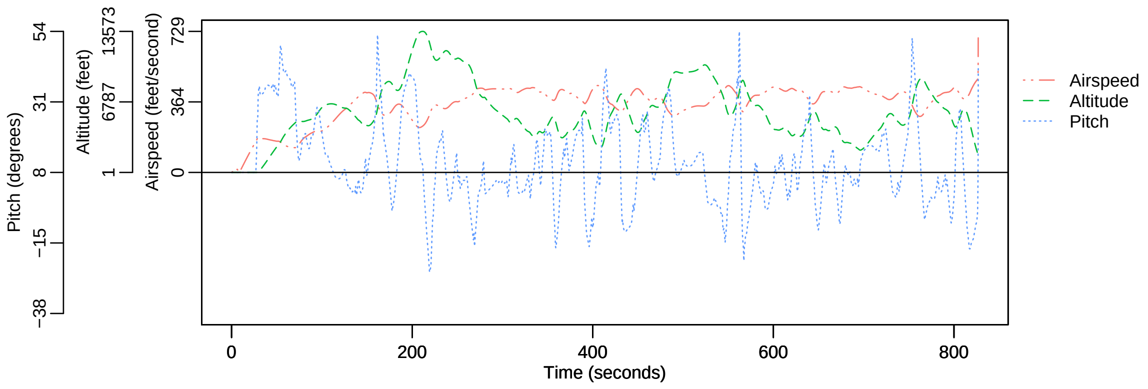


Figure 22. Plots of the selected AVAS time series.

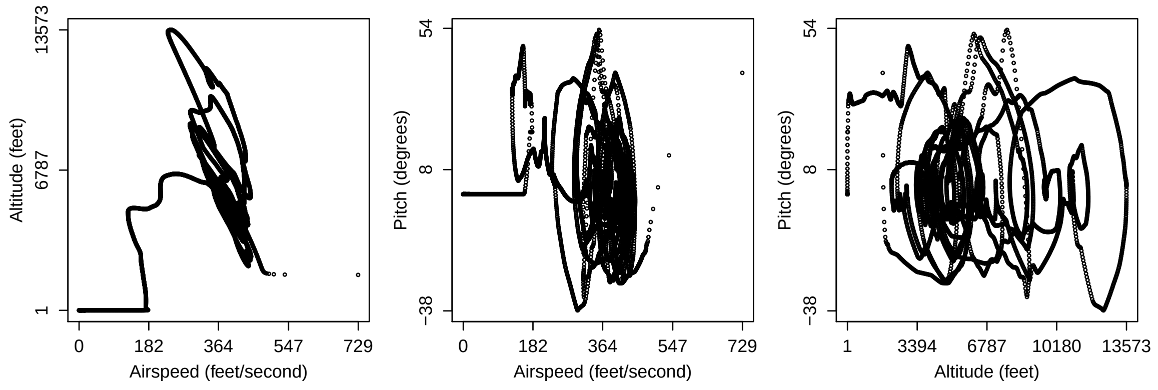


Figure 23. Scatter plots demonstrating the relationship between each pair of variables in the AVAS time series.

this function includes the statistical p-values concerning the results of four different tests for Granger non-causality; if the average p-value is less than 0.05, we reject the null hypothesis and assert that X Granger-causes Y [46].

4.1.3 EDM Techniques

Ye et al. suggest the following sequence of EDM techniques to best interpret a dataset's characteristics [30]:

1. Conduct nearest neighbor forecasting via simplex projection to identify the embedding dimension E which maximizes the prediction skill ρ [47];

2. Use simplex projection and E to determine whether the system exhibits deterministic chaos;

3. Employ sequential locally weighted global linear maps (S-maps) to characterize any nonlinearity present in the data [48]; and

4. Utilize convergent cross mapping (CCM) to generate shadow manifolds, evaluate predictive accuracy, and quantify causality [31].

In essence, "simplex projection is the process of iteratively selecting [a point] Y_t in a shadow manifold and b other points whose histories over time t are most similar to the currently selected point ... A simplex is a generalization of a triangle or tetrahedron to an arbitrary number of dimensions" [12, 47, 49]. One then uses the weighted average of the future values of the b other points to make predictions about future values of Y_t. The difference between these predictions and the actual future values gives a forecast skill ρ. By repeating this process with shadow manifolds of different dimensionalities, one can identify the embedding dimension E that optimizes ρ [50]. The (strong) Whitney embedding theorem says the following [51]:

Theorem. *Any m-manifold of class C^R ($r \geq 1$ finite or infinite) may be imbedded* [sic] *by a regular C^r-map in E^{2m}, and by such a map in a one-one manner in E^{2m+1}.*

In simpler terms, the theorem states that the embedding dimension E for an attractor manifold has an upper bound of $2D + 1$, where D is the true dimension (the number of variables) of the system [12, 50]. One can thus use simplex projection to definitively identify the optimal E in a finite amount of time.

S-map projection is another iterative process, but it instead uses all neighboring points to create linear regression vectors. By aggregating these regression vectors, one approximates an n-dimensional spline. One then compares this spline to the shadow manifold attractor to measure ρ [48, 49, 50]. When generating the regression estimates, a nonlinear tuning parameter θ weights the neighbors with respect to their distance to the current focal point Y_t. Finally, "if ρ is maximized when $\theta = 0$, then the time series may be assumed to belong to a simple linear system instead of a dynamic system" [12, 48, 50].

As Stone et al. claim, "This process provides insight into the true dimensionality of the dynamic system responsible for generating [observational] data without requiring complete understanding of the system itself" [12]. Accurate knowledge of E is a prerequisite to effectively applying CCM to multiple time series to detect causality. Alternatively, a proper S-map analysis of time series relationships may indicate whether these relationships belong to a simple linear system. If so, computationally simpler methods, like Granger causality or auto-regressive linear models, could replace the more complex CCM technique in detecting causality [1, 30, 48]. Finally, knowledge of the dimensionality of a system may assist in creating a high quality model of said system. Such a model—and the results of a causality analysis—likely enables an effective IDS for various CPSs.

To conduct this analysis, we use the Sugihara Laboratory's rEDM repository on GitHub. This codebase enables EDM analysis using the R programming language. The repository includes the following functions (among others):

- `simplex`, which corresponds to the first and second EDM techniques;
- `s_map`, which corresponds to the third EDM technique; and
- `ccm` and `ccm_means`, which correspond to the fourth EDM technique.

These functions, together with a few helper functions, facilitate effective EDM analysis. Appendix A details the code used to apply EDM to the steering dataset and to plot the results of the analysis. Section 4.2 depicts the results of the EDM and Granger causality analyses. Section 5.2 discusses the implications of these results. For the interested reader, Rennie provides an in-depth description of EDM, to include the mathematics behind simplex projection, S-map analysis, and CCM [52].

4.2 Results

This section presents the results of the Granger causality and EDM analyses for both datasets. Sections 4.2.1 and 4.2.2 show that, for a linear dataset, Granger causality enables a somewhat effective causality analysis but that EDM does not. Conversely, for nonlinear data, Sections 4.2.3 and 4.2.4 present an ineffective use of Granger causality and a more effective use of EDM.

4.2.1 Linear Data — Granger Causality

Figure 24 depicts, for 20 different lags, the Granger causality p-value between each pair of time series in the linear dataset. Each p-value is the average of the p-values for four different Granger causality tests for a specific lag parameter. For example, when steering wheel angle is lagged by one second (relative to left wheel RPM), the average p-value when using left wheel RPM to predict steering wheel angle is 0.7818. We only reject the null hypothesis[13] when $p < 0.05$, so this means that left wheel RPM does not Granger-cause a one-second-delayed steering wheel angle.

Figure 24 also shows the limitations of Granger causality. For example, according to the figure, steering wheel angle Granger-causes both wheel RPMs when the RPM series is lagged by two or more seconds. This is somewhat consistent with a standard automobile's dynamics. However, it is not expected that the RPMs cause steering wheel angle, but Granger-causality asserts that they do when the steering angle is lagged by two, three, or eight to twelve seconds. This is a limitation of the technique; the result is simply due to the similarity of the time series and is not actually indicative

[13]As a reminder, the null hypothesis is "X does not Granger-cause Y".

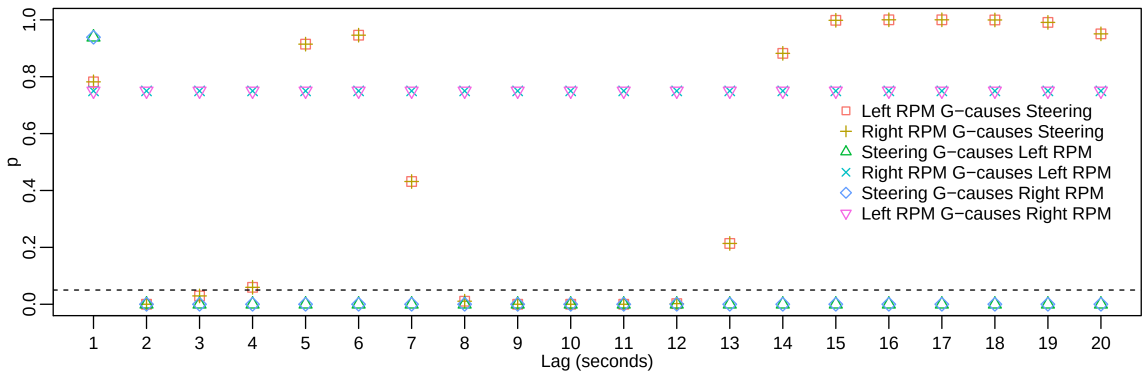


Figure 24. Granger causality between each pair of steering system time series.

of the steering system's relevant dynamics. The results do provide some insight in general, but this insight is likely not sufficient for suitable IDS design.

4.2.2 Linear Data — EDM

To effectively apply CCM to make predictions and quantify causality, we require knowledge of the optimal embedding dimension E for each time series in the system. By iteratively utilizing simplex projection to quantify predictive accuracy at different values for E, we identify the optimal value. Figure 25 illustrates the results of this process for each steering system time series. Although the differences in forecast skill, or ρ,[14] are difficult to discern, numerically, ρ is maximized for each time series when $E = 2$. However, it is important to note that, because the values are all so similar, the choice of E is largely irrelevant in this particular case. Still, we let $E = 2$ for the remainder of the EDM analysis techniques. To be clear, this means the techniques construct a two-dimensional shadow manifold, where each dimension is a time series lagged by some multiple of τ. When predicting steering wheel angle, for example, EDM constructs a shadow manifold using steering wheel angle and one copy of steering wheel angle, where the copy is lagged by τ. For this dataset, we let τ equal one second.

[14]Forecast skill is a measure of the ability to forecast future values of a given time series.

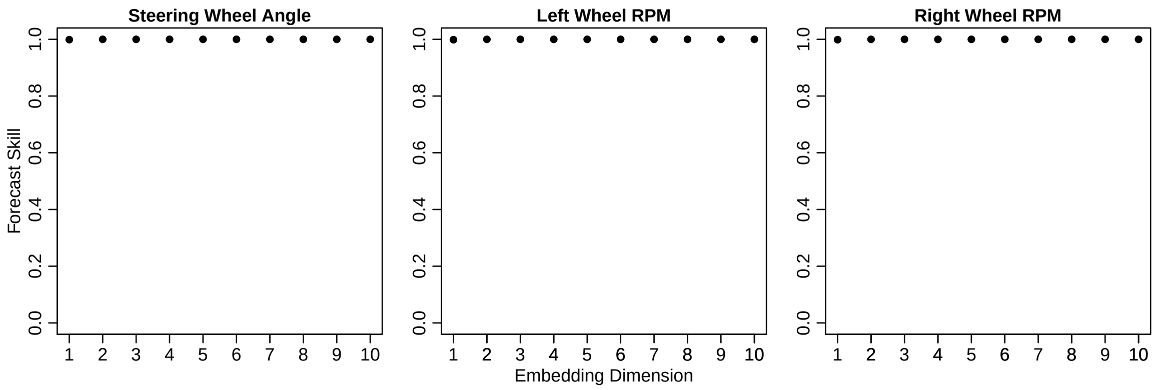


Figure 25. Plots illustrating the optimal embedding dimension for each steering system time series.

If we keep E constant and vary the time to prediction tp, simplex projection enables an analysis of a system's deterministic chaos (or lack thereof). Figure 26 shows how ρ remains constant as tp increases for each of the three time series. In other words, the ability to make predictions further in the future is essentially the same as for those closer in time, which indicates non-chaotic behavior for the three variables. This is due to the nature of the simulated steering system: the steering wheel angle consists of a set of fairly predictable points over time, and the left and right wheel RPMs are directly computed from those points. For this reason, the system is not likely to exhibit chaotic behavior, and EDM confirms this. EDM does not enable a deeper analysis of the system's chaos.

S-map analysis fits local linear maps to the system to describe its nonlinearity. This is different from simplex projection, which analyzes each point's nearest neighbors. By varying the nonlinearity tuning parameter θ in the S-map function call and plotting against ρ, we obtain the plots shown in Figure 27. When $\theta = 0$, S-map equally weights all points; as θ increases, the function more heavily weights points close to the point under analysis. Thus, when θ is higher, the function assumes more nonlinearity in the system. However, for all three variables and for every value of θ, ρ is effectively equal to one. This indicates the absence of nonlinearity in each time series. For this linear system, it seems that EDM's nonlinearity analysis is not particularly useful. It is possible that the analysis does pertain to more robust linear systems, but this requires further research.

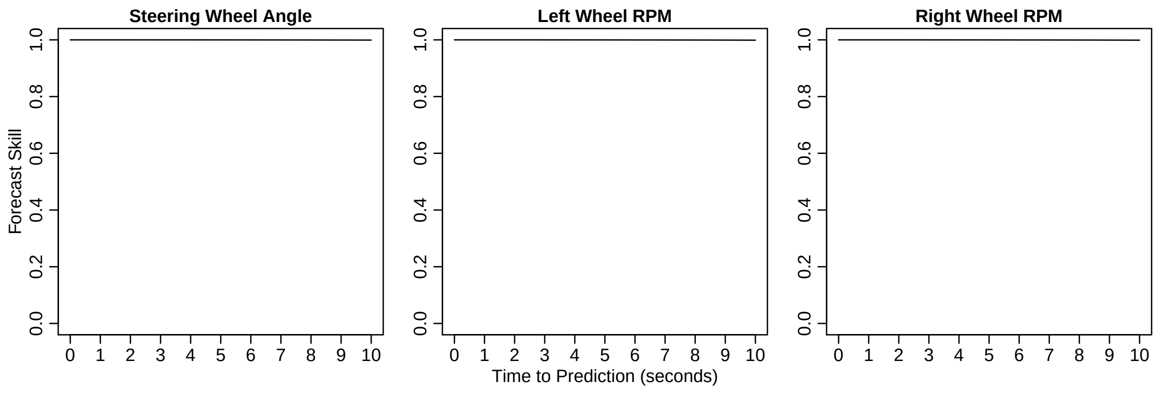


Figure 26. Plots illustrating the deterministic chaos present in each steering system time series.

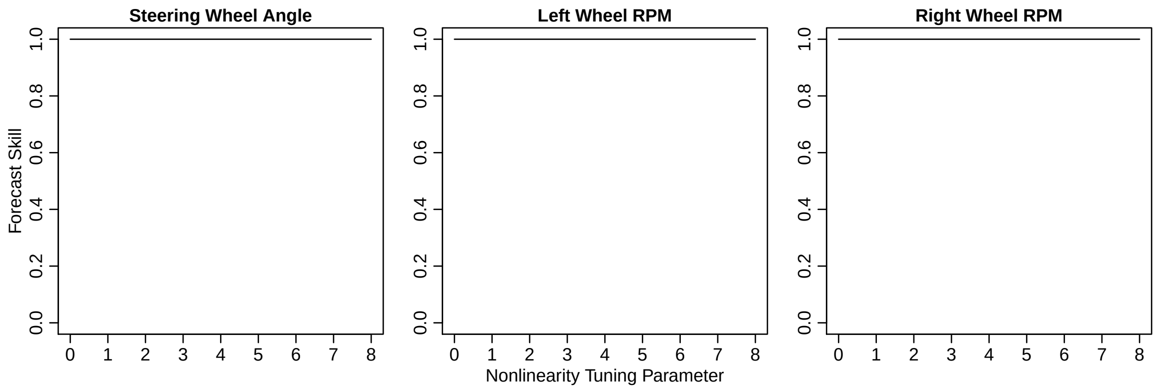


Figure 27. Plots illustrating the nonlinearity of each steering system time series.

EDM also enables next-point predictions. Figure 28 overlays these predictions on each time series. Clearly, these predictions are extremely accurate, which indicates that the three variables do not change significantly from one observation to the next. The plots also show the prediction variance by way of a shaded polygon, but the variance is so low that the polygons are all-but-invisible. Remember that Figure 26 already implied this: for every value of tp, ρ is very high. Figure 29 depicts the prediction errors (residuals); the vast majority of these errors are exceptionally small. Additionally, we devised a naive prediction model. This model simply predicts that the point at time $t + 1$ has the same value as the point at time t. In other words, the simple model predicts *no change* for the next value. Table 16 compares the root-mean-square error (RMSE) between the naive model and EDM. As the table illustrates, EDM noticeably outperforms the baseline predictor for each time series, but one must consider the significant increase in computational complexity for EDM. Finally, these time series are incapable of large, instantaneous changes, so accurately predicting the next point is not very impressive and is not often useful in practical applications. However, these predictions could still assist in IDSs of sufficiently low complexity. Of course, methods other than EDM may also suffice for linear systems.

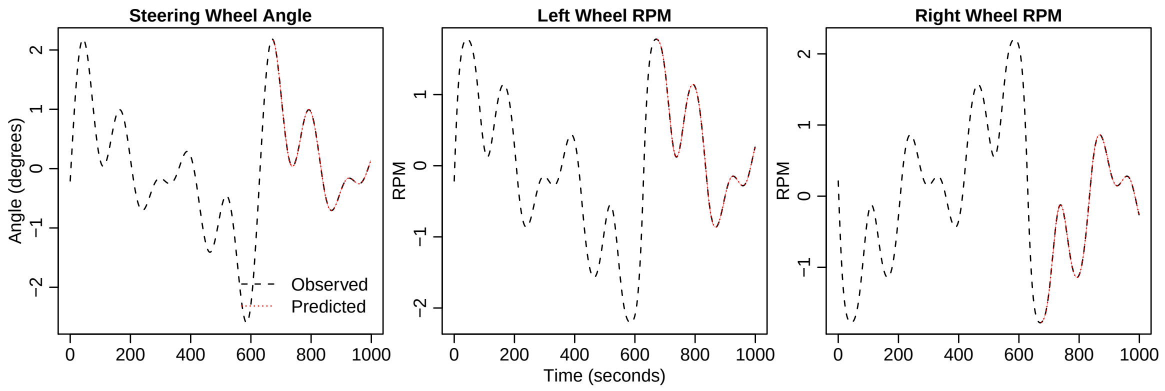


Figure 28. Plots illustrating the predictions for each steering system time series.

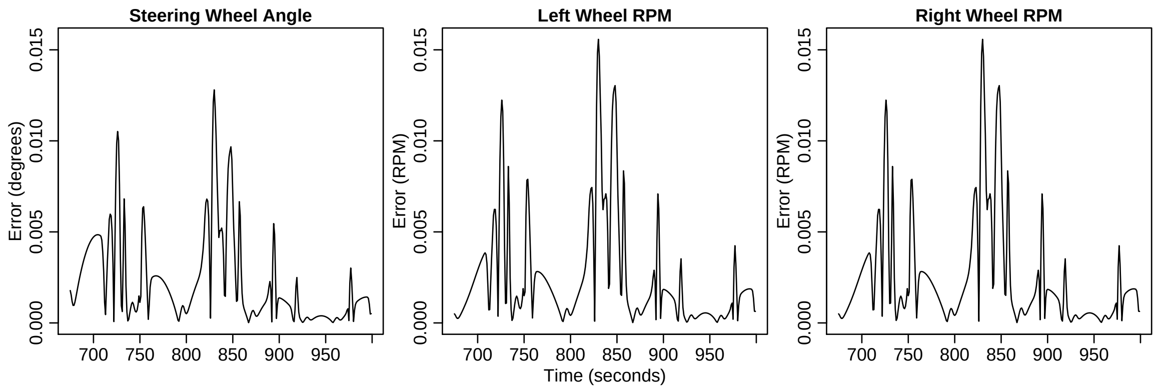


Figure 29. Plots illustrating the prediction error for each steering system time series.

Table 16. RMSE for Each Steering System Time Series

Time Series	Naive Prediction RMSE	EDM Prediction RMSE
Steering wheel angle	0.009424	0.003351
Left wheel RPM	0.005742	0.003893
Right wheel RPM	0.005742	0.003893

Figure 30 depicts inter-variable dynamics within the system. The figure plots cross-map skill ρ[15] against library size—the number of points used to compute ρ—for each pair of variables. Each plot contains two lines, one for X xmap Y and one for Y xmap X. Here, X xmap Y refers to the CCM analysis technique which uses the shadow manifold of X to forecast the shadow manifold of Y. For a given library size, the resulting value for ρ indicates this predictive capability. The three plots show that ρ is equivalent across library sizes and in both directions for every pair of time series. This means that steering information is encoded in the RPM data and that RPM information is similarly encoded in the steering data, which in turn implies an expected causal effect in both directions. Unfortunately, it appears that EDM does not enable insight concerning pairwise causality for this dataset.

Finally, Figure 31 depicts the system's causality through time. The lines again represent the results of using X to forecast Y, but we now plot ρ against tp. As

[15]Cross-map skill quantifies the ability to use one shadow manifold to identify values in another.

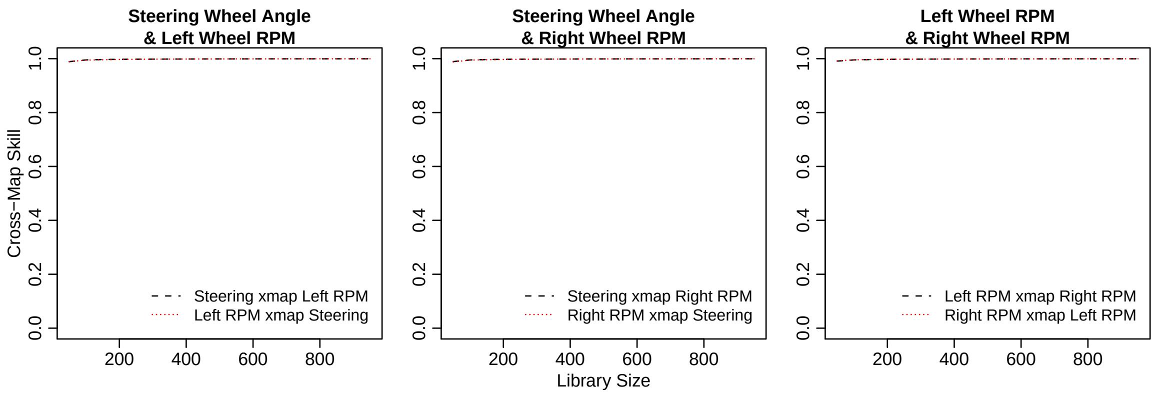


Figure 30. Plots illustrating the causality between each pair of steering system time series.

previous Sugihara Lab researcher Hao Ye writes, "Note here that negative values of tp ... indicate that *past* values of Y are best cross-mapped from the reconstructed state of X. This suggests a dynamical signal that appears first in Y and later in X, and is consistent with Y causing X" [53]. When tp is positive, the opposite holds. For this system, regardless of tp and of the variables in question, $\rho \approx 1$. Thus, according to EDM, each variable has a strong causal effect on every other variable regardless of the time to prediction. This is unlikely, and it supports the claim that EDM does not appear to enable sophisticated analysis of the system's causality.

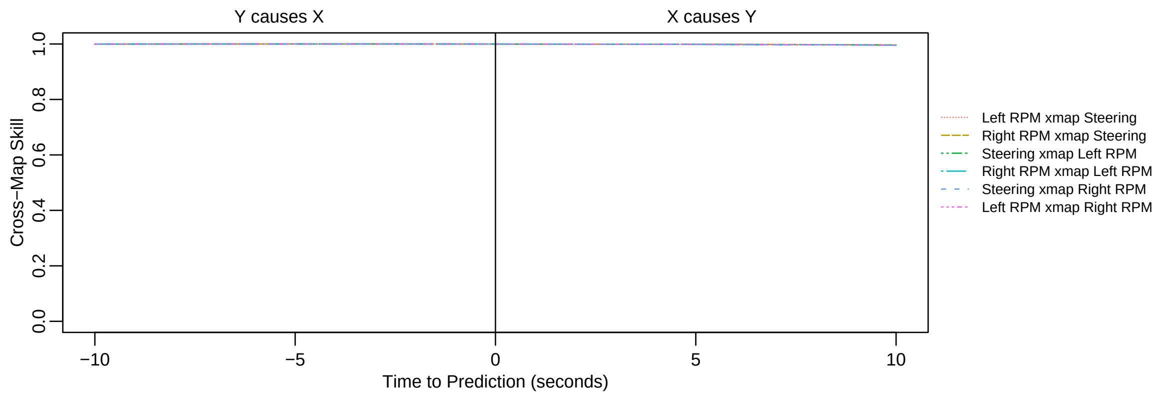


Figure 31. Plots illustrating predictive capability by analyzing the causality between each pair of steering system time series.

4.2.3 Nonlinear Data — Granger Causality

Figure 32 depicts the Granger causality for the selected AVAS time series. Because the system follows nonlinear dynamics, the analysis afforded by the Granger causality functions is inferior to that of Section 4.2.1. Successful application of the technique assumes linear dynamics, but altitude, airspeed, and pitch form a highly nonlinear system. This nonlinearity confuses the causality equations and gives a p-value less than 0.05 for each of the 120 results, which in turn implies that any of the system's variables Granger-causes any other variable for at least 20 seconds. This is not intuitive, and it does not lead to a better understanding of the variable interactions. For example, it is not likely that an airplane's airspeed causally affects its altitude—the two variables can be correlated, of course—but Granger causality asserts that it does. Furthermore, Granger causality assumes *direct* causality and is unable to account for secondary interactions between time series.[16] This maintains Clive Granger's original assumption that the technique does not apply well to nonlinear systems.

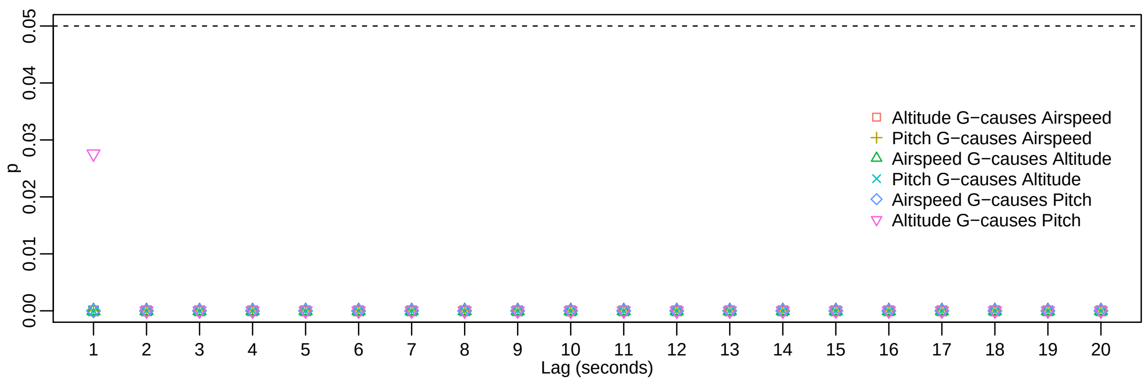


Figure 32. Granger causality between each pair of selected AVAS time series.

[16]If A causes B and B causes C, then A causes C. Granger causality can fail to identify the relationship between A and C. Alternatively, Granger causality may identify causality between X and Y (in either direction) when in fact both are caused by Z.

4.2.4 Nonlinear Data — EDM

For each of the three AVAS time series, Figure 33 presents the forecast skill ρ for various embedding dimensions E. Visually, differences in ρ are minuscule, but the optimal embedding dimension is two for each series. We thus let $E = 2$ for the remainder of the EDM analysis techniques in this section. Additionally, we again let τ equal the time between two observations in a given time series: one second.

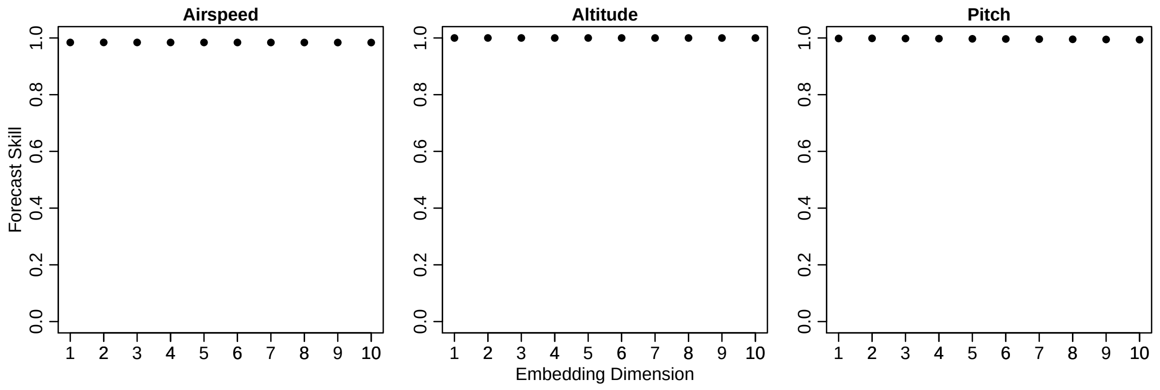


Figure 33. Plots illustrating the optimal embedding dimension for each selected AVAS time series.

Figure 34 plots the forecast skill ρ against the time to prediction tp to illustrate the system's deterministic chaos. For each time series, the figure shows that predictions further in the future are less accurate than earlier predictions. Clearly, this effect is strongest for pitch and weakest for altitude, but this is evidence of chaotic behavior for all three variables.

As before, we plot ρ against θ to characterize each variable's nonlinearity. Figure 35 shows these plots. For all three variables, $\rho \approx 1$ for every tuning parameter; this implies the absence of nonlinear dynamics in the time series. This is not an intuitive result for a system with demonstrated nonlinearity. For this reason, we cannot definitively claim the presence or absence of nonlinear dynamics.

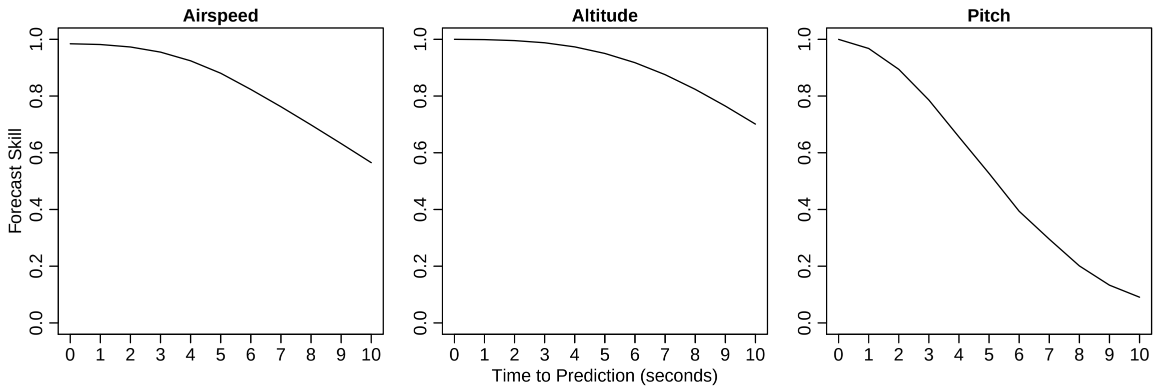


Figure 34. Plots illustrating the deterministic chaos present in each selected AVAS time series.

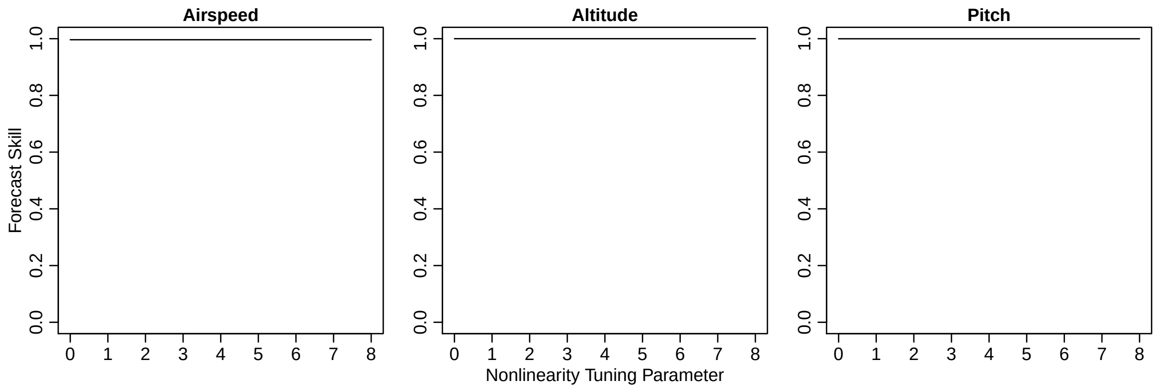


Figure 35. Plots illustrating the nonlinearity of each selected AVAS time series.

Figure 36 presents the next-point predictions given by EDM for each time series. Unsurprisingly, the variance in the predictions—as shown by the nearly imperceptible shaded polygon—is slight. As Figure 34 strongly indicated, none of the variables change significantly between a pair of observations. Figure 37, which depicts the prediction errors, confirms that EDM's predictions are highly accurate. Additionally, Table 17 demonstrates that EDM vastly outperforms the baseline model for two of the three time series; although the simple predictor performs better for pitch, the difference in RMSEs is insignificant. It is once again possible that even these short-term predictions could assist in IDS development. However, it is important to note an obvious limitation of EDM predictions: the technique cannot foresee values not contained in the library. This explains the large outlier predictions.

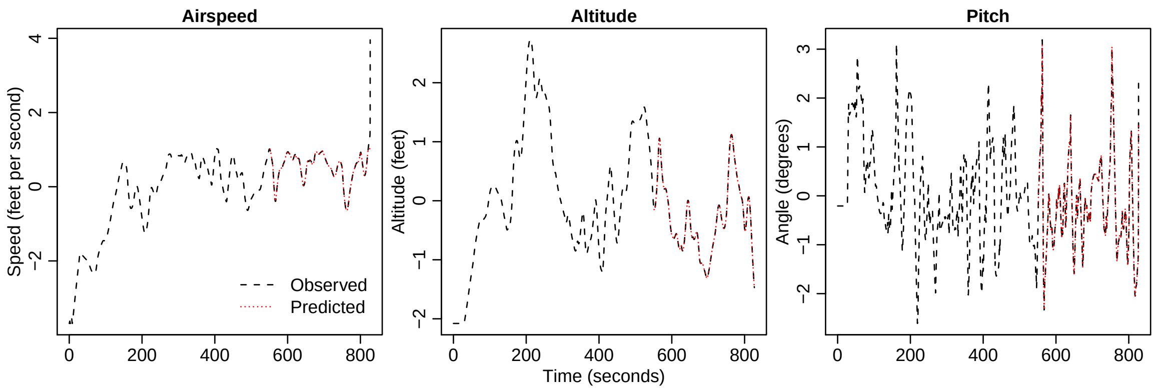


Figure 36. Plots illustrating the predictions for each selected AVAS time series.

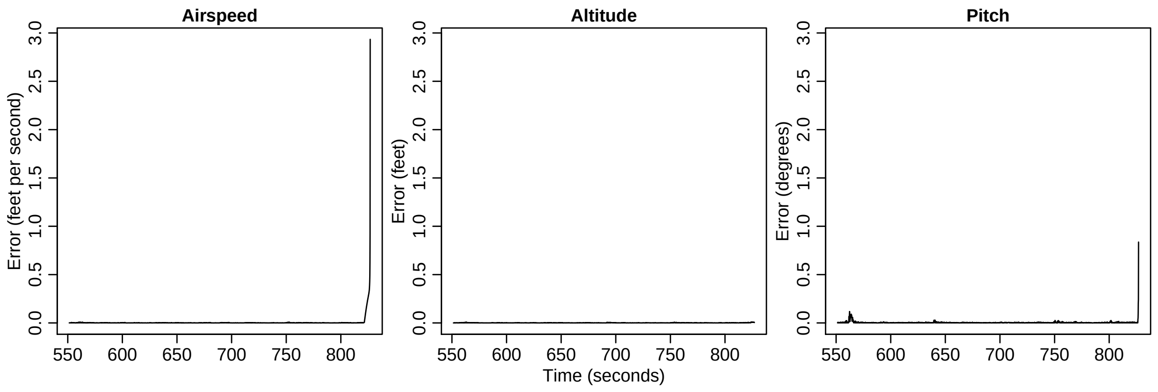


Figure 37. Plots illustrating the prediction error for each selected AVAS time series.

Table 17. RMSE for Each Selected AVAS Time Series

Time Series	Naive Prediction RMSE	EDM Prediction RMSE
Airspeed	3.907005	0.070161
Altitude	18.192201	0.001535
Pitch	0.013749	0.019748

Figure 38 depicts cross-map skill for each pair of time series. The leftmost plot shows that airspeed's manifold can effectively forecast altitude's but that the opposite relationship is noticeably weaker. The middle plot shows that the difference in cross-map skill between airspeed xmap pitch and pitch xmap airspeed decreases as library size increases. The rightmost plot shows a more extreme case of this: above a certain library size, cross-map skill for altitude xmap pitch surpasses cross-map skill for pitch xmap altitude. Furthermore, the pairwise relationships weaken as we examine the plots from left to right. Finally, in all cases, the results indicate diminishing returns in improving ρ by increasing library size, but it is still possible that this analysis enables insight vital to better IDS design.

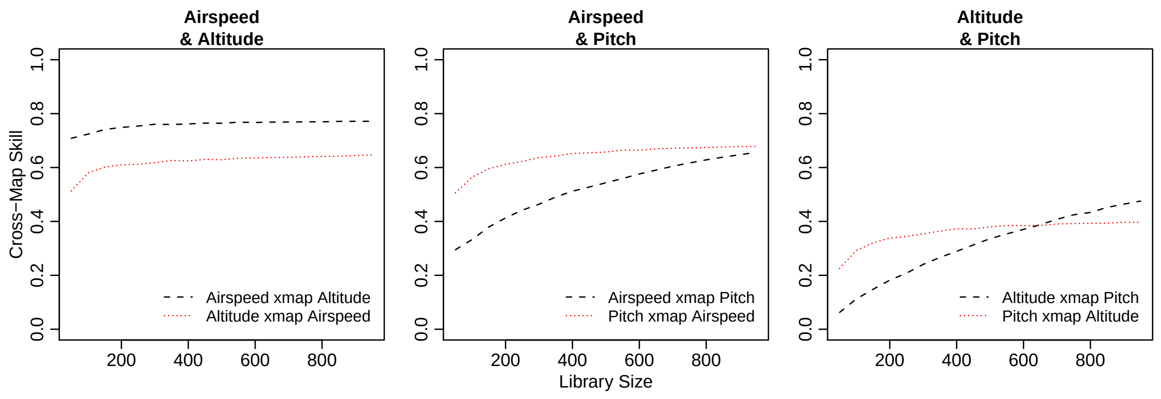


Figure 38. Plots illustrating the causality between each pair of selected AVAS time series.

The last figure, Figure 39, plots cross-map skill against time to prediction. Consider for example airspeed xmap pitch. When tp is slightly less than zero, ρ is maximized; this implies that airspeed best predicts pitch when lagged by about one second. In other words, pitch strongly affects airspeed after one second. This is an expected behavior. When tp is positive, ρ quickly decreases and thus we assert that airspeed does not have a strong causal effect on pitch. This too is consistent with the standard interpretation of an airplane's mechanics. Of course, further analysis of the figure affords other conclusions. Note that the lines X xmap Y and Y xmap X are not necessarily symmetrical because the process of using a shadow manifold M to forecast another shadow manifold $\overline{M}$ is itself an asymmetrical process.

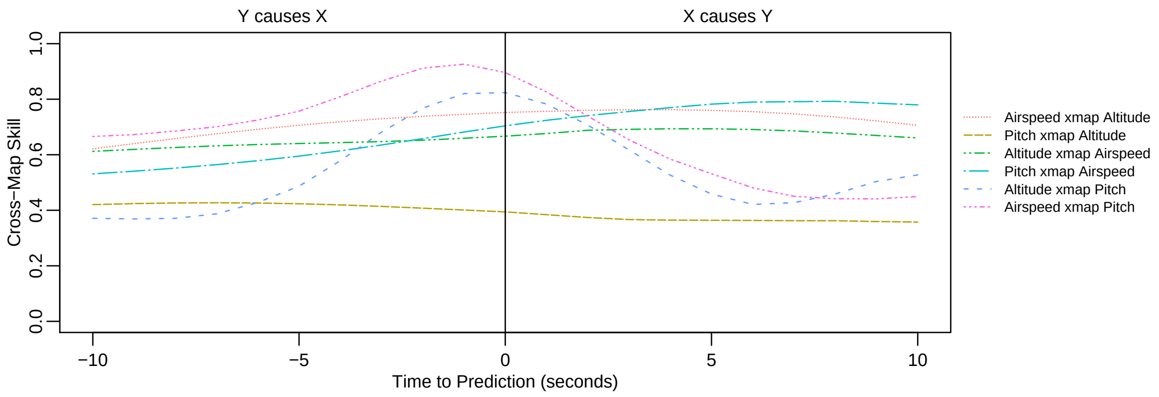


Figure 39. Plots illustrating predictive capability by analyzing the causality between each pair of selected AVAS time series.

4.3 Summary

This chapter discussed two sets of time series, one linear in nature and the other nonlinear, and two different techniques for analyzing the time series. For each set, we first applied Granger causality to identify the causality between each pair of time series. The results indicate some success when applying the technique to the linear system, but they also indicate that Granger causality does not appear to apply to a nonlinear system. We then applied EDM to the two systems to identify nonlinearity and chaos, make next-point predictions, and quantify causality over time. This technique appears to perform better for nonlinear systems, but it still enables some analysis for linear systems. Section 5.2 examines the ramifications of these results.

V. Conclusions

This chapter discusses the real-world implications of the findings presented in Chapters III and IV. Section 5.1 summarizes the controller area network (CAN) research, examines the implied risks to modern automobiles, and presents a potential avenue for future work. Similarly, Section 5.2 describes implications of the findings concerning the time series analysis techniques. Additionally, the section considers limitations of the experiments and details several opportunities for future researchers. Finally, Section 5.3 restates the research contributions and concludes this thesis.

5.1 Fingerprinting Vehicles with CAN Bus Data Samples

The research discussed in Chapter III has two primary conclusions:

1. A vehicle's CAN data uniquely identifies said vehicle.

2. An attacker can use one of the tools presented to improve CAN attacks.

Clearly, one can fingerprint vehicles with CAN data samples. In three different experiments, we evaluated classification performance using CAN samples consisting of ordered payload data, of ordered arbitration IDs, and of unordered payload data. For each experiment, the best classifier consists of a convolutional neural network (CNN) trained on the full set of samples and meets or exceeds a balanced classification accuracy of 98.83%. This performance indicates that deep learning methods can, with high probability, fingerprint a vehicle using only its raw CAN data.

Overall, our findings indicate a significant CAN vulnerability that an attacker familiar with deep learning can easily exploit. Such an attacker can use a well-tuned CNN (or a multilayer perceptron if resources or knowledge are limited) to build a database of known vehicles; the attacker can then reference this database when intruding on a new CAN bus to better structure the desired attack. This risks operator and passenger safety, and it warrants action on the part of automobile manufacturers. It also warrants further research into other deep learning applications on the CAN.

As detailed in Section 3.5, devising a Siamese neural network (SNN) capable of learning the difference between CAN segments from different vehicles is a logical next step for this research. Although our exploratory efforts were unsuccessful, the suggestions provided to future researchers may enable a stronger SNN. This would be very helpful to the malicious intruder: such an attacker would not need to collect CAN segments from every potential vehicle to effectively attack new vehicles.

5.2 EDM as a Basis for an Intrusion Detection System

The Avionics Vulnerability Assessment System, while useful for this research, is limited in scope. Future researchers who wish to affirm the conclusions found here should generate data with an extensively tested simulator, or they should obtain real data from real aircraft. In the same vein, the simulated linear system is elementary and wanting for more real-world characteristics. Regardless, the results presented in Section 4.2 suggest three primary findings:

1. For both linear and nonlinear systems, Granger causality's limitations ensure that intrusion detection system (IDS) architects should search elsewhere for effective analysis techniques.

2. Although empirical dynamic modeling (EDM) can quantify behaviors present in linear systems, the results are often limited and so are not likely to aid in the development of IDSs.

3. For nonlinear systems, EDM is an easy-to-use suite of tools capable of evoking detailed insights that may assist in IDS design.

Although both (1) and (2) imply that the analysis techniques explored here are unsuitable for linear systems, cyber-physical systems (CPSs) are often nonlinear. It is important to note, however, that our limited linear system is likely not fully representative of a real-world linear system. For this reason, future researchers may wish to verify the applicability of the first two conclusions to a robust linear system.

Concerning (3), we assert that the analysis of a nonlinear system afforded by EDM successfully enables the understanding required by the first step towards an IDS. Section 4.2's results demonstrate an ability to effectively quantify a nonlinear system's causality, and this in turn enables better system insight for IDS architects. However, note that, although this research demonstrates the potential of EDM, we cannot realize its true value without larger, more realistic, and more complex datasets and without demonstrated success on a wider range of critical CPSs. Future work should thus address these limitations as well as the remaining steps towards an IDS—namely, obtaining quality data, identifying whether new traffic conforms to the patterns exhibited by the data, and creating a system to notify the administrator when the traffic does not. Future researchers may also wish to conduct a deep exploration of other time series forecasting and analysis techniques, like autoregressive integrated moving average, for IDS design. Still, we believe EDM to be a powerful, emerging tool relevant to critical infrastructure protection, and we strongly suggest further research into its applications to IDSs and beyond.

5.3 Contributions & Conclusion

This work first explored a risk to a specific, common CPS before considering methods to protect any given CPS from intruders. In doing so, our research has made several contributions to CAN security, to IDS development, and to cybersecurity as a whole. In Chapter III, we showed empirically that a vehicle's raw CAN traffic can uniquely identify the vehicle, and we theorized ways in which an attacker could exploit this vulnerability. This is the first known demonstration of this CAN vulnerability and of its associated risks. In Chapter IV, we examined two techniques for time series analysis to determine whether either technique can enable a viable IDS for vulnerable systems like the CAN. The discussion of EDM's applicability to IDS development may serve as the motivation for additional research efforts. The provided library of all available EDM functions could enable other studies concerning nonlinear time series analysis in a wide range of fields. Finally, it is our belief that more research into (1) mitigating the demonstrated CAN vulnerability and (2) further applying EDM to IDS development will prove beneficial to the field of cybersecurity and, specifically, to the protection of CPSs.

Appendix A. EDM Analysis in R

The following R code serves as a demonstration of the rEDM implementation we used for the linear dataset. We stripped all irrelevant details, including plot formatting and exporting, for brevity and clarity. As presented, the functions require an input file with a specific format. Table 18 depicts this format.

Table 18. Input File Format for rEDM Functions

Time	Steering wheel angle	Left wheel RPM	Right wheel RPM
0	0.000000000	0.500000000	0.500000000
1	0.037775020	0.531704461	0.468295539
2	0.075493318	0.562495611	0.437504389
3	0.113098283	0.592274058	0.407725942
...	...	...	...

As mentioned in Section 4.1.3, we use the Sugihara Laboratory's rEDM repository on GitHub for the core EDM functionality. That section describes this functionality. Our code, meanwhile, collates the various rEDM tutorials hosted by the Sugihara Laboratory into a single, easy-to-follow library. We include the following functions:

- `ComputeE`: This function computes the optimal embedding dimension E for some input time series.
- `ComputeTp`: This function computes the optimal time to prediction tp for some input time series to analyze deterministic chaos.
- `PlotNonlinearity`: This function computes the nonlinearity present in some input time series.
- `PlotPredictions`: This function makes and plots next-point predictions for some input time series.
- `PlotCausality`: This function computes pairwise causality against library size for each pair of time series in an input set.

- `PlotCausalityMeans`: This function computes pairwise causality over time for each pair of time series in an input set.

- `RunEdm`: This function applies all other functions to an input set of time series.

Of course, researchers interested in our efforts may wish to modify this code to suit their own purposes. In this case, we recommend consulting the rEDM repository[17] and the Sugihara Laboratory's profile,[18] both on GitHub. The latter also includes repositories for EDM functions in Python and in C++.

[17]https://github.com/SugiharaLab/rEDM
[18]https://github.com/SugiharaLab

```r
# ------------------------------------------------------------
# Initialization Details
# ------------------------------------------------------------

# load required libraries
library(rEDM)
library(plyr)
library(ggplot2)

# load dataset
df <- read.csv("data/steering.csv")

# segment dataset into library/prediction sets
n <- NROW(df)
lib <- c(1, floor(2/3 * n))
pred <- c(floor(2/3 * n) + 1, n)

# segregate and scale each time series
steering <- df[c("Time", "Steering.wheel.angle")]
left <- df[c("Time", "Left.wheel.RPM")]
right <- df[c("Time", "Right.wheel.RPM")]
steering$Steering.wheel.angle <- scale(steering$Steering.wheel.angle)
left$Left.wheel.RPM <- scale(left$Left.wheel.RPM)
right$Right.wheel.RPM <- scale(right$Right.wheel.RPM)

# ------------------------------------------------------------
# Key EDM Functionality
# ------------------------------------------------------------

# identify the optimal embedding dimension (E)
ComputeE <- function(var) {
    out <- simplex(var, lib=lib, pred=pred, E=1:10)
    out[is.na(out)] <- 0
    out$rho[out$rho < 0] <- 0
    E <- which.max(as.numeric(unlist(out[c("rho")])))
    E <- as.numeric(unlist(out[c("E")]))[E]

    plot(out$E, out$rho, type="l", main=colnames(var)[-1], xlab="E", ylab="ρ")
    return(E)
}

# identify the optimal time to prediction (tp)
ComputeTp <- function(var, sequence=0:10) {
    opt_e <- ComputeE(var)
    out <- simplex(var, lib=lib, pred=pred, E=opt_e, tp=sequence)
    tp <- which.max(as.numeric(unlist(out[c("rho")])))
    tp <- as.numeric(unlist(out[c("tp")]))[tp]

    plot(out$tp, out$rho, type="l", main=colnames(var)[-1], xlab="tp", ylab="ρ")
    return(tp)
}

# identify any nonlinearity present in the system
PlotNonlinearity <- function(var) {
    opt_e <- ComputeE(var)
    out <- s_map(var, lib, pred, E=opt_e)
    plot(out$theta, out$rho, type="l", main=colnames(var)[-1], xlab="θ", ylab="ρ")
}
```

```r
# make and plot next-point predictions
PlotPredictions <- function(var, sequence=0:10) {
    opt_e <- ComputeE(var)
    opt_tp <- ComputeTp(var, sequence)
    out <- simplex(var, lib=lib, pred=pred, E=opt_e, tp=sequence, stats_only=FALSE)
    preds <- na.omit(out$model_output[[opt_tp + 1]])

    plot(var, type="l", main=colnames(var)[-1], xlab="Time", ylab="Value")
    lines(preds$time, preds$pred, col="blue", lty=2)
    polygon(c(preds$time, rev(preds$time)), c(preds$pred - sqrt(preds$pred_var),
            rev(preds$pred + sqrt(preds$pred_var))), col=rgb(0,0,1,0.3), border=NA)
}

# conduct and plot pairwise causality analyses over time
PlotCausality <- function(vars=list(steering, left, right), tp=-10:10) {
    # identify the optimal embedding dimension (E) for each column
    i <- 1
    opt_e <- list()
    var_names <- list()

    for (var in vars) {
        var_names[i] <- colnames(var)[-1]
        opt_e[i] <- ComputeE(var)
        i <- i + 1
    }

    opt_e <- unlist(opt_e)
    var_names <- unlist(var_names)

    # get every combination of var1 xmap var2
    # add an (E) column that corresponds to the lib column
    params <- expand.grid(lib_column=var_names, target_column=var_names,
        tp=tp, stringsAsFactors=FALSE)
    params <- params[params$lib_column != params$target_column,]
    rownames(params) <- NULL
    params$E <- as.integer(mapvalues(params$lib_column, var_names,
        opt_e, warn_missing=FALSE))

    # compute causality
    out <- do.call(rbind, lapply(seq_len(NROW(params)), function(i) {
        ccm(df, E=params$E[i], random_libs=FALSE, lib_sizes=n,
            lib_column=params$lib_column[i],
            target_column=params$target_column[i],
            tp=params$tp[i], silent=TRUE)
    }))

    # add a new column to label each pair
    out$direction <- paste(out$lib_column, "xmap", out$target_column)

    # plot the causalities
    labels <- paste(as.character(round(0.1 * tp, 2)), "(s)")
    ggplot(out, aes(tp, rho, colour=direction)) + geom_line() + geom_point() +
        geom_vline(xintercept=0, linetype="dashed") + labs(x="tp", y="ρ") +
        scale_x_discrete(limits=tp, labels=labels)
}
```

```
# conduct and plot pairwise causality analyses
PlotCausalityMeans <- function(var1, var2) {
    # compute causality means
    label1 <- colnames(var1)[-1]
    label2 <- colnames(var2)[-1]
    opt_e <- ComputeE(df[c("Time", label1, label2)])
    var1_xmap_var2_means <- ccm_means(ccm(df, E=opt_e, num_samples=100,
        lib_column=label1, target_column=label2, lib_sizes=seq(50, 950, by=50),
        random_libs=TRUE, replace=TRUE))
    var2_xmap_var1_means <- ccm_means(ccm(df, E=opt_e, num_samples=100,
        lib_column=label2, target_column=label1, lib_sizes=seq(50, 950, by=50),
        random_libs=TRUE, replace=TRUE))

    # compute parallel maxima
    y1 <- pmax(0, var1_xmap_var2_means$rho)
    y2 <- pmax(0, var2_xmap_var1_means$rho)

    # plot the causality means
    title <- paste(label1, "&", label2)
    limits <- c(min(min(y1), min(y2)), max(max(y1, max(y2))))
    plot(var1_xmap_var2_means$lib_size, y1, type="l", ylim=limits,
        main=title, xlab="Library Size", ylab="ρ", col="red",)
    lines(var2_xmap_var1_means$lib_size, y2, col="blue")
    legend(x="topleft", col=c("red", "blue"), lwd=1, bty="n", inset=0.02, cex=0.8,
        legend=c(paste(label1, "xmap", label2), paste(label2, "xmap", label1)))
}

# ----------------------------------------------------------
# Primary Function to Orchestrate Ensemble of EDM Functions
# ----------------------------------------------------------

# calls all other functions
RunEdm <- function(vars=list(steering, left, right)){
    for (var in vars) {
        PlotNonlinearity(var)
        PlotPredictions(var)
    }

    for (var1 in vars) {
        for (var2 in vars) {
            if (colnames(var1)[-1] != colnames(var2)[-1]) {
                PlotCausalityMeans(var1, var2)
            }
        }
    }

    PlotCausality(vars)
}
```

Bibliography

1. C. W. J. Granger, "Investigating Causal Relations By Econometric Models," *Econometrica*, vol. 37, no. 3, pp. 424–438, 1969.

2. ACM Transactions on Cyber-Physical Systems, "Cyber-Physical Systems (TCPS): About," 2018. [Online]. Available: https://tcps.acm.org/about.cfm [Accessed: 2019-05-31]

3. National Institute of Standards and Technology, "Introduction to Time Series Analysis," in *NIST/SEMATECH e-Handbook of Statistical Methods*, 1st ed. NIST, 2012, ch. 6.

4. V. Kotu and B. Deshpande, "Time Series Forecasting," in *Data Science: Concepts and Practice*. Cambridge, Massachusetts, USA: Morgan Kaufman Publishers, 2019, ch. 12, pp. 305–327.

5. R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Melbourne, Australia: OTexts, 2018.

6. Robert Bosch GmbH, "CAN Specification Version 2.0," 1991. [Online]. Available: http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf [Accessed: 2019-07-08]

7. S. Corrigan, "Introduction to the Controller Area Network (CAN)," 2016. [Online]. Available: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf [Accessed: 2019-07-08]

8. F. Hartwich and Robert Bosch GmbH, "CAN with flexible data-rate," *CAN Newsletter*, pp. 10–19, feb 2012.

9. National Instruments, "Understanding CAN with Flexible Data-Rate (CAN FD)," 2019.

10. ——, "Controller Area Network (CAN) Overview," 2019. [Online]. Available: https://www.ni.com/en-us/innovations/white-papers/06/controller-area-network--can--overview.html [Accessed: 2019-07-08]

11. International Organization for Standardization, "Road vehicles – Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements – Part 1: General information and use case definition," 2018. [Online]. Available: https://www.iso.org/standard/46273.html [Accessed: 2019-05-31]

12. B. Stone, S. Graham, B. Mullins, and C. Schubert Kabban, “Enabling Auditing and Intrusion Detection for Proprietary Controller Area Networks,” Dissertation, Air Force Institute of Technology, 2018.

13. R. Buttigieg, M. Farrugia, and C. Meli, “Security Issues in Controller Area Networks in Automobiles,” in *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2018, pp. 93–98.

14. D. E. Denning, “An Intrusion-Detection Model,” *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, 1987.

15. P. Shirani, M. A. Azgomi, and S. Alrabaee, “A Method for Intrusion Detection in Web Services Based on Time Series,” in *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2015, pp. 836–841.

16. A. Greenberg, “Hackers Remotely Kill a Jeep on the Highway—With Me in It,” 2015. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/ [Accessed: 2019-09-30]

17. C. Miller and C. Valasek, “Remote Exploitation of an Unaltered Passenger Vehicle,” *Black Hat USA*, vol. 2015, pp. 1–91, 2015.

18. S. Woo, H. J. Jo, and D. H. Lee, “A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 993–1006, 2015.

19. Z. Tyree, R. A. Bridges, F. L. Combs, and M. R. Moore, “Exploiting the Shape of CAN Data for In-Vehicle Intrusion Detection,” in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, 2019, pp. 1–5.

20. M. J. Kang and J. W. Kang, “Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security,” *PLoS ONE*, vol. 11, no. 6, pp. 1–17, 2016.

21. H. J. Liao, C. H. Richard Lin, Y. C. Lin, and K. Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

22. D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, “A survey of deep learning-based network anomaly detection,” *Cluster Computing*, pp. 1–13, 2017.

23. J. Pearl, *Causality: Models, Reasoning, and Inference*, 2nd ed. Cambridge, Massachusetts, USA: Cambridge University Press, 2009.

24. M. Eichler, "Causal Inference in Time Series Analysis," in *Causality: Statistical Perspectives and Applications.* John Wiley & Sons, Ltd, 2012, ch. 22, pp. 327–354.

25. BiObserver, "Visualization of Granger causality," 2014. [Online]. Available: https://commons.wikimedia.org/wiki/File:GrangerCausalityIllustration.svg [Accessed: 2019-09-27]

26. F. Takens, "Detecting Strange Attractors in Turbulence," in *Dynamical Systems and Turbulence, Warwick 1980.* Berlin, Germany: Springer Berlin Heidelberg, 1981, pp. 366–381.

27. G. Boeing, "Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction," *Systems*, vol. 4, no. 4, p. 37, 2016.

28. Sugihara Lab: Quantitative Ecology and Data-Driven Theory, "Empirical Dynamic Modeling," 2019. [Online]. Available: http://deepecoweb.ucsd.edu/nonlinear-dynamics-research/edm/ [Accessed: 2019-05-31]

29. E. Lorenz, "Deterministic Nonperiodic Flow," *Journal of the Atmospheric Sciences*, vol. 20, pp. 130–141, 1963.

30. H. Ye, A. Clarke, E. Deyle, and G. Sugihara, "rEDM: An R package for Empirical Dynamic Modeling and Convergent Cross Mapping," pp. 1–19, 2019. [Online]. Available: https://cran.r-project.org/web/packages/rEDM/vignettes/rEDM.html [Accessed: 2019-12-12]

31. G. Sugihara, R. May, H. Ye, C.-h. Hsieh, E. Deyle, M. Fogarty, and S. Munch, "Detecting Causality in Complex Ecosystems," *Science*, vol. 338, no. October, pp. 496–500, 2012.

32. G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control.* Upper Saddle River, New Jersey, USA: Prentice-Hall, Inc., 1994.

33. A. H. Yaacob, I. K. Tan, S. F. Chien, and H. K. Tan, "ARIMA based network anomaly detection," *2nd International Conference on Communication Software and Networks, ICCSN 2010*, no. 1, pp. 205–209, 2010.

34. T. Sejnowski J., *The Deep Learning Revolution.* Cambridge, Massachusetts, USA: The MI, 2018.

35. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* Cambridge, Massachusetts, USA: The MIT Press, 2016.

36. C. Nicholson, “A Beginner’s Guide to Neural Networks and Deep Learning,” 2019. [Online]. Available: https://skymind.ai/wiki/neural-network [Accessed: 2019-09-26]

37. A. Burkov, *The Hundred-Page Machine Learning Book.* Andriy Burkov, 2019.

38. J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, “Signature Verification using a ”Siamese” Time Delay Neural Network,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 4, pp. 669–688, 1993.

39. M. Marchetti and D. Stabili, “Anomaly detection of CAN bus messages through analysis of ID sequences,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1577–1583.

40. M. Enev, A. Takakuwa, K. Koscher, and T. Kohno, “Automobile Driver Fingerprinting,” in *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 1, 2015, pp. 34–50.

41. X. Qin and W. Lee, “Statistical Causality Analysis of Infosec Alert Data,” in *International Workshop on Recent Advances in Intrusion Detection.* Springer Berlin Heidelberg, 2003, pp. 73–93.

42. J. B. Cabrera, L. Lewis, X. Qin, W. Lee, R. K. Prasanth, B. Ravichandran, and R. K. Mehra, “Proactive detection of distributed denial of service attacks using MIB traffic variables-a feasibility study,” *2001 7th IEEE/IFIP International Symposium on Integrated Network Management Proceedings: Integrated Management Strategies for the New Millennium*, vol. 00, no. c, pp. 609–622, 2001.

43. J. B. Cabrera, L. Lewis, X. Qin, W. Lee, and R. K. Mehra, “Proactive Intrusion Detection and Distributed Denial of Service Attacks - A Case Study in Security Management,” *Journal of Network and Systems Management*, vol. 10, no. 2, pp. 225–254, 2002.

44. F. Pedregosa, G. Varaquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

45. F. Chollet and others, “Keras,” 2015. [Online]. Available: https://keras.io [Accessed: 2019-06-24]

46. J. Perktold, S. Seabold, and J. Taylor, “StatsModels: Statistics in Python,” 2009.

47. G. Sugihara and R. May, “Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series,” *Nature*, vol. 344, pp. 734–741, 1990.

48. G. Sugihara, “Nonlinear forecasting for the classification of natural time series,” *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences*, vol. 348, no. 1688, 1994.

49. J. M. Lee, *Introduction to Topological Manifolds*, 2nd ed. New York, New York, USA: Springer US, 2011.

50. C. W. Chang, M. Ushio, and C. hao Hsieh, “Empirical dynamic modeling for beginners,” *Ecological Research*, vol. 32, no. 6, pp. 785–796, 2017.

51. H. Whitney, “Differentiable Manifolds in Euclidean Space,” *Proceedings of the National Academy of Sciences*, vol. 21, no. 7, pp. 462–464, 1935.

52. N. Rennie, “Empirical Dynamic Models: A Method for Detecting Causality in Complex Deterministic Systems,” pp. 1–20, 2018.

53. H. Ye, “Using rEDM to quantify time delays in causation,” 2019. [Online]. Available: https://cran.r-project.org/web/packages/rEDM/vignettes/rEDM-time-delay-ccm.html [Accessed: 2019-12-03]

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* xx–xx–2020	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sep 2018 — Mar 2020

4. TITLE AND SUBTITLE Cyber-Physical System Intrusion: A Case Study of Automobile Identification Vulnerabilities and Automated Approaches for Intrusion Detection	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Crow, David R., 2d Lt, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering an Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT-ENG-MS-20-M-012
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Intentionally Left Blank	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Today's vehicle manufacturers do not tend to publish proprietary packet formats for the CAN. This is a form of security through obscurity, but obfuscating the network in this way does not adequately hide the vehicle's unique signature. To prove this, we train two distinct deep learning models on data from 11 different vehicles. Our results indicate that one can determine which vehicle generated a given sample of CAN data. A sophisticated attacker who establishes a presence on an unknown vehicle can use similar techniques to identify the vehicle and better format attacks. To protect critical CPSs against attacks like those enabled by this vulnerability, system administrators often employ IDSs. One requires an understanding of the behavior and causality of the CPS to develop an IDS. This research explores two different time series analysis techniques, Granger causality and EDM, which may contribute to this understanding. Our findings indicate that Granger causality is not a suitable approach to IDS development but that EDM might be. We thus encourage further research into EDM applications to IDSs.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Scott Graham, AFIT/ENG
U	U	U	UU	96	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255-6565 x4581; scott.graham@afit.edu