# Network Anomaly Detection

## Introduction

Industrial Internet of Things, aka IIOT, has connected billions of devices to the internet. While this allows for incredible benefits, it also exposes the devices to threats over the internet, also known as cyber threats. According to recent reports, roughly one in three data breaches now involve an IOT device. This shows just how significant target IOT devices are for attackers. This is mainly because many IOT devices lack basic security, which makes them easy prey for attackers [1][2]. There have been several incidents of IOT devices being attacked. One major incident was the Mirai botnet attack. This attack showed how IoT gadgets can be collectively exploited to disrupt internet services via a denial-of-service attack, aka DOS.

Because of this vulnerability issue, IOT devices must implement advanced security measures beyond the standard firewalls, which can quickly become insufficient when dealing with evolving IoT malware and network-based attacks. This brings us to the topic that we will be covering today, which is a form of Intrusion Detection tailored for IoT devices. An Intrusion Detection System, aka IDS, is a system that will be used to monitor both device and network activity, looking for malicious behavior; if found, it will trigger an alert [2]. Because of the extreme complexity of IoT devices, rather than looking at an algorithm approach, we will be looking at a machine learning approach instead. We can use machine learning to automatically recognize patterns of attacks vs regular behavior in the network [2]. Ensemble learning methods combine multiple weak learners to make a stronger model. These models have shown promise regarding intrusion detection [5]. We will use this method with an industrial IoT security dataset known as the BRUIIoT dataset, which contains a broad range of scenarios for attack and expected behavior. This aims to build a model to distinguish between malicious traffic and routine operations to create an effective defense in IoT systems.

## Prior Work

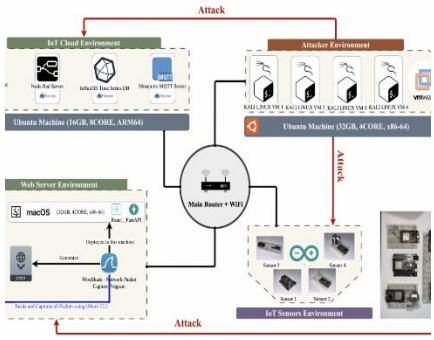
In the past, many efforts have been made to solve attack/anomaly detection through networks. Many of these relied heavily on rule-based systems and statistical network traffic analysis. Despite these efforts, they could never perfectly capture the dynamic behavior of modern IIoT[6]. However, recently, there has been a different approach to these problems, one of which is the dataset method we will be using. A few examples of this are the BoTIoT (2018) and the TON IoT (2020) from UNSW, and the most recent Edge-IIoTset(2022) [3][4]. These datasets were created in a realistic testbed environment to include the types of attacks and other traffic patterns commonly seen in IoT systems. The Edge-IIoTset dataset contains 14 different attacks, ranging from Denial-of-Service (DoS/DDoS) floods, information gathering (scanning/probing), man-in-the-middle, injection exploits, and malware (backdoor/RAT, ransomware [3]. Datasets such as these have allowed researchers to train and benchmark various Intrusion Detection Systems. Along with dataset defense systems, other forms of defense have also been created, one of which is known as a signature-based intrusion detection system, which looks for known malicious patterns. Machine Learning and Deep Learning have come to play a critical role in modern anomaly-based IDS. This is mainly due to their ability to generalize and detect attacks. This uses classifiers such as SVM, k-NN, and decision trees. However, recently it has moved more towards ensemble methods and deep neural networks because of their higher accuracy [5].

## Implementation

### *Understanding the Attacks*

Before implementing an Intrusion Detection System, we must first understand what these devices look like and how an attacker might try to exploit them. An IIoT device has several environments, including sensors, cloud, a server for monitoring traffic, and a central router[4] [7] [6]. These can be exploited by intercepting/altering packets through the transport layer, Distributed Denial of Service (DDoS), and exploiting various network protocol vulnerabilities. The structure of an IIoT device can be seen in Figure 1.

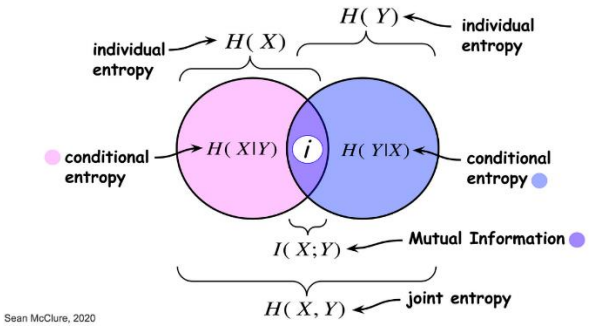
Figure 1[4]




***Data Collection and Preparation***

We will use the BRUIIoT dataset to train and evaluate the models for this project. This dataset was collected using a realistic Industrial IoT network testbed and was released in early 2025 to help with research in IIoT security [4]. This dataset contains network traffic under both regular operation and in various cyberattacks. This data was collected using 59 million packets, which were then processed to create 3 million labeled network traffic records for training [4]. This is done using Mutual Information (MI) based feature selection[4][8][9]. Mutual Information-based feature selection is used by measuring mutual dependencies between two variables randomly, or in easier words it is the amount of information that can be obtained about one variable through the other variable. A nonnegative value indicates the degree of dependence between the two variables, with the idea that the higher the number, the greater the dependence. [8][9].

Figure 2[9]




While the author says that this resulted in around 150 features for the dataset, it came out to around 250 features (possible typo) when downloaded. These features will be used to capture statistical patterns. For example, the features include things such as, counts of unique source IPs communicating with a device in a time window, ratios of certain packet types, TCP protocol flags patterns, frequencies of specific request types, etc. The goal is to use these subtle differences between normal and attack traffic, for example, a DDoS may cause an abnormally high number of unique IPs due to the multiple bots that it uses.

We will rely heavily on these provided features for our implementation since they have already been expertly selected. To help ensure consistency and ensure gradient boosting models converge faster, I normalized feature distributions by min-max scaling or standardization. I also minimized feature importance from the output of the tree-based models.

***Machine Learning and Model Tuning***

To implement this system we evaluated several machine learning models. These models include:

- **AdaBoost**, aka Adaptive Boosting, is a boosting algorithm that builds an ensemble of weak learners in a sequence so that each new model focuses on the mistakes made by the previous model. This was used with decision tree-based learners.
- **Random Forest**: Random forest is an ensemble of decision trees that are trained with bootstrap aggregating. When training a random forest, each tree will be trained on a random subset of the dataset and will also use a random subset of features at each split, which helps to introduce some diversity between the different trees. The prediction is then made by using majority voting on its trees. However, because the trees can grow too deep, we did see some overfitting at the start of training.
- **Extra Trees:** Extra trees are very similar to Random Forest; however, instead of computing the optimal split, they will choose to split randomly. In return, this reduces variance while becoming more

biased. Extra Trees were included in hopes that they would be less prone to overfitting due to their extra randomness.

- Gradient Boosting: Gradient boosting is similar to AdaBoost in that it builds the trees sequentially; however, it optimizes for binary classification loss by fitting each new tree with the ensemble's residual errors up to that point.
- XGBoost: Boost is an improved implementation of gradient boosting, including regularization improvement. This model was involved because it often outperforms Gradient Boosting on large datasets.
- CatBoost: CatBoost is another Model from the gradient boosting library, which is known for its fast learning and built-in handling of categorical variables. CatBoosting also uses an order boosting technique to reduce overfitting, which is why it was used.

When starting the training for these models, each model was initially in something of a base form, meaning that there was no tuning for hyperparameters. This was done to get an initial idea of how the models were performing because each model would need its own adjustments to improve it. We started with the library defaults; however, when it was time to fine-tune some of the models, I used a systematic hyperparameter search by using a combination of grid search and some manual adjustment.

- AdaBoost: When I was using the default 50 estimators of depth 1, our model was underfit, this lead me to try larger numbers including 100,300, and 500) which led to an improvement in the model when paired with the following learning rates: 0.01,0.1,0.5. The model that performed the best used 300 n_estimators paired with a learning rate of 0.1
- Random Forest: The main parameters that were used in the Random Forest Model were the number of trees, maximum tree depth, minimum samples split, and minimum samples per leaf. When run with the default 100 tree depth, the model overfitted. In order to mitigate this, I ended up limiting the depth to 10, and 20, along with this, I also increased the n_estimators to 100, 300, and 500. The best model used a max depth of 20 and 300 trees.
- Extra Trees: When working with the Extra Trees, I had a very similar approach to the Random Forest, and this model also performed the best when the depth was limited to 10 and 20, and when paired with the n_estimators 100,300, and 500. Besides this, this was paired with max_features being set to “sqrt”. The best model used 300 trees, a max depth of 20, and a min sample of 4.
- Gradient Boosting: When trained with the default parameters of 100 estimators, depth of 3, and learning rate of 0.1, I noticed that while the model performed well, it had a slightly lower recall. In order to improve that, I tuned the n_estimators, learning rate, and the max depth. The most efficient version ended up using 300 estimators, learning rate of 0.05 and a tree depth of 4. This drastically improved the performance of the model.
- XGBoost: While training the XGBoost, there was very little need for any tuning of the model. With the default parameters of maxdepth = 6 and a learning rate of 0.3, the model performed very well, so to avoid wasted time and complexity, I ended up leaving it as is.
- CatBoost: CatBoost also performed really well without any major tuning. I believe this was due to CatBoost's internal mechanics, like ordered boosting and automatic handling of categorical features. However like XGBoost I ended up leaving it as is. Throughout the Training Process I used cross validation on the training sets to help evaluate the parameter combinations. I also moinotored the difference in the training accuracy vs the testing accuracy to make sure that no overfitting was occurring. The

models RandomForest and Gradient Boosting both initially showed signs of overfitting, which led me to add regularization. The models XGBoost and CatBoost had less overfitting out of the box because of there built in mechanisms.

At the end of the training process I ended up with a tuned version of AdaBoost, Random Forest, Extra Trees, and Gradient Boost. This was because the tuned versions of these models performed the best, while the models XGBoost and CatBoost remained very similar to their default settings because that is was performed the best for those models. The training process took quite some time due to the size of the dataset and the number of models that needed to be trained. I found that each training session took around a day and a half to complete, this made it quite difficult to try several variations of the models because it would take so long to train. All training was done using python paired with the scikit-learn libraries and the native libraries for XGBoost and CatBoost.

***Results***

In order to ensure that the models were performing well, I used a variety of metrics. The Metrics include Accuracy, Precision, Recall, F1-score, and MCC, aka Matthews Correlation Coefficient, and ROC-AUC, aka Receiver Opertationg Characteristic and Area Under Curve).

- Accuracy: defined as the fraction of total instances that were correctly classified
- Precision: the fraction of predicited attacks that were actually attacks, which was used to determine false alarms
- Recall: This is the fraction of actual attacks that were correctly detected and is used to determine how many attacks we caught.
- Mathews Correlation Coefficient: This is a value that ranges from -1 to 1; it considers all categories from a confusion Matrix, like True Positive, True Negative, False Positive, False Negative. And MCC score of 1 means that it is making perfect predictions, 0 means that it is no better then a random prediction, while -1 means it is not picking correctly at all [5].
- ROC-AUC: This is used to plot the true positive rates against the false positive rates across different thresholds. AUC ranges from 0.5 to 1 0.5, which means that it is randomly picking, one means the model is picking perfectly.

Through these Metrics we will be treating positive classes as attacks since that is the primary purpose of these models. The Performance Metrics for the models can be seen in Figure 3:

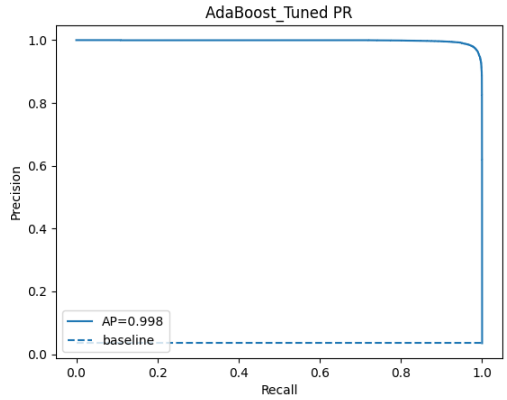
Figure 3

| Model | Accuracy | Precision | Recall | F1-score | MCC | ROC-AUC |
|---|---|---|---|---|---|---|
| XGBoost | 0.999047 | 0.984702 | 0.988101 | 0.986399 | 0.985907 | 0.99997 |
| CatBoost | 0.999222 | 0.986717 | 0.991088 | 0.988898 | 0.988498 | 0.999979 |
| AdaBoost_Tuned | 0.998482 | 0.975422 | 0.981301 | 0.978353 | 0.977571 | 0.99992 |
| GradientBoost_Tuned | 0.998365 | 0.972283 | 0.981198 | 0.97672 | 0.975884 | 0.999898 |
| RandomForest_Tuned | 0.997239 | 0.954869 | 0.966723 | 0.96076 | 0.959349 | 0.999738 |
| ExtraTrees Tuned | 0.99747 | 0.955853 | 0.972544 | 0.964126 | 0.962854 | 0.999715 |

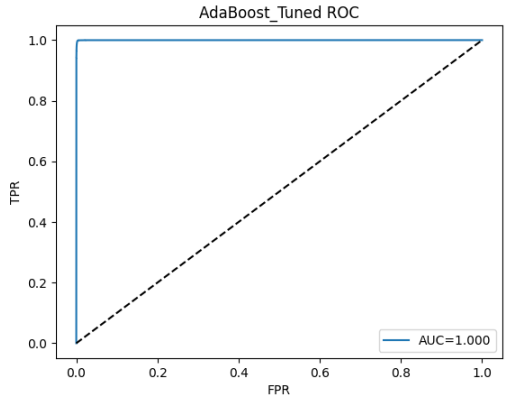
As shown in Figure 3, All the models achieved a very high accuracy and detection rate, while this was concerning at first because I believe that all the models were overfitting, however after looking at the other metrics I believe that they weren't overfitting because even on the testing and recall the models still perform very accurately.

- AdaBoost, When I was initially training the AdaBoost had the lowest precision score with a score of (0.8687) and it had a F1 score of (0.8956) while this still isn't necessarily bad, I believed that we could improve it, so after some fine tuning we were able to increase the performance of the model dramatically. The new Precision score shot up to 0.9756 and had a Recall of 0.9813 while giving an F1 score of 0.9783. Below is the ROC Curve and the Precision Recall Curve:

Precision Recall

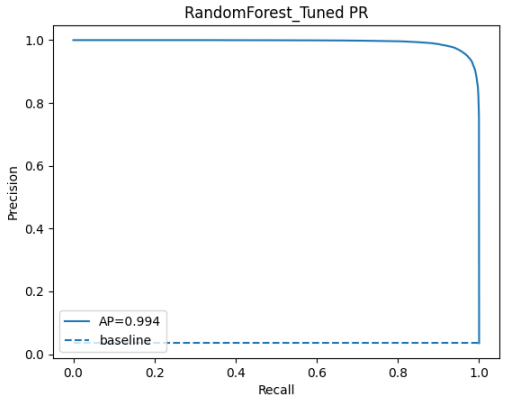



ROC Curve

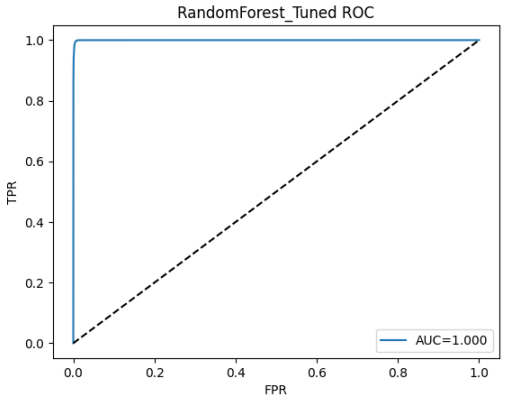



- Random Forest: When Training the initial Random Forest, the Base form already performed very well with an accuracy of 99.88%, precision score of 0.9798, Recall score of 0.9856, an F1 Score of 98.27, When I saw these numbers I was concerned about overfitting do to such a high score, however after several modifications I was still getting a similar score, however I believe because the dataset was made and cleaned by experts for this specific, purpose I believe that this could be due to there being less noise in the data which is resulting it a much more accurate model, This being said, when I was modifying the Random Forest to check for overfitting while the accuracy did in fact decrease it still remained very high with an accuracy of 99.84%, precision score of 0.9723, Recall score of 0.9812, an F1 Score of 97.67.

Precision Recall

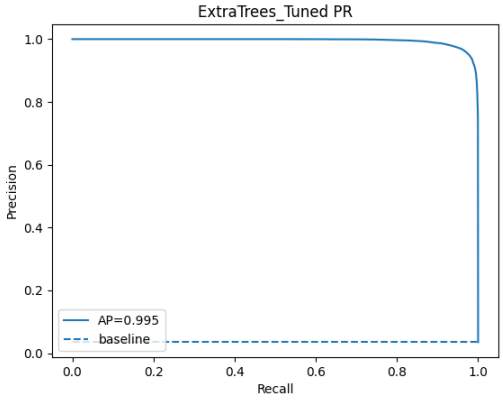



ROC Curve

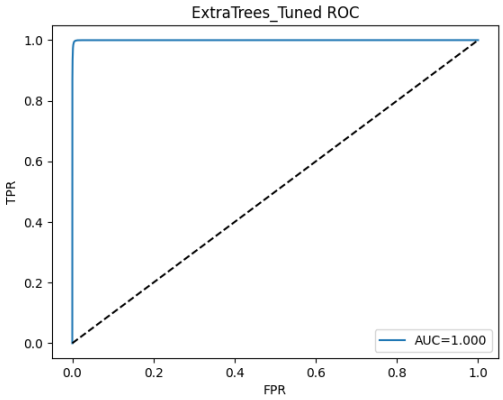



- Extra Trees: The Extra Trees model also performed very well with an accuracy of 99.82%, a Precision of 0.968, a Recall of 0.981, and an F1 Score of 0.975. Because all of the models were ran in one file I had the same scare as a did with the Random Forest and Tried to fix any possible overfitting, this resulted in a very slight decrease in accuracy, with an accuracy of .9974&, Precision of 0.,9559, Recall of 0.9725, and an F1 Score of 0.9641, however the model still performed very well.

Precision Recall

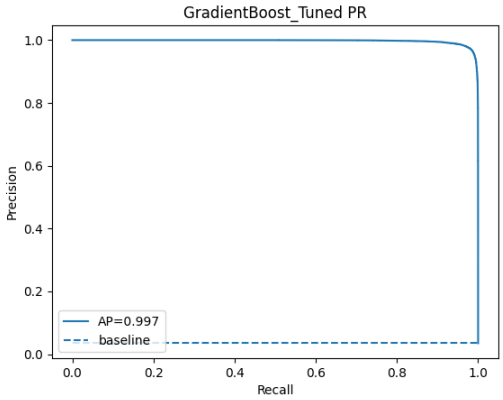



ROC Curve

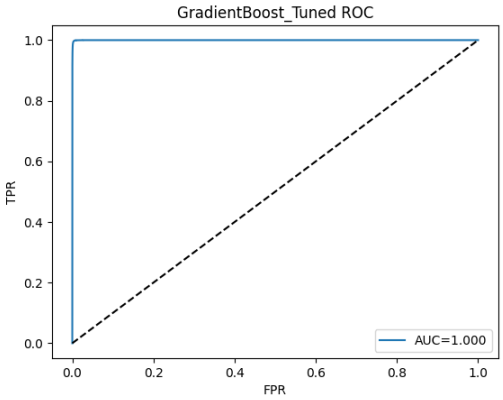



- Gradient Boosting: When starting the initial training the Gradient Boostin initially had a lower performance, with an accuracy of 99.70%, Precision of 0.9569, Recall of 0.9587, and a F1 Score of 0.9579, After Tuning, the model achieved an Accuracy of 00.84%, Precision of 97.23, Recall of 0.9812, and a F1 Score of 0.9767.

Precision Recall

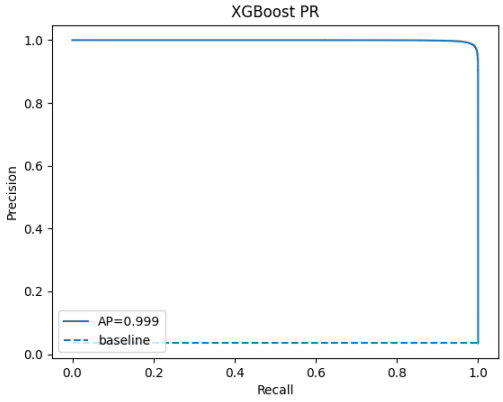


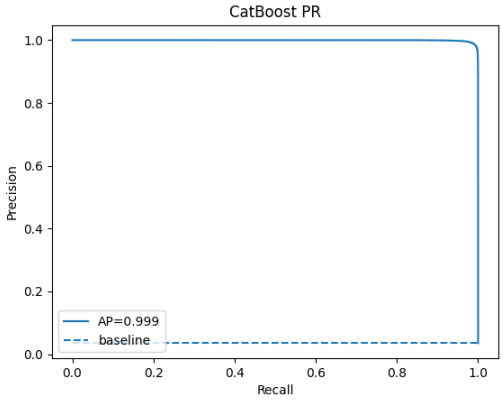

ROC Curve




- XGBoost: when I was training the XGBoost I was at first very concerned about the performance however, because of the previous models performances's even after tuning, I believe that this model was performing as intended from the start. The model gave an accuracy of 99.9047%, a Precision of 98.47%, a Recall of 0.9881%, and an F1 Score of 98.64%.

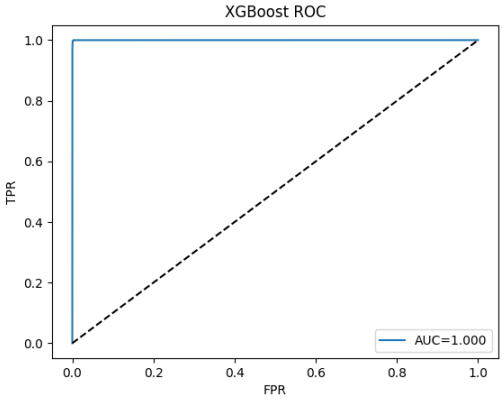
Precision Recall

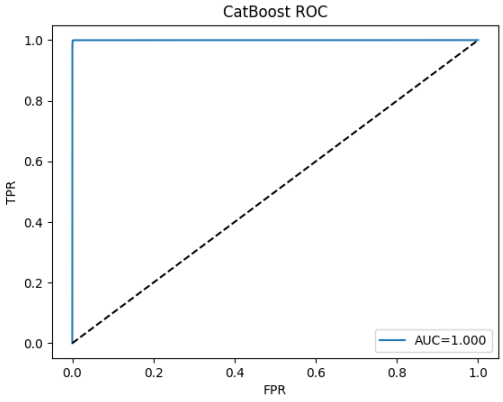



ROC Curve

ROC Curve




- CatBoost: CatBoost was the top performer of all of the models by a very slight margin. It achieved an accuracy of 99.9222%, Precision of 0.9867, Recall of 0.9911, and a F1 Score of 0.9889, Same as XGBoost the model performed very well out of the box, and after seeing how all of the other models were performing I don't believe that it was overfitting.

Precision Recall

***Conclusion***:

In Conclusion while Industrial Internet of Things Devices are very important in the workforce, they are also very vulnerable to cyber threats. This project's primary goal was to increase this security through the use of Machine Learning Models. Using the BRURIIoT dataset, that was created by a team of experts that used Mutual Information to narrow down the dataset from 59 million rows to 3 million rows that were ready to be used for training, we were able to train six different Models including: AdaBoost, XGBoost, CatBoost, Random Forest, Gradient Boost, and Extra trees. While there was major concern for overfitting at first due to the extremely metric scores, I have concluded that this is due to the dataset having little to no noise because it was cleaned by experts with the specific purpose of training models to detect attacks on IIoT devices. This left us with these very

high-performing models that are able to act as an intrusion detection system for IIoT devices. This shows that with the proper data preparation and time, we could implement safeguards for IIoT devices all around the globe, which will result in a safer working environment.

*Future Work*

This Intrusion Detection System was designed to work with IIoT Devices and used a binary system of detecting the attack, I believe that this could be developed even further to what an attack is as well. This would allow for companies to understand the type of attack that is going after there device in order to better prepare against it.

**Works Cited**


1. **Verizon DBIR (2024) – IoT Breach Statistics:** Sean Blanton, *"IoT Security Risks: Stats and Trends to Know in 2025,"* JumpCloud Blog, Jan. 10, 2025. (Highlights from Verizon's 2024 Data Breach Investigations Report) jumpcloud.com

2. **Kantharaju *et al.* (2024) – SAPGAN IDS for IoT:** V. Kantharaju, H. Suresh, M. Niranjanamurthy, S.I. Ansarullah, F. Amin, A. Alabrah, *"Machine learning based intrusion detection framework for detecting security attacks in internet of things,"* Scientific Reports, vol. 14, Article 30275, Dec. 2024. nature.com

3. **Ferrag *et al.* (2023) – Edge-IIoTset Dataset:** M.A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, H. Janicke, *"Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning,"* IEEE DataPort (Dataset), DOI:10.21227/mbc1-1h68, Apr. 2023. ieee-dataport.org

4. **Al Islam *et al.* (2025) – BRURIIoT Dataset:** F. Al Islam, M. Shamsuzzaman, M. S. Islam, S.A. Sakib, A.S.M.M. Rahaman, *"BRURIIoT: A Dataset for Network Anomaly Detection in IIoT with an Enhanced Feature Engineering Approach,"* IEEE DataPort (Dataset), DOI:10.21227/fqqe-g413, Mar. 2025. ieee-dataport.orgieee-dataport.org

5. **Chicco & Jurman (2020) – MCC vs Accuracy/F1:** D. Chicco and G. Jurman, *"The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation,"* BMC Genomics, vol. 21, Article 6, Jan. 2020.

6. Visconti P, Rausa G, Del-Valle-Soto C, Velázquez R, Donato Cafagna, Fazio RD. 2024. Machine Learning and IoT-Based Solutions in Industrial Applications for Smart Manufacturing: A Critical Review. Future Internet. 16(11):394–394. doi:https://doi.org/10.3390/fi16110394. https://www.mdpi.com/1999-5903/16/11/394.

7. What Is Industrial IoT (IIoT)? Cisco. https://www.cisco.com/c/en/us/solutions/internet-of-things/what-is-industrial-iot.html#~role-of-it.

8. Information Gain and Mutual Information for Machine Learning. 2024 Apr 15. GeeksforGeeks. https://www.geeksforgeeks.org/information-gain-and-mutual-information-for-machine-learning/.

9. McClure S. 2020 Nov 7. A Deep Conceptual Guide to Mutual Information - The Startup - Medium. Medium. https://medium.com/swlh/a-deep-conceptual-guide-to-mutual-information-a5021031fad0.