Malmo University
Report Project

Autoencoders for Malware Detection in IoT Device

Vu Phan
Department of Computer Science and Media Technology
Malmo University
Malmo, Sweden
vuphann25@gmail.com

Pietro Benecchi
Department of electronics, computer science and bioengineering
Polytechnic of Milan
Milan, Italy
pietro.benecchi@mail.polimi.it

Abstract

This paper explores the use of autoencoders for malware detection in IoT devices, comparing Basic, Variational, GAN, and LSTM architectures. Building on previous machine learning approaches, the study evaluates these models using the IoT-23 dataset, focusing on optimizing detection thresholds to maximize recall while maintaining accuracy.

I. Introduction

The Internet of Things (IoT) has transformed everyday life and is getting increasingly integrated across various industries, from automating daily tasks to powering industrial sensors and measurement devices. According to Vailshery [3], the number of connected devices worldwide is expected to continue to grow, and is forecasted to almost double from 15.9 billion in 2023 to more than 32.1 billion IoT devices in 2030. While IoT offers numerous benefits, there are still plenty of challenges and issues that need to be considered, particularly in security. Risks in IoT are increasing at an alarming rate and are calling for protection of data, which has become a significant concern for both manufacturers and consumers [4].

Due to the complex and integrating nature of IoT systems, traditional security measures like encryption, authentication, access control and network security have been struggling to keep up with the evolving cyber threats in IoT environments [6]. While essential, the existing security methods alone are not sufficient to fully secure the IoT ecosystem. Recent advancements in machine learning (ML) and deep learning (DL) have shown promising results over the last few years, where [6] recognize ML and DL as powerful techniques for analyzing data patterns, enabling the identification of 'normal' and 'abnormal' behavior based on how IoT components and devices operate within their environment. By utilizing anomaly detection, these techniques are able to yield crucial actionable information to various sectors, raising its significance to secure IoT networks [5].

Building upon these advancements, this study aims to contribute to the existing body of knowledge and investigate the effectiveness of deep learning techniques for anomaly detection. The techniques will be used to determine whether it is able to significantly improve the performance metrics compared to other classical approaches and enhance the detection of malicious traffic in IoT networks.

For this research, a recently published dataset, namely the IoT-23 dataset will be used which is a large dataset containing both malicious and benign IoT traffic especially created for researchers to help develop machine learning algorithms.

II. Related Works

Previous research on the topic has explored various methodologies for malware detection in IoT networks, using both machine learning and deep learning techniques. These studies have aimed to improve the accuracy and efficiency of network anomaly detection by evaluating different algorithms, datasets, and modeling approaches. For this study, numerous related works involving the IoT-23 dataset were explored.

In a study conducted by [7], a comparative analysis was emphasized on using some of the ML and DL methods for network anomaly detection, evaluating their performance and time cost to identify which of the algorithms had the best performance. The experiments were performed on the IoT-23 dataset, where the specific algorithms that were utilized for the experiments can be seen on Table 1.

Table I: **Categorization of techniques experimented on IoT-23 dataset.**

Method	**Literature**
SVM, Decision Tree, Naive Bayes, CNN	*Liang & Vankayalapati (2021)*
Random Forest, DNN, LSTM	*Dutta, Choraś, Pawlicki & Kozik (2020)*
CNN, LSTM	*Sahu, A. K., Sharma, S., Tanveer, M., & Raja, R. (2021)*

Among the tested algorithms, the Decision Tree model demonstrated the highest accuracy (0.73) while also achieving the lowest computational cost of 3 seconds, resulting in it being the most efficient algorithm in their study. Although, it is mentioned that while CNN had a lower testing accuracy (0.6935) and a higher time cost (242 seconds) than Decision Trees, it is possible for CNN to have better performance when dealing with a more complex dataset.

In a related study [8], the focus was on using deep learning for network anomaly and cyber-attack detection in Internet of Things (IoT) environments. The study was conducted with the help of three different datasets, one of which was the IoT-23 dataset. The authors highlight the fact that given the existence of enormous network traffics, traditional methods of machine learning implemented for network anomaly detection are deemed as inefficient. In alignment with what [6] recognized, Deep learning techniques have increasingly demonstrated their success in detecting and classifying anomalies at both the network and host levels [8].
To address these challenges, the authors implemented both standalone deep learning models of Deep Neural Networks (DNN) and Long Short-Term Memory (LSTM), as well as an hybrid ensemble approach, which combines both models to address the issue of network anomaly detection in IoT. The results were tested and compared against other classical machine learning techniques such as Random Forest (RF) and Support Vector Machine (SVM). The results showed that both the individual deep learning models and the ensemble approach outperformed the traditional methods, demonstrating the advantages of deep learning in handling complex network anomaly detection tasks.

Similar to the previous statements by [6] and [8], [9] shares the sentiment that the existing security mechanisms for machine learning models in IoT devices address only a limited range of attacks and rely on outdated or limited datasets for evaluation, highlighting the need for Deep Learning models to fill that gap. [9] proposes a hybrid model to tackle the attack detection problem by employing a Convolutional Neural Network (CNN) for feature extraction, followed by a Long Short-Term Memory (LSTM) network for classification.

These collective findings share the commonality of a growing interest on the potential of deep learning methodologies, in enhancing IoT security by overcoming the limitations of traditional approaches to offer more efficient solutions for anomaly and attack detection.

III. Dataset

The dataset utilized for this study, namely the IoT-23 dataset, was first published in January 2020 and designed to support research and development of machine learning algorithms [2]. It is a publicly available labeled dataset containing network traffic captures of both benign and malicious IoT network activity.
The dataset consists of **23 network traffic captures**, or scenarios, divided into two main categories:
Malware-infected IoT network traffic (20 captures), and Benign IoT network traffic (3 captures).
Each malicious scenario corresponds to a specific malware sample executed on a Raspberry Pi device, where the attacks include various botnets and attack patterns that simulate real-world cyber threats. For the benign scenarios, the captures contain normal network traffic generated by **real IoT devices** (Philips Hue Smart LED Lamp, Amazon Echo and Somfy Smart Doorlock) operating in an unrestricted internet environment.
The open-source software platform Zeek was used to perform network analysis, providing detailed connection metadata on the network flow by extracting structured logs in the format of conn.log.labeled. The Zeek conn.log file is generated from the Zeek network analyser using the original pcap file. The features and definition for it can be found on Table II.

A. Data Preprocessing

Before analyzing the dataset, preprocessing was necessary to clean the data and retain only the most relevant features for our study. Given the dataset's large size, we opted to use a lighter version provided by the researchers, which excludes pcap files and contains only labeled flows. This decision was made to optimize computational efficiency and meet time constraints.
Our preprocessing approach is strongly referenced by [7] to allow for near-identical direct comparison of results. This is to ensure consistency and reproducibility in the analysis, and eventually minimize any potential biases that might arise from differing preprocessing methods.

Initially, the Python library Pandas was used to load all 23 separate captures of the IoT-23 dataset into dataframes. Each capture was read while skipping the first 10 rows and loading the next 100,000 rows. These dataframes were then merged into a single dataset and cleaned by removing variables that had no impact on the results. The excluded variables include: *ts, uid, id.orig h, id.orig p, id.resp h, id.resp p, service, local orig, local resp, and history.* Furthermore, categorical variables such as proto and conn_state were converted into numerical representations using dummy encoding. Missing values were replaced with 0, and boolean values (False/True) were mapped to 0 and 1, where 0 represents benign data and 1 indicates malicious data. Lastly, the combined dataset was saved as the *"iot23_combined_complete.csv"* file. It contains a total of 1,444,674 records across 25 features.
Each of the 20 malicious scenarios is labeled based on the type of attack or malware detected in the network traffic, which can be seen on Table 2 and include: *Attack, C&C, C&C-FileDownload, C&C-HeartBeat, C&C-HeartBeatAttack, C&C-HeartBeat-FileDownload, C&C-*

Mirai, C&CTorii, DDoS, FileDownload, Okiru, OkiruAttack, and PartOfAHorizontalPortScan.

Table II: **Compilation of features and description for our IoT-23 dataset.**

Feature	**Definition**
duration	Length of the connection (seconds).
orig_bytes	Bytes sent from source to destination.
resp_bytes	Bytes received by source from destination.
missed bytes	Bytes lost during the connection.
orig_pkts	Packets sent from source to destination.
orig_ip_bytes	IP-layer bytes sent by source.
resp_pkts	Packets received by source from destination.
resp_ip_bytes	IP-layer bytes received by source.
label	Connection classification (benign or malicious).
proto_icmp	ICMP protocol usage (binary).
proto_tcp	TCP protocol usage (binary).
proto_udp	UDP protocol usage (binary).
conn_state_OTH	Other connection state (binary).
conn_state_REJ	Connection rejected (binary).
conn_state_RSTO	Connection reset (binary).
conn_state_RSTOS0	Connection reset in SYN+ACK state (binary).
conn_state_RSTR	Connection reset with RST flag (binary).
conn_state_RSTRH	Connection reset with handshake (binary).
conn_state_S0	Incomplete connection (binary).
conn_state_S1	Waiting for response (binary).
conn_state_S2	Waiting for events (binary).
conn_state_S3	Connection finishing (binary).
conn_state_SF	Successful connection (binary).
conn_state_SH	Secure handshake (binary).
conn_state_SHR	Secure handshake with retry (binary).

IV. Methods

Autoencoders (AEs) are neural network architectures designed for unsupervised representation learning. They consist of an encoder, which maps input data to a lower-dimensional latent space, and a decoder, which reconstructs the input from this representation. Autoencoders have been widely used in dimensionality reduction, anomaly detection, and generative modeling. In our study, we implement four different types of autoencoders: Basic (Vanilla) Autoencoder, Variational Autoencoder (VAE), Long Short-Term Memory (LSTM) and Generative Adversarial Network(GAN), an advanced NN based on autoencoders. Each of these architectures serve a unique purpose and leverages different mechanisms to encode and decode data.

A. Autoencoder

The basic **autoencoder** is a **feedforward neural network** trained to minimize the reconstruction error between the input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$. It consists of an encoder function $f_\theta(\mathbf{x})$ that maps the input to a latent representation $\mathbf{z}$, and a decoder function $g_\phi(\mathbf{z})$ that reconstructs the input:

$$\mathbf{z} = f_\theta(\mathbf{x}), \quad \hat{\mathbf{x}} = g_\phi(\mathbf{z})$$

where θ and ϕ are the learnable parameters of the encoder and decoder, respectively. The model is optimized using a loss function such as mean squared error (MSE) or binary cross-entropy (BCE), depending on the data type. Effective methods and techniques were previously described by [1].
In our research, the encoder follows a progressively decreasing layer structure to extract latent representations. The decoder is a mirror of the encoder, progressively reconstructing the input from the latent space. A summary of our autoencoder model is shown in Table III, where the encoder compresses the 24-dimensional input through three dense layers with ReLU activation, and concluding by reducing it to a 3-dimensional latent space. The hierarchical reduction enables the model to learn a compact representation of the input data while retaining essential features.
The decoder takes the 3-dimensional latent vector as input and reverses the process by expanding the latent vector back to 24 neurons through three dense layers, also using ReLU activation. For the final layer, a sigmoid function is applied to ensure values remain within a normalized range. This structure ensures that the model learns an efficient mapping from high-dimensional input to a lower-dimensional latent space and then reconstructs the data as accurately as possible.

The model is trained using the Mean Squared Error (MSE) as the loss function, which quantifies the reconstruction error between the original input and the reconstructed output. A lower reconstruction error suggests benign data, whereas higher errors may indicate anomalies. To enhance training efficiency and prevent overfitting, we incorporated EarlyStopping. This technique monitors the validation loss during training and halts the process when no further improvement is observed, ensuring optimal generalization. In our specific case, we found out that 10 was the best number of epochs.

Table III: **Autoencoder Model Summary**

Layer (type)	**Output, Shape**	**Parameters**
Encoder		
InputLayer	None, 24	none
Dense1 (Dense)	None, 12	300
Dense2 (Dense)	None, 6	78
Dense3 (Dense)	None, 3	21
Decoder		
InputLayer	None, 3	none
Dense1 (Dense)	None, 6	24
Dense2 (Dense)	None, 12	84
Dense3 (Dense)	None, 24	312
Model Parameters		
Total Parameters	819	
Trainable Parameters	819	
Non-trainable Parameters	0	

B. *Variational Autoencoder*

We implemented VAE for anomaly detection by introducing random noise into the input data and applying KL divergence in the noise function. The added noise simulates anomalies and forces the autoencoder to differentiate between typical and anomalous patterns based on reconstruction errors. The formula for sampling from the latent space in a Variational Autoencoder (VAE) using the reparameterization trick is:

$$z = \mu(x) + \sigma(x) \cdot \epsilon$$

where:

- $\mu(x)$ is the mean of the latent variable z, predicted by the encoder.
- $\sigma(x) = \exp\big(0.5 \cdot \log(\sigma^2(x))\big)$ is the standard deviation (obtained from the log variance $\log(\sigma^2(x))$).
- ϵ is a random variable sampled from a standard normal distribution $\mathcal{N}(0, I)$.

However, despite this approach, the results have not been as effective as anticipated in terms of accuracy and recall.
Table IV summarizes our Variational Autoencoder (VAE) model, which introduces a probabilistic latent space. The encoder maps the 24-dimensional input through three dense layers with ReLU activation, generating two separate outputs: mean and log variance (log_var), each with a dimensionality of 3. These define a Gaussian distribution from which the latent variable is sampled. The decoder reconstructs the input by expanding the sampled latent vector back to 24 dimensions through two dense layers with ReLU activation, followed by a final sigmoid-activated layer to normalize the output.

Table IV: **Variational Autoencoder Model Summary**

Layer (type)	**Output, Shape**	**Parameters**
Encoder		
Dense1 (Dense)	None, 24	600
Dense2 (Dense)	None, 12	300
Dense3 (Dense)	None, 6	78
mean (Dense)	None, 3	42
log_var (Dense)	None, 3	42
Decoder		
Dense1 (Dense)	None, 6	42
Dense2 (Dense)	None, 12	84
OutputLayer (Dense)	None, 24	312
Model Parameters		
Total Parameters	4,322	
Trainable Parameters	1,440	
Non-trainable Parameters	0	
Optimizer Parameters	2,882	

C. *Generative Adversarial Networks*

In the context of anomaly detection, we also explored the use of Generative Adversarial Networks (GANs). GANs are a class of generative models that consist of two main components: the Generator and the Discriminator. These two components are trained simultaneously in a competitive setting, where the generator tries to produce realistic data that mimics the original distribution, and the discriminator attempts to differentiate between real and generated data.

The Generator G in a GAN is responsible for generating data from a random noise vector $\mathbf{z}$, which originates from a latent space. In our case, we started with a latent space of dimension 3, meaning that the generator receives a 3-dimensional vector $\mathbf{z}$ as input. The generator then transforms this latent vector into a reconstructed data point, ideally resembling the distribution of the original data:

$$\hat{\mathbf{x}} = G(\mathbf{z})$$

where $\hat{\mathbf{x}}$ is the generated data. The goal of the generator is to produce realistic data that is indistinguishable from the real data, as judged by the discriminator.

The Discriminator D, on the other hand, is a binary classifier that receives both real data from the dataset and generated data from the generator. Its task is to distinguish between real, benign samples (from the dataset) and fake samples (from the generator). The discriminator outputs a probability that the input is real:

$$D(\mathbf{x}) = \text{Prob}(\mathbf{x} \text{ is real})$$

During training, the discriminator attempts to correctly classify real versus generated data, while the generator tries to improve its output to fool the discriminator into thinking it is real.

The training process of GANs is a two-player minimax game between the generator and discriminator. The generator seeks to minimize the probability that the discriminator correctly classifies generated samples as fake, while the discriminator seeks to maximize this probability, correctly identifying real from fake samples.

The objective function for the GAN training is typically written as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log\left(1 - D(G(\mathbf{z}))\right)\right] \quad (1)$$

Here, $p_{\text{data}}(\mathbf{x})$ is the distribution of the real data, and $p_{\mathbf{z}}(\mathbf{z})$ is the distribution of the random noise input to the generator. The generator G is trained to minimize the term $\log(1 - D(G(\mathbf{z})))$, while the discriminator D is trained to maximize the two terms: $\log D(\mathbf{x})$ and $\log(1 - D(G(\mathbf{z})))$.

A summary of our GAN model is shown on Table V, where the model efficiently generates realistic 24-dimensional feature vectors through an adversarial training process. The encoder reduces the input data to a compact latent space, outputting a 3-dimensional latent vector.
The generator starts with the 3-dimensional latent vector and expands it through four dense layers with ReLU activation, and then reconstructing it back to a 24-dimensional feature vector.
The discriminator, a binary classifier, takes the 24-dimensional feature vector as input, passing it through four dense layers with ReLU activation which is followed by a sigmoid-activated output layer to finally classify the input as either real or fake.

Table V: **GAN Model Summary**

Layer	**Output, Shape**	**Parameters**
Generator		
Dense	None, 3	12
Dense	None, 6	24
Dense	None, 12	84
Dense	None, 24	312
Discriminator		
Dense	None, 24	600
Dense	None, 12	300
Dense	None, 6	78
Dense	None, 3	21
Output Layer (Dense)	None, 1	4
Model Parameters		
Total Parameters	1435	
Trainable Parameters	1435	
Non-trainable Parameters	0	

D. LSTM Autoencoders

Long Short-Term Memory (LSTM) autoencoders combine the sequence modeling capabilities of LSTM networks with the unsupervised learning framework of autoencoders. Unlike traditional feedforward autoencoders, LSTM autoencoders are specifically designed to capture temporal dependencies and patterns in sequential data. The structure is closer to AE but we have provided additional LSTM layers which is shown on Table VI:

Table VI: **LSTM Autoencoder Model Summary**

Layer	**Output, Shape**	**Parameters**
Encoder		
InputLayer	None, 1, 24	0
LSTM	None, 1, 16	2,624
LMST	None, 8	800
Decoder		
Repeated Vector	None, 1, 8	0
LSTM	None, 1, 8	544
LSTM	None, 1, 16	1,600
Dropout (0.2)	None, 24	0
TimeDistributed Layer (Dense)	None, 1, 24	408
Model Parameters		
Total Parameters	6,519	
Trainable Parameters	6,519	
Non-trainable Parameters	0	

The model, designed for sequence reconstruction consists of an encoder that processes input through two LSTM layers (24 and 12 dimensions) with ReLU activation and dropout, compressing it into a 3-dimensional latent representation via a dense bottleneck layer.
The decoder then expands this latent vector across time steps, using two LSTM layers (12 and 24 dimensions) with dropout, followed by a TimeDistributed dense layer that restores the original sequence structure with ReLU activation. The model is optimized using mean absolute error (MAE) to enhance the reconstruction accuracy.

V. Results

A. Choosing the right threshold

Choosing an appropriate threshold is a critical component in anomaly detection using autoencoders, as it governs the balance between detecting anomalies and minimizing false positives.

For this study, the performance metric selected is Recall, with the aim of minimizing false negatives. In the context of malware detection, failure to identify a malware (i.e., classifying it as benign) could lead to undetected system issues. To identify the optimal threshold, various threshold values will be tested, and the most effective combination of metrics will be determined. Although the primary objective is to enhance Recall, Accuracy will also be monitored to ensure a balanced evaluation. The threshold values will be derived from the percentiles of the reconstruction error distribution. This method has been applied to AE, VAE and LSTM. On the other hand, GANs operates differently—producing probability distributions rather than reconstruction errors. A similar approach will be applied, but based on the characteristics of the generated probability distribution:

1. Perform predictions on the test set, which consists of both benign and anomaly samples.
2. Define a threshold and select an appropriate evaluation metric.
3. The classification rule is as follows:
   If $f(x) <$ threshold, then the data is classified as benign. Otherwise, it is classified as an anomaly.

B. *Hardware*

In this study, we utilized Google Colab with standard resources, including a system with **12.7 GB RAM** and **107.7 GB** Disk space to evaluate the time performance of our proposed models. The experiments were executed on an **Intel(R) Xeon(R) CPU @ 2.20GHz**.

C. *Evaluation metrics*

In this section, we provide definitions of the metrics that we used in evaluation part.

Confusion Matrix

A **confusion matrix** is a performance measurement tool for machine learning classification problems. It is a table that allows the evaluation of the performance of a classification algorithm by comparing the predicted labels with the actual labels. The matrix consists of four key elements:

- **True Positives (TP)**: The number of instances that were correctly predicted as positive.
- **False Positives (FP)**: The number of instances that were incorrectly predicted as positive.
- **True Negatives (TN)**: The number of instances that were correctly predicted as negative.
- **False Negatives (FN)**: The number of instances that were incorrectly predicted as negative.

The confusion matrix can be represented as:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Recall and Accuracy

Two important metrics derived from the confusion matrix are **Recall** and **Accuracy**.

Recall (Sensitivity or True Positive Rate)

Recall measures the ability of a classifier to correctly identify positive instances. It is the ratio of correctly predicted positive observations to the total actual positives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

A high recall indicates that most of the actual positive instances were correctly identified, but it may also imply that some negative instances were incorrectly classified as positive.

Accuracy

Accuracy measures the overall performance of the classifier by calculating the proportion of correctly classified instances (both positive and negative) to the total number of instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

D. *Test Results*

The results of our experiments showcased various performance levels across the different models for anomaly detection. The primary evaluation metrics considered are accuracy, recall, precision, and F1-score. Benign data is represented as (0), while malicious data as (1).

1. *Autoencoders*

As shown in Table VII, the Basic Autoencoder achieved an accuracy of **86.33%** with a time cost of around **183 seconds**. The recall resulted in **99.96%**. The detailed classification report is as follows:

Table VII: **Autoencoder Results**

metrics	**precision**	**recall**	**f1-score**	**support**
0	0.00	0.00	0.00	39400
1	0.86	1.00	0.93	249535
accuracy	–	–	0.86	288935
macro avg	0.43	0.50	0.46	288935
weighted avg	0.75	0.86	0.80	288935
time cost				183 seconds

2. *Variational Autoencoders (VAE)*

The Variational Autoencoder, as shown in Table VIII underperformed compared to the Basic Autoencoder with an accuracy of only **27.38%** and a time cost of **203 seconds**. The recall was only **21.58%**. The classification report can be seen below:

Table VIII: **Variational Autoencoder Results**

metrics	precision	recall	f1-score	support
0	0.11	0.64	0.19	39400
1	0.79	0.22	0.34	249535
accuracy	–	–	0.27	288935
macro avg	0.45	0.43	0.27	288935
weighted avg	0.70	0.27	0.32	288935
time cost				203 seconds

3. Generative Adversarial Networks (GAN)

The GAN model is presented on Table IX, which performed similarly to the Normal Autoencoder by achieving an accuracy of **86.36%** and a time cost of **207 seconds**. The recall reached up to **100%**. The classification report is shown below:

Table IX: **GAN Results**

metrics	precision	recall	f1-score	support
0	0.00	0.00	0.00	39400
1	0.86	1.00	0.93	249535
accuracy	–	–	0.86	288935
macro avg	0.43	0.50	0.46	288935
weighted avg	0.75	0.86	0.80	288935
time cost				207 seconds

4. Long Short-Term Memory (LSTM)

As shown in Table X, the LSTM model demonstrated a slightly lower performance compared to Autoencoders and GANs, with an accuracy of **76.32%** and a time cost of 443 seconds. The recall resulted in **84.12%**. The classification report is as follows:

Table X: **LSTM Results**

metrics	precision	recall	f1-score	support
0	0.21	0.27	0.24	39400
1	0.88	0.84	0.86	249535
accuracy	–	–	0.76	288935
macro avg	0.55	0.56	0.55	288935
weighted avg	0.79	0.76	0.77	288935
time cost				443 seconds

5. Summary

A summary of all our models is presented on Table XI. The Normal Autoencoder and GAN models demonstrated superior performance, achieving the highest recall and accuracy. Regarding the LSTM model, it performed moderately well on detecting malicious data with high precision and accuracy. However, it also had the highest time complexity out of all the models tested. Variational Autoencoders had by far the lowest accuracy and recall.

The primary reason for VAEs suboptimal performance lies in the limitations of using a simple loss function like the reconstruction loss in the context of anomaly detection. While the VAE model is capable of learning meaningful representations of data through its probabilistic framework, this does not inherently translate into robust anomaly detection when applied to noisy or less structured data. The reconstruction loss, although effective in some contexts, struggles to capture the subtle differences between normal data and anomalies, especially when the noise is random and does not follow the same distribution as the data.
In the future, it would be valuable to explore more sophisticated loss functions or regularization techniques, such as adversarial training or incorporating domain-specific knowledge to enhance the sensitivity of the autoencoder model to anomalies, drawing insights from sources like [11].

Overall, based on the results, Normal Autoencoders and GANs appeared to be the most effective models for identifying malicious data in our dataset.

Table XI: **Experiment Results**

Method	Accuracy	Time Cost
Autoencoder	0.86	183 seconds
Variational Autoencoder	0.27	203 seconds
GAN	0.86	207 seconds
LSTM	0.76	443 seconds

E. Comparison of Results

As discussed in Section *A.*, our preprocessing approach was designed to ensure consistency and reproducibility for the analysis of the dataset. This was crucial to allow for a fair and direct comparison of our results with those presented in the referenced paper.

Based on the results from the paper by [7] and our results as shown in Table XII, it is evident that our models, particularly the Autoencoder and GAN significantly outperform the models in the paper by [7]. Both our Autoencoder and GAN models achieved an accuracy of **0.86**, which is substantially higher than any of the models evaluated in the referenced paper. The highest accuracy reported in their study was the Decision Tree model with an accuracy of **0.73**, which is still noticeably lower than our results. In contrast, the Variational Autoencoder (VAE) in our experiments had an accuracy of **0.27**, which is lower than any of the models in the paper by [7]. Lastly, our LSTM model also outperformed any of their models with an accuracy of **0.76**, despite being slightly behind Autoencoder and GAN in terms of accuracy.

However, there are notable differences in the time complexity of these models which are not shown in the table. First of all, their Decision Tree model by [7] had a remarkably low processing time of just 3 seconds whereas our deep learning models required significantly more computation time.
This discrepancy arises from differences in hardware and computing environments. Our models were trained on Google Colab, relying on cloud-based resources, which may have influenced training times based on our choice of runtime type. Additionally, the differences in hardware, optimizations, and processing capabilities could also have had an impact on the efficiency.

Table XII: **Comparison of Results with Paper [7]**

Results from paper [7]	
Method	**Accuracy**
Naive Bayes	0.30
SVM	0.69
Decision Tree	0.73
CNN	0.69
Our Results	
Method	**Accuracy**
Autoencoder	0.86
Variational Autoencoder	0.27
GAN	0.86
LSTM	0.76

Overall, our results demonstrate that deep learning-based models such as Autoencoders, LSTM and GANs are more effective for anomaly detection in this context when compared to the traditional machine learning methods like Naive Bayes, SVM, Decision Trees, as presented in [7].

VI. Conclusion

Our research sought to investigate the effectiveness of deep learning techniques for anomaly detection, by comparing our results with other existing research papers with a focus on detecting malicious traffic in IoT networks. Using the IoT-23 dataset, our experiments revealed that Autoencoders and GANs outperformed traditional machine learning methods, achieving the highest accuracy and recall rates with the least time cost. Specifically, Autoencoders achieved an accuracy of 86.33% with a time cost of 183 seconds, while GANs achieved an accuracy of of 86.36% with a time cost of 207 seconds. In contrast, the Variational Autoencoder performed poorly with an accuracy of 27.38%, while the LSTM model, although moderately accurate, was slower with a time cost of 443 seconds. Collectively, these findings suggest that deep learning models offer significant advantages in anomaly detection for IoT security.

VII. Future Work & Limitations

In the future, we aim to explore advanced neural network techniques. Initially, we plan to experiment with different loss functions, such as adversarial training or the integration of domain-specific knowledge in Variational Autoencoders to assess their impact on accuracy and recall. Exploring advanced variations of VAEs, including conditional VAEs or hybrid models, may further improve anomaly detection capabilities. Additionally, we intend to develop novel autoencoder architectures for anomaly detection, such as Convolutional Autoencoders.
Considering our hardware, a CPU-based runtime was used to execute the experiments. Given the limitations of CPU computation, it lacks the parallel processing advantages of other alternatives, such as GPUs and TPUs, which can significantly accelerate training times. This limitation becomes especially impactful when working with deep learning models, as it affects the efficiency of training, especially for large-scale datasets which could be one of the factors for our longer training times. Furthermore, Our study was limited by time constraints, restricting our literature review to just three sources. Future research should expand the review with more references to provide a more comprehensive comparison and deeper insights into existing approaches.

VIII. Usage of AI-based Tools

In the development of our study, AI-based tools like ChatGPT 3.5 were incorporated and mainly utilized to refine the structure of this paper, enhance the readability, and improve the overall clarity of our research. These tools assisted us by organizing complex concepts into clear explanations, ensuring that technical concepts were effectively communicated. However, while it provided valuable refinements, all technical content, analyses, and conclusions were independently developed and reviewed to ensure accuracy and relevance. Overall, the integration contributed to a more polished presentation of our findings.

References

[1] Asif Ahmed Neloy and Turgeon, M. (2024). A comprehensive study of auto-encoders for anomaly detection: Efficiency and trade-offs. Machine Learning with Applications, pp.100572–100572. doi: https://doi.org/10.1016/j.mlwa.2024.100572.

[2] Sebastian Garcia, Agustin Parmisano, & Maria Jose Erquiaga. (2020). IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]. Zenodo. http://doi.org/10.5281/zenodo.4743746

[3] Vailshery, L. (2024). IoT Connected Devices Worldwide 2019-2030. [online] Statista. Available at: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[4] Bahizad, S. (2020). Risks of Increase in the IoT Devices. 2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom). doi: https://doi.org/10.1109/cscloud-edgecom49738.2020.00038.

[5] Chatterjee, A. and Ahmed, B.S. (2022). IoT anomaly detection methods and applications: A survey. Internet of Things, [online] 19, p.100568. doi: https://doi.org/10.1016/j.iot.2022.100568.

[6] Al-Garadi, M.A., Mohamed, A., Al-Ali, A.K., Du, X., Ali, I. and Guizani, M. (2020). A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security. IEEE Communications Surveys & Tutorials, 22(3), pp.1646–1685. doi: https://doi.org/10.1109/comst.2020.2988293.

[7] Liang, Y., Vankayalapati, N. (2021). Machine learning and deep learning methods for better anomaly detection in IOT-23 Dataset. Department of Computer Science, Lakehead University, ON, Canada. Available at: https://github.com/yliang725/Anomaly-Detection-IoT23/blob/main/Research%20Paper/Research%20Paper.pdf

[8] Dutta, V., Choraś, M., Pawlicki, M. and Kozik, R. (2020). A Deep Learning Ensemble for Network Anomaly and Cyber-Attack Detection. Sensors, 20(16), p.4583. doi: https://doi.org/10.3390/s20164583.

[9] Sahu, A.K., Sharma, S., Tanveer, M. and Raja, R. (2021). Internet of Things attack detection using hybrid Deep Learning Model. Computer Communications, 176, pp.146–154. doi: https://doi.org/10.1016/j.comcom.2021.05.024.

[10] Di Mattia, F., Galeone, P., De Simoni, M. and Ghelfi, E., 2019. A survey on gans for anomaly detection. arXiv preprint https://doi.org/10.48550/arXiv.1906.11632

[11] Arxiv.org. (2021). Variational Autoencoder for Anomaly Detection: A Comparative Study. [online] Available at: https://arxiv.org/html/2408.13561v1