Industrial Control System Anomaly Detection Using Convolutional Neural Network Consensus

Aviraj Sinha[§], Michael Taylor[§], Nathan Srirama, Theodore Manikas, Eric C. Larson, and Mitchell A. Thornton
Darwin Deason Institute for Cyber Security
Southern Methodist University, Dallas, Texas, USA
{avirajs,taylorma,nsrirama,manikas,eclarson,mitch}@smu.edu

***Abstract*—Industrial control systems provide transportation, essential utilities, and the manufacturing of goods to the masses. It is critical that controlled processes are executed correctly and according to schedule. Monitoring the system's performance during its operation is an important approach for maintaining high levels of reliability and availability. We present a system monitoring capability that implements parallel multi-view neural networks to detect anomalous behavior in an industrial control system by predicting operational states. By deploying the prediction capability within the system, system operation can be monitored in a semi-supervised manner to ensure the actual system state lies within an appropriate region of the state space that was previously predicted by the neural networks. Furthermore, if the two predictive models diverge in their classification of state (breaking consensus), it is likely that system operation has been compromised due to faulty equipment, communication errors, or some other source of malfunction. To achieve different "views" of the system, one predictive model is trained to analyze the data flow of system control packets and the other model is trained to analyze gyrometric signals obtained from physical sensors in the control system. We demonstrate that this methodology can detect anomalous behavior of an example industrial control system by emulating its operation in the presence of injected anomalies. Results indicate highly accurate anomaly detection during system operation.**

***Index Terms*—Industrial systems, anomaly detection, machine learning, multi-view classification**

I. Introduction

Industrial and manufacturing facilities rely on computer controlled electro-mechanical frameworks, referred to as Industrial Control Systems (ICS), to efficiently support production and processing objectives. Key attributes of an ICS include safety, reliability, and in more recent years, resilience to potential cyberattacks that can disrupt functionality. In extreme cases, cyberattacks can cause damage or harm to personnel supporting the facility [1]–[4]. Detecting anomalies in an ICS can increase resilience to cyberattacks or other errant behavior. For these reasons, we are motivated to devise methods for automated anomaly detection.

The connections and devices that enable the communication between the components in an ICS installation are supplied by various vendors and are generally interoperable due to the use of standardized computer interfaces and networking protocols that support modern ICS implementations. It is typical that an ICS will demonstrate state-like behavior that characterizes its overall functionality. That is, the ICS cycles through various operating points that can be classified as a particular state. However, in many applications the state space is very large thus making it impractical to capture the complete behavior with traditional methods.

In this work, an ICS network is augmented to include by convolutional neural networks (CNNs) in order to determine the overall health of the devices and components in an industrial environment. Machine learning techniques have been applied to industrial reliability analysis in previous approaches [5]–[11]. Our technique differs in that it relies upon multiple data sources as shown in Figure 1. Since our classification uses multiple data sources with unsupervised error thresholds, the classification is robust to random deviations in sensor and traffic patterns. The neural networks separately analyze the state of the system using two different input streams: (1) the packet data sent along the network and, (2) time series signals from a gyroscope. Each input corresponds to a different "view" of the system state. When the system is functioning properly, the state classified by each model should match or be reasonably similar allowing consensus from the two models to be achieved [12]. However, when faulty equipment or other errors cause unexpected behavior in the system, the classification will diverge, causing loss of consensus. Because the system diverges from normal behavior, this classification can also be described as anomaly detection.

We outline our contributions as:

1) We propose the use of dual CNN state prediction models for detecting anomalies in the context of industrial control systems via consensus. Each CNN uses a different set of features for prediction: (a) analyzing patterns in packet data and (b) analyzing sensor data from co-located sensing points placed in the ICS.
2) We designed and collected data from a test bed of ICS operations (normal and anomalous) using commercial ICS hardware components. This test bed is used to evaluate and characterize the performance of the proposed method.

[§]These two authors contribute equally to the work

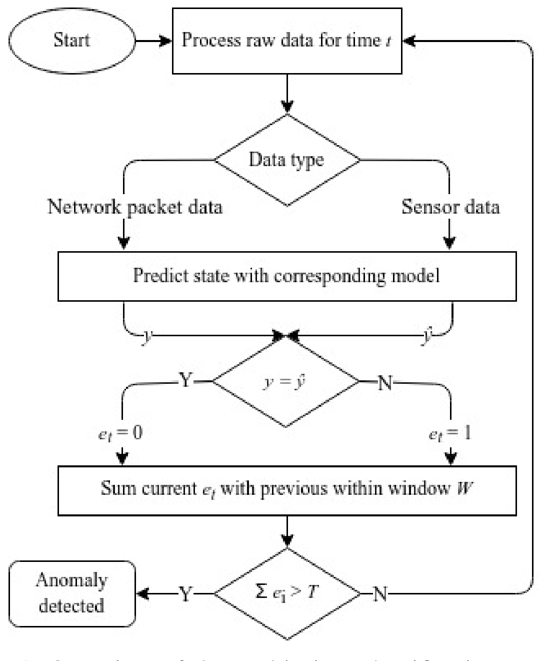


Fig. 1: Overview of the multi-view classification system.

TABLE I: List of symbols used in this paper

Symbol	Description
y	Packet State Prediction (expected by network)
$\hat{y}$	Gyrometer State Prediction (observed by sensor)
s	Data input into model over a time period t
e	Prediction error $e_t = \begin{cases} 0 & y = \hat{y} \\ 1 & y \neq \hat{y} \end{cases}$ (1) for data s
W	Window size for error rate of multiple predictions
T	Threshold for anomalous error rate
A_i	Anomaly for ith window when $\left(\sum_{t=i-W}^{i} e_t\right) > T$ (2)

II. Related Work

Light weight machine learning techniques such as support vector machines (SVMs) and clustering have been employed for detecting anomalies in an ICS [5]. These approaches, however, require careful feature engineering such as characterizing the mean packets bytes and interarrival times [8], [9]. This characterization can cause significant impact in the deployment of an ICS anomaly detector because the feature engineering process must be calibrated to each new ICS. Therefore, many researchers are motivated to use methods from the deep learning community, whereby the model can learn system variable baselines without extensive information gathering.

Deep learning has been steadily gaining popularity for ICS anomaly detection due to its ability to classify both multi- and single-stage attacks. In particular, intrusion detection has been widely tested on a popular dataset, the secure water (SWaT) dataset, which consists of data collected from a scaled-down water treatment plant ICS [1]. The data available for this ICS consists of 26 time series streams (from sensors and actuators) and packet information from the data bus (which uses the Modbus protocol). For the most part, related work using this data has focused either on intrusion detection using packets from the Modbus protocol [11] or the usage of the time series streams to detect anomalies as they occur [7], [10]. Because Modbus is typically unencrypted and therefore more open to attack, many methods look at only packet data.

[1]https://itrust.sutd.edu.sg/itrust-labs-datasets/

Research on ICS sensor data (also from SWaT) has also been performed with long short term memory (LSTM) recurrent networks [10]. Recently, 1D convolutional models for ICS sensor anomalies have had comparable results to LSTMs, but with greater efficiency [7]. 1D convolutions are well suited for time series sensor data due to its design to classify time series through parallel convolutions.

III. Background

A. Deep Learning Concepts

In this section, we describe a popular method for sequential data classification, the LSTM, as well its limitations in this context. We then work to overcome those limitations through the usage of CNN (convolutional neural networks). The main idea is to use CNNs for prediction of not only sensor data, but also raw network bit patterns.

The LSTM cell is a type of layer, or internal data structure, in a recurrent neural network. LSTM cells are useful in machine learning tasks that use time series data, and they can be implemented in classification and regression decision-making tasks. What makes the LSTM layer powerful is that it is able to “remember” important information and “selectively forget” what is not essential to the created decision-making model. Because data ordering is respected, future predictions are made according to a sequence of past events [13].

Many deep learning algorithms can analyze either ICS payload data or sensor data individually such as LSTMs, since they are efficient for simple time series tasks [14]. We focus on convolutional neural networks because they have recently been shown to rival LSTMs performance in the context of an ICS, but with typically more efficiency due to optimized GPU computations [6]. CNNs are feed forward neural networks that were originally proposed for image processing [15], [16]. Having multiple convolutional layers allowed complex spatial patterns to be captured.

The convolutional neural network’s goal is to maximize an output activation of a convolutional classification layer; this is done by training the filter weights to minimize a loss function using gradient descent with gradients computed from backpropagation [17]. These convolutional steps are often denoted as the feature extraction phase. After convolution, it is common practice to down sample the number of outputs by pooling over a space of outputs. One commonly used down sampling technique between convolutional layers is the max pooling operation; max pooling operations distill data to lower dimensions by taking only the maximums values over a segment, which helps capture the significant feature values as well as speed up performance. After the feature extraction and pooling a dense (or fully connected) layer uses the resulting output activations to make a decision. For additional details on CNNs please refer to [18].

One unconventional approach used in our experiment is using CNNs for packet payload classification. That is, using a CNN to analyze raw data in the payload of a network packet. Convolutions are typically employed for analyzing time series sensor data, such as speech, motion data, etc. We find that CNNs are efficient for detecting properties in an ICS payload bit patterns since the ICS command payload tends to follow a recognizable pattern.

B. ICS Network Protocol

There are a variety of communication protocols which can be utilized by Industrial Control Systems, including Modbus, HART, Profibus, Profinet, BACnet, and others. Each was developed with slightly different use cases in mind, but for the most part they are interchangeable within an ICS. A defining feature of these protocols is that they are designed to adhere to strict timing constraints. This is based on a desire for speed and reliability within these industrial systems. The research presented deals with a simulated ICS network which implements the Modbus protocol. The focus for this paper will therefore be on the Modbus protocol, and any reference to ICS protocols, communication systems, or packets will be stated with Modbus in mind.

The Modbus protocol is one of the most commonly used communication protocols within industrial control environments because it is designed specifically with industrial applications in mind. It was introduced in 1979 as a technique to establish controller-worker relationships between programmable logic controllers, remote terminal units, and other systems that were being introduced into industrial systems at the time[2]. An important feature to note about the Modbus protocol is that it was not designed with innate security. There is an implicit trust between all devices on a Modbus network, and packets are not encrypted at any point during communication.

IV. ICS Testbed Description

In order to collect data for the purposes of this research, a miniature ICS network was created. The network includes a Tolomatic industrial motor connected to a human-machine interface (HMI) controller, an accelerometer for acceleration and gyro-metric data, and an inline Raspberry Pi which handles packet collection and simulates network anomalies, as shown in Figure 2. The motor is bolted to a solid foundation and placed under various load configurations in order to better simulate a real industrial environment. The accelerometers and gyrometers were also bolted to the foundation to collect vibrations emanating from the motor.

Dataflow within the created network travels as follows: The HMI sends commands to the motor, which can turn it on or off, change its speed, etc. The HMI will send a continual stream of commands, even if there are no desired changes for the motor. Commands are sent through the inline Raspberry Pi for logging purposes, and then forwarded messages to the motor. The motor in turn responds with a continual stream of gyrometer data, which is also routed through the inline

[2]https://modbus.org

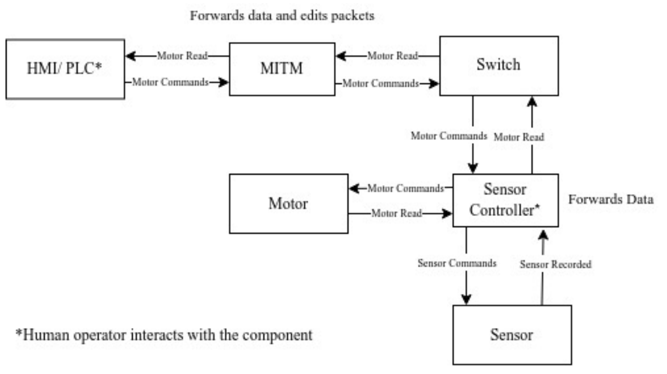


Fig. 2: ICS Testbed Overview Diagram

Raspberry Pi. This data is stored as Comma Separated Values (CSVs) of the following categories: gyroscope (X, Y, and Z rotation axes). These axes represent the orientation of the attached sensor in the three dimensional space.

The created ICS system communicates using Modbus packets between the HMI and the industrial motor. Communication is structured such that all messages sent from the HMI to the motor will result in a response message sent back to the HMI. Modbus packets within the system therefore represent Read/Write commands for the motor, and motor data sent back to the HMI.

A. Anomaly Generation

Our experiments involve creating anomalies through a Man-in-the-Middle (MITM). MITM attacks are carried out by a third party that has secretly taken control of the communication channel between two or more endpoints in the network. Such changes would allow for the attacker to gain complete control over an asset in the network by dropping or modifying all communication from a controller [19]. Additionally, the attacker can spoof all communication in a way that makes the asset appear to be operating as expected. The repercussions of such an attack could range anywhere from a small amount of downtime to catastrophic damage to equipment, infrastructure, and even personnel.

For our experimental setup, the MITM device was set up in a way which placed all incoming packets on the two network connections into a kernel queue. The device also continuously ran a process which pulled the packets from the kernel queue for analysis and potential modification based on a command line prompt made available to a user. The program would first determine which network asset was sending the message and which network asset was receiving the message. Afterwards, the program would check if the devices were part of any attack strings entered by the user. If the devices were not part of any attack strings or there were no attack strings provided by the user the packet would be forwarded. However, if the devices did match with an attack string the program would make the requisite modifications to the packet data before forwarding. Attacks included stopping the motor, changing the speed of the motor, and changing the direction of the motor. The user also had the option to spoof the returned data from the motor to the HMI so that it appeared to be operating as expected. This was achieved by maintaining a record of the actual speed and direction data from the most recent communication from the HMI.

V. Experimental Setup

A. Method Description

As mentioned in related works, most ICS anomaly methods depend on detecting trace changes in either packet or sensor data. Packet detection methods rely more on changes caused by command packet manipulation, whereas for sensor anomaly detection focus on detecting abnormal behavior in the physical system. Network based methods were more popular because they could find anomalies before any damage was done; however, now network packets can be easily spoofed by MITMs. Though, looking at both of these data types separately would be helpful for identifying anomalies, using them together can take advantage of the relationship between the network data and the sensor data — the fact that command payloads will result in specific patterns of sensor behavior. In the case of an ICS, the network data conveys the expected ICS state y while the gyrometer sensor data presents the actual observed ICS state $\hat{y}$. The benefit of this relationship is that a MITM (man in the middle attacks) will be required to spoof both command payloads and sensor information at the same time in order for $y = \hat{y}$. Though this is a possible weakness of this approach, spoofing both network and sensor data would require very in-depth knowledge of the ICS system at all times.

For our experiment, two CNNs are used to classify y and $\hat{y}$ as one of six ICS states. When $y = \hat{y}$ an error is marked as 1 for that sample. Over time random errors can accumulate; however, over specific windows of time, W, such as an attack, the percentage of errors per prediction should surpass a threshold, T, and be labeled an anomaly as defined in equation 2. The anomalous state is a label that was not available during training. As a result of using labeled data and unlabeled data for classification, our model falls under the category of semi-supervised learning.

This technique contrasts with previous methods that would accumulate errors as standard deviations from the mean rather than model prediction errors. As a consequence, previous methods assume a normal distribution of differences for both sensor and payload data so that thresholds could be established for detecting deviations—which is not always a reasonable assumption, especially as windows of error accumulation become shorter and the central limit theorem has fewer guarantees. We note that a number of more complicated detection algorithms are possible that discern not only the number of errors, but also how far apart each prediction might be. We find that empirically this formulation, while simple, works well and requires the least amount of fine tuning—only T and W, are required for calibration.

B. Machine Learning Implementation

Our model architecture is shown in Table II. The input to the model is either raw time series sensor data or bit streams from the payloads in packets over time. Proper training, validation, and testing splits are performed at the ratio of 70:20:10 to ensure the model can accurately detect ICS states from payloads and sensors. Data preprocessing, training results, and details on how models are used together are mentioned in the next subsections. To create the model, we use the Keras package for design and training [20]. The remainder of discussion in this subsection assumes some familiarity with this software package—details are provided for reproducibility. The model uses a combination of convolutional, max pooling, and fully connected (dense) neural network layers. All activation functions are ReLU except for the final softmax activation for classification. The loss function employed for training is cross entropy across the six possible ICS states. An adaptive momentum (ADAM) optimizer is employed with a learning rate of 1e-5 and is used to iteratively update the weights. The model is trained for 100 epochs and dropout is used to help prevent over fitting.

TABLE II: Deep Convolutional State Classifiers

Layer	Input Payload	Input Gyrometer
Input	424x1 (Average of 100)	100x3
Conv 1D (ReLU)	424x32	100x32
Max Pooling 1D	212x32	50x32
Conv 1D (ReLU)	212x32	50x64
Max Pooling 1D	106x32	25x64
Flatten	3392	1600
Dropout	3392	1600
FC/Linear	64	64
FC/Linear(Softmax)	6	6

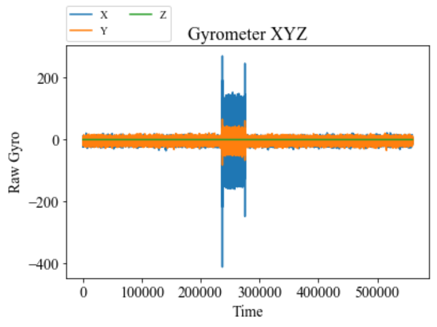
C. Experimental Design

1) Dataset Description:

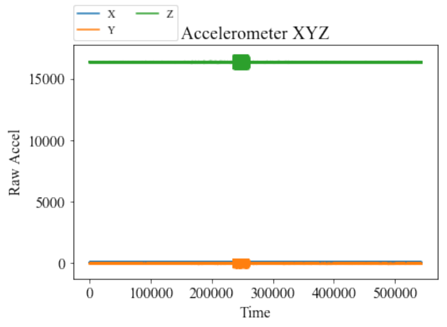
a) ICS State Labels: The dataset includes packet payload and corresponding sensor data. For our classification task, we focus on classifying packets/sensors data as one of the ICS states: 'FAST', 'HALT', 'MEDIUM', 'OFF', 'REVERSE', 'SLOW'.

b) Gyrometer: The sensors include gyrometer data for all X, Y, and Z directions; this data is collected as a constant stream as the sensor controller continuously logs data from the motor at a fixed sample rate of 10 thousand samples per second. The gyrometer data is logged as floating point values represent angular velocity as degrees per second. From the diagram in Figure 3, we can see the change in our raw data types during for a single MITM attack around sample 25000. For our gyrometer sensors, we can visually observe that there is a state change from the random short burst of speed and forces. From this example of an ICS state change, we know that a state change can be quick to create a noticeable anomaly; this means our model should take enough samples to detect the state change but not take enough that the attack be reduced by the surrounding noise of the baseline data.

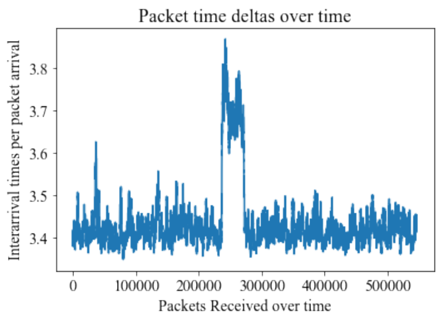
c) Modbus payload: The second data type includes the Modbus payload. Rather than being a constant stream of data input each payload arrives at different times from the



(a)



(b)



(c)

Fig. 3: Visualizations of sensor streams and payload data in the dataset

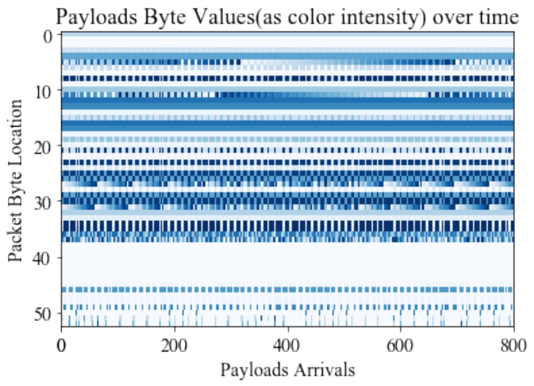


Fig. 4: The color intensity represents byte values 0-255

controller. Since, the network data was preprocessed from PCAP files the payload data was converted from its original byte format to binary. In each individual data payload there were 53 bytes, each byte was between 0 to 255. For payload preprocessing for our model, the 53 bytes were converted to binary for machine learning input changing the input width from 53 bytes to 424 bits. In Figure 4 payload data is represented in its byte format where each of the 53 bytes are vertically stacked and pixel color intensity represents the 0-255 value for that byte. That is, the color changes in each row represent the change in value of the field in the packet over time. This coloring allows us to visualize that some byte locations in our packet have static, cyclic, or random values over time. In contrast to the sensor data, we cannot immediately observe the ICS state change from this visualization, we hypothesize that the machine learning convolutional model will be able to classify states of different time windows.

d) Unused Data: Another data type we collected but did not use in our model are interarrival times, also known as packet time deltas or in statistical terms, the first difference. Each packet contains a timestamp for which it arrives through the controller to the motor. By taking the differences between two adjacent timestamps we can get the amount of time between each packet arrival from the controller to the motor actuator. From our experiments, as shown in our raw dataset diagram Figure 3c, we have observed that interarrival times (time between packet arrivals) increase a certain constant amount during an ICS state change before returning to the baseline interarrival time; as a result, interarrival times may have potential for detecting anomalous state changes. However, due to the lack of experimental classification results and distinguishing interarrival times characteristics between states, the ICS states themselves cannot be classified. Overall, there has not been enough experimentation with other models to determine if interarrival times can help classify ICS states, and since our semi-supervised method heavily relies on comparing expected ICS states to observed values, we will not create a model that depends upon interarrival times. In future works, the data may be useful to signify whether a state change is occurring. For a similar reason, accelerometer data was not used in our model; classification results were higher for gyrometer input. Though the data from both interarrival times and accelerometer are useful for detecting anomalies in both the network and sensor data streams, there is not a significant improvement in classification performance when combining all data streams with our current machine learning techniques.

2) Preprocessing:

a) Feature Transformation: After raw sensor and packet data is obtained, a series of preprocessing steps are performed. The absolute values of the raw values were taken in order to specifically detect the magnitude of the rotational and straight-line forces; this is done since the direction itself is oscillatory around the axis, so the magnitude is the primary source of classification information. For this reason, we use an absolute value to reduce the neural network learning needed to find the magnitude. Payload data contains constant noise from a variety of packets that ping and maintain the connection. By taking a moving average of 100 of the packet bitstreams we were able to improve classification by accounting for a constant amount of noise on the network.

b) Downsampling for Time alignment: After the low level preprocessing, there are still many challenges with dealing aligning samples of packet data to the corresponding sensor data. This time alignment step is important because we need to classify the ICS state from the sensor and packet data for the same time period to decide if there is an anomaly during that period. The first problem is the two data types have varying amounts of data for each period of time because the sensor data is arriving in constant intervals while the packet is arriving sporadically. In order to have around the same amount of data for the time period we downsample the sensor data taking every other sensor reading. This reduction of sensor data to half its readings allows payload data to be aligned to its corresponding sensor readings in time.

c) Windowing for Delays: Other timing problems are specific to the problem of having multiple classifiers on an ICS. One problem is that the packet payload messages sent on the network take some time to impact the ICS actuators, especially mechanical peripherals because of start up transients. This adds delay between the observed state from the PCAP analysis and the observed state from the sensors. Furthermore, the packet payload arrival time varies depending on whether an ICS state transition is occurring, which has the effect of giving a variable sampling rate to the ICS. This means each payload cannot directly be correlated with a sensor output, because many payloads can be correlated to only a few ICS sensor changes, and vice versa.

These two timing effects can be mitigated by using a larger input size for our machine learning model. As sample input size increases, the variable sampling rate and differences among sample rates become less impactful. For our purposes, we used 100 samples of payload data and 100 samples of sensor data (ending at the same point in time) as input for each CNN model. 100 samples seems to be a conservative sample size that works for our classification because the sensor visuals seem to show that the state changes happen over less than 100 samples; by increasing the sample input size to the ratio of the number of samples it takes to change states we can reduce misclassification errors to a single prediction during an ICS state change. Overall, this input sample size parameter is critical to meet custom performance metrics—a large input size may increase predictive accuracy for system states but may also increase the time for the models to identify anomalies, especially anomalies over small time periods.

3) Combining models to perform a semi-supervised anomaly detection:

First, our CNN models train and test on a series of 100 samples (s) ,for both payload and sensor data streams. For the combined algorithm for anomaly detection, we use the two previous described models and monitor when errors between the models would occur. Since the trials are about 500,000 samples each and the models predict from 100 samples, there will be about 5000 predictions per trial.

A sliding window of size 20 is used to calculate error prediction percentage over time. In other words, every group of 20 predictions, produces an error rate. The plots shown in Figure 6c visualize the results of both the payload and gyrometer sensor classifiers and then the error rate per moving window of 20 predictions. In this case, using a smaller average window size, we can control false positive to false negative ratio, which can be optimized based on the costs of a misclassification.

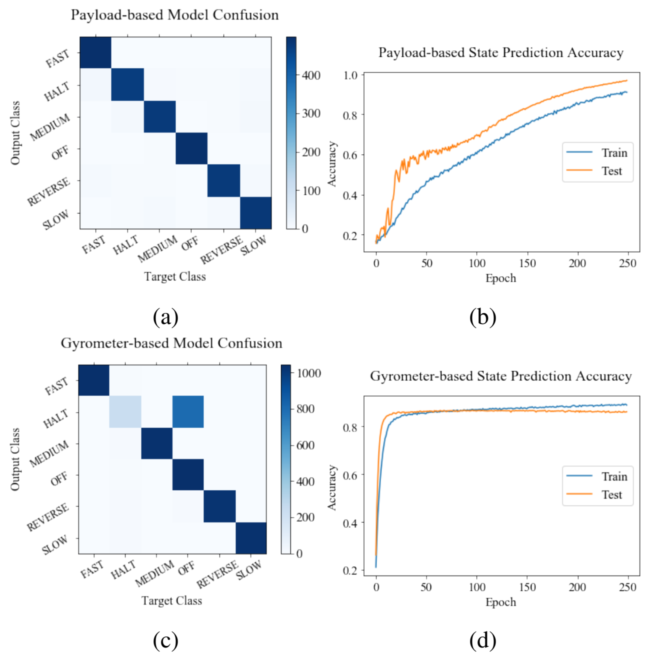


Fig. 5: Results of classification training for individual datastreams

The value of the hyperparameter window size W and threshold T are empirically chosen from observing the natural patterns on our ICS network. The selection of a moving window error rate of 20 is used because, while random misclassification can occur, after around 20 predictions the error rate is observed to be fairly low. A threshold of 18% for the error rate is used to identify anomalies since the baseline error rate for a window of size 20 is around 15% for our models. One aspect to note is that the two hyperparameters are inversely related: the greater the window size the lower the threshold needed.

VI. Results

The results of our model effectiveness depend both on the strength of our individual supervised classifiers and the techniques we use to detect discrepancies between them. While theoretically anomalies could be detected even if the individual classifiers are not high performing, it is preferred so that the result is readily interpretable. In the sections below, we visualize and analyze supervised classification results and evaluate their usage for unsupervised detection.

A. Training Results

To analyze the effectiveness of our classifiers, we visualize the confusion matrices, along with the F1 scores and weighted averages, shown in Figure 5. These confusion matrices allow us to see which ICS states can be distinguished and which states are misclassified. From the plots and learning curves in Figures 5b and 5d, it is apparent that the lesser performing model uses only the gyrometer, with near perfect classification except for misclassifications for the ‘halt’ and ’off’ states (an easily interpreted confusion). We hypothesize that the gyrometer is better suited for classifying direction of motion than the accelerometer because the acceleration magnitude is quite similar regardless of rotation direction. Whereas the gyrometer rotation signals can be in the negative direction (helping to elucidate rotation state more clearly). Moreover, the gyrometer device can also be subtly influenced by magnetic fields from the motor, which could also explain why the CNN was able to distinguish direction of rotation from the gyrometer [21]. Finally, the payload classifier shows the highest performance—but this classifier also takes some time to recognize different states so the misclassifications will be exacerbated somewhat by the latency of the classifier. Though the results of the individual data streams have misclassifications, these misclassifications may be mitigated through adjusting the threshold of classification errors for anomaly detection, discussed next.

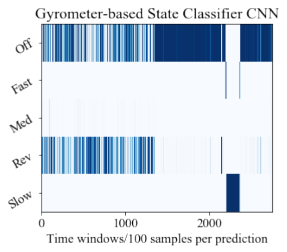
B. *Unsupervised Anomaly Detection Results: Qualitative*

Our combined model, as mentioned in our methodology, combines our classifier outputs, tracking the number of occurrences when the classifiers differ in the state prediction. When the accumulation of these errors within a window of time surpasses a certain threshold, the anomaly is marked. Visualizations of our anomaly detection for a few samples is shown in Figure 6 for gyrometer and payload classification, as well as their combined accumulated discrepancy rate in Figure 6c. As seen in the diagram, the usage of two classification models can be used to identify anomalous behavior by putting a threshold on the average discrepancy rate. In our testing, we found that a threshold of 18% worked well in flagging the anomalies, as is apparent in the plots. Though the individual supervised models themselves contain some errors in classification, comparing them in an unsupervised way allows a robust anomaly detection. As a result, this model is particularly robust to the large amount of noise and timing effects for the multiple streams of data found on an ICS system. We now turn our attention from a qualitative analysis of the anomaly detection threshold to a more quantitative view of performance across many thresholds.

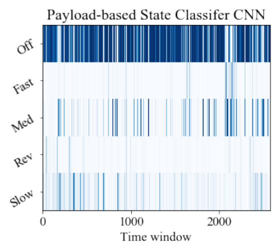
C. *Unsupervised Anomaly Detection Results: Summative*

The baseline training data includes gyrometer and packet data from 6 unique ICS state trials that have been divided into 5000 samples of input size 100. The testing dataset consists of more static data along with data with anomalous behavior caused by a MITM interacting with sensor and payload data. Our test dataset includes 18 baseline trial runs and 24 anomalous trials.

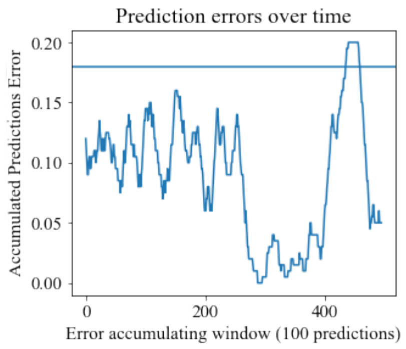
A precision-recall curve (PRC) is used to detect the precision to recall ratio as the threshold of anomaly detection is adjusted. This method can reveal to what degree the overall classifier performs greater than random chance, revealing the true skill. By sweeping our threshold from 0.0% to 100.0% of errors within a window (and keeping the window size constant) we can create a diagram where, as recall of



(a)



(b)



(c)

Fig. 6: Anomaly detection results using Gyrometer and Packet Classification

anomalies increases, the false positives also increase and precision decreases. Thus, we can observe the tradeoff for the classifier at varying thresholds. Though similar to a receiver operating characteristic (ROC), a precision recall curve is preferred when the classes are heavily imbalanced and positive classes are much more important [22]. In our results, detecting true positives has greater utility since our model has consistent results at detecting the baseline (true negative) at every threshold and will have minimal false negatives, if any. Furthermore, an ROC curve with a static true negative cannot be effectively visualized, since the x-axis will show a static specificity = FP/(FP+TN). In addition to improved visualization through the PRC curve, emphasis on recalling true positives is important since the model must be able to detect and mitigate threats before they cause a permanent major failure to the ICS system. From Figure 7, the calculated area under the precision-recall curve(AUPRC) is about 86%.

From the precision-recall curve, we took the optimal threshold where precision and recall are equal (sometimes referred to as the equal error rate point or EER). At this threshold of around 0.17, we ran the model on our test set. The performances can be statistically summarized. F1 score: 0.89, Sensitivity(Recall): 0.87, Precision: 0.88. The results are obtained by analyzing the true positive and false negatives from anomaly injections and false positives and true negatives from baseline. These results represent the strength of the classifier after it is tuned to be an optimal threshold for this dataset. We conclude that detecting this percentage of our anomalies generated is quite strong because the inserted anomalies in the system were of relatively short

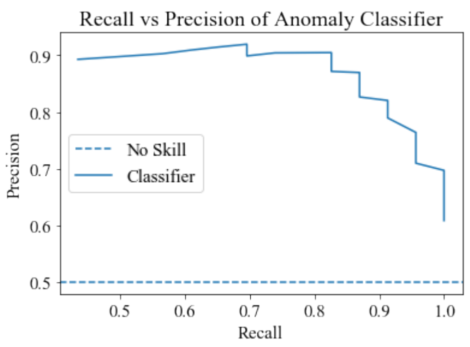


Fig. 7: Precision-Recall Curve for Anomaly Detection Results

duration. Though some anomalies were not detected, a more sustained MITM attack would eventually trigger an alarm, and this is considered likely since anomalies would need to be prolonged to cause a major system failure. Overall, our classifier is robust to the random noise of multiple classifiers and can accurately distinguish anomalies from baseline data.

D. Latency of Classifier

Another important metric is latency of prediction. Our method will have a non-zero latency due to determining anomalies only after the error rate of a window of time surpasses a certain point. For every prediction there are 100 data points of sensor data and packets, and every 20 predictions there is a potential anomaly flagged. We define latency as: $W \cdot (e_i - e_a)/r$ where W is the window (number of samples per prediction), $e_i - e_a$ are the number of predictions between the first error and the error where the anomaly threshold is crossed, and r is the sampling rate in samples per millisecond. These variables are needed to calculate the latency in milliseconds.

Prediction delays can be used to estimate the latency. For our example, the median number of predictions between the first incorrect prediction and the anomaly(threshold crossed) is 39.5. This means that about 3950 sensor and payload data were used in total before the error was confirmed. At a rate of 10 samples per milliseconds, 395 milliseconds of sensor data passes until detection. When taking account of all timing information, our combined model setup is fast enough to quickly classify and compare windows of data from two datastreams. Whether this anomaly detection is fast enough depends on the ICS system requirements and fault tolerance; acknowledging this variability, our model has controllable hyper parameters such as window size and a threshold that can be modified to increase the speed of detection.

VII. Conclusion

In summary, our classification technique reliably finds anomalies based upon the difference in prediction between two parallel neural networks, using two different, but complementary, views of the network to measure consensus: command packets and actuator sensors. This model can eventually be expanded to work with more complex and multistage ICS systems such as that of the SWaT dataset discussed in previous works. In the future, the area of multi-modal machine learning may allow more complex interaction between payload and sensor data streams. This type of model would make it easier to incorporate data that does not directly identify ICS states such as interarrival times.

References

[1] D. Pliatsios, P. Sarigiannidis, T. Lagkas, and A. G. Sarigiannidis, "A survey on scada systems: Secure protocols, incidents, threats and tactics," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 1942–1976, third quarter 2020.

[2] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, "A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3453–3495, Fourth quarter 2018.

[3] B. Babu, T. Ijyas, Muneer P., and J. Varghese, "Security issues in scada based industrial control systems," in *2017 2nd International Conference on Anti-Cyber Crimes (ICACC)*, March 2017, pp. 47–51.

[4] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A. Sadeghi, M. Maniatakos, and R. Karri, "The cybersecurity landscape in industrial control systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1039–1057, May 2016.

[5] S. D. Anton, S. Kanoor, D. Fraunholz, and H. D. Schotten, "Evaluation of machine learning-based anomaly detection algorithms on an industrial modbus/tcp data set," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018, pp. 1–9.

[6] "Detecting cyber attacks in industrial control systems using convolutional neural networks."

[7] M. Kravchik and A. Shabtai, "Detecting cyberattacks in industrial control systems using convolutional neural networks," 2018.

[8] S. D. Anton, S. Kanoor, D. Fraunholz, and H. D. Schotten, "Evaluation of machine learning-based anomaly detection algorithms on an industrial modbus/tcp data set," 2019.

[9] X. He, E. Robards, R. Gamble, and M. Papa, "Anomaly detection sensors for a modbus-based oil and gas well-monitoring system," in *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*. IEEE, 2019, pp. 1–8.

[10] J. Kim, J.-H. Yun, and H. C. Kim, "Anomaly detection for industrial control systems using sequence-to-sequence neural networks," 2019.

[11] J. Liu, X. Song, Y. Zhou, X. Peng, Y. Zhang, P. Liu, and D. Wu, "Deep anomaly detection in packet payload," 2019.

[12] S. Dasgupta, M. L. Littman, and D. McAllester, "Pac generalization bounds for co-training," *Advances in neural information processing systems*, vol. 1, pp. 375–382, 2002.

[13] F. Chollet, *Deep Learning with Python*. Manning Publications, 2017.

[14] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series."

[17] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural networks*, vol. 1, no. 4, pp. 339–356, 1988.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] H. Lan, X. Zhu, J. Sun, and S. Li, "Traffic data classification to detect man-in-the-middle attacks in industrial control system," *2019 6th International Conference on Dependable Systems and Their Applications (DSA)*, pp. 3453–3495, 2019.

[20] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[21] K. Kunze, G. Bahle, P. Lukowicz, and K. Partridge, "Can magnetic field sensors replace gyroscopes in wearable sensing applications?" in *International Symposium on Wearable Computers (ISWC) 2010*. IEEE, 2010, pp. 1–4.

[22] J. Davis and M. Goadrich, "The Relationship Between Precision-Recall and ROC Curves," in *ICML '06: Proceedings of the 23rd international conference on Machine learning*. New York, NY, USA: ACM, 2006, pp. 233–240. [Online]. Available: http://portal.acm.org/citation.cfm?id=1143874