2023

A Secure and Efficient IIoT Anomaly Detection Approach Using a Hybrid Deep Learning Technique

Bharath Reedy Konatham
Wright State University

A SECURE AND EFFICIENT IIOT ANOMALY DETECTION APPROACH USING A HYBRID DEEP LEARNING TECHNIQUE

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

BHARATH REEDY KONATHAM
B.Tech., Gandhi Institute of Technology and Management, India, 2018

2023
Wright State University

WRIGHT STATE UNIVERSITY
College of Graduate Programs and Honors Studies

July 27, 2023

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY `Bharath Reedy Konatham` ENTITLED `A Secure and Efficient IIoT Anomaly Detection Approach Using a Hybrid Deep Learning Technique` BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF `Master of Science`.

Fathi Amsaad, Ph. D.
Thesis Director

Thomas Wischgoll, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

Fathi Amsaad, Ph. D

Lingwei Chen, Ph. D

Raymer, Michael L, Ph. D

Anton Netchaev, Ph. D

Shu Schiller, Ph.D.
Interim Dean, College of Graduate Programs & Honors Studies

Abstract

Konatham, Bharath Reedy. M.S, Department of Computer Science and Engineering, Wright State University, 2023. A secure and Efficient IIoT Anomaly Detection Approach Using a Hybrid Deep Learning Technique.

The Industrial Internet of Things (IIoT) refers to a set of smart devices, i.e., actuators, detectors, smart sensors, and autonomous systems connected throughout the Internet to help achieve the purpose of various industrial applications. Unfortunately, IIoT applications are increasingly integrated into insecure physical environments leading to greater exposure to new cyber and physical system attacks. In the current IIoT security realm, effective anomaly detection is crucial for ensuring the integrity and reliability of critical infrastructure. Traditional security solutions may not apply to IIoT due to new dimensions, including extreme energy constraints in IIoT devices.

Deep learning (DL) techniques like Convolutional Neural Networks (CNN), Gated Recurrent Units (GRU), and Long Short-Term Memory (LSTM) have been the focus of recent research to increase the precision and effectiveness of anomaly identification. This Thesis initially investigates a unique hybrid DL-enabled approach that provide the needed security analysis before successful attacks are launched against IIoT infrastructure. For that, different hybrid models are developed, trained, tested, and validated using Convolutional Neural Networks (CNN), Gated Recurrent Units (GRU), Short-Term Memory (LSTM), Autoencoder, and XGBoost algorithms.

Experimental results show that the proposed XGBoost ML model exhibits the highest performance, as compared to other models, and excels across multiple metrics, including recall, precision, F1-score, and false alarm rate (FAR). The results also show that hybrid CNN+GRU model is closely behind, which exhibited strong performance in accurately detecting anomalies in encrypted IoT traffic. Specifically, Our experimental results show that the developed hybrid CNN+GRU model outperforms the others, achieving an accuracy of 94.94%, a recall of 92.29%, a precision of 98.49%, an F1 score of 95.24%, and a low false alarm rate of 0.001. However, it is essential to note that the hybrid model requires a longer convergence time, indicating a trade-off between performance and computational efficiency. Notably, individual CNN and GRU models also showcase strong performance as time-efficient alternatives.

In conclusion, our adopted comprehensive dataset and rigorous evaluation proves that we have developed practical deep-learning approaches to obtain an accurate measure for an efficient IIoT anomaly detection framework. Finally, as a future work, this study highlights the significance of selecting the appropriate model for anomaly detection in IIoT systems. For that, XGBoost and CNN+GRU showcase of future research and the potential for achieving high accuracy and effectively identifying anomalies. The codes and used developed data sets of this research are attached in the appendix section at the end of this thesis to guide future works in developing advanced hybrid architectures and optimizing computational efficiency to enhance the security of IIoT systems further.

Contents

List of Figures

List of Tables

Acknowledgment

I would like to express my deepest gratitude to the individuals who have contributed to my knowledge and supported me throughout this thesis.

Firstly, I would like to thank the Department of Computer Science and Engineering at Wright State University for providing a stimulating academic environment and fostering a culture of innovation and collaboration. The opportunity to work alongside talented peers and dedicated faculty members has been invaluable to my growth as a researcher.

I am especially grateful to my thesis advisor, Dr. Fathi Amsaad, for his exceptional guidance, expertise, and unwavering support. His mentorship and encouragement have played a crucial role in shaping the direction of my research and motivating me to strive for excellence. His passion for the subject matter and his dedication to my academic development have been truly inspiring.

I would like to extend my heartfelt appreciation to my mother, Roja Konatham, for her unconditional love, unwavering belief in my abilities, and constant support throughout my academic journey. Her sacrifices, encouragement, and wise counsel have been the foundation of my success, and I am deeply grateful for her presence in my life.

I would also like to acknowledge the invaluable support and friendship of my best friend, Simra Tabassum. Her unwavering belief in my abilities, encouragement during challenging times, and shared moments of laughter and joy have been a constant

source of strength and inspiration. I am grateful for her genuine friendship and the positive impact she has had on my personal and academic life.

Lastly, I would like to express my appreciation to all my friends, colleagues, and loved ones who have supported me along this journey. Their words of encouragement, understanding, and belief in my potential have been instrumental in keeping me motivated and focused.

I am indebted to all these individuals for their contributions, guidance, and unwavering support. This thesis would not have been possible without their belief in me and their commitment to my success

1 Introduction

1.1 Research Significance

The rapid growth of Internet of Things (IoT) technology has paved the way for the integration of smart devices in various domains, including industrial environments. The Industrial Internet of Things (IIoT) has revolutionized industries by enabling real-time monitoring, predictive maintenance, and data-driven decision-making. With the proliferation of IIoT devices, a massive amount of data is generated, providing valuable insights for optimizing industrial processes and improving productivity.

However, along with the benefits of IIoT, the increasing interconnectivity also introduces new challenges, particularly in terms of security and anomaly detection. Anomaly detection plays a crucial role in identifying abnormal behavior or events in IIoT systems, enabling timely response and mitigation of potential risks. Traditional anomaly detection approaches are often limited in their ability to handle the complexity and scale of IIoT data.

1.2 Anomaly

Finding the cases that stand out as being different from all the rest is a typical requirement when studying real-world data sets. Such occurrences are referred to as anomalies, and the objective of anomaly detection (also known as outlier identification) is to identify all such occurrences in a data-driven manner [6].

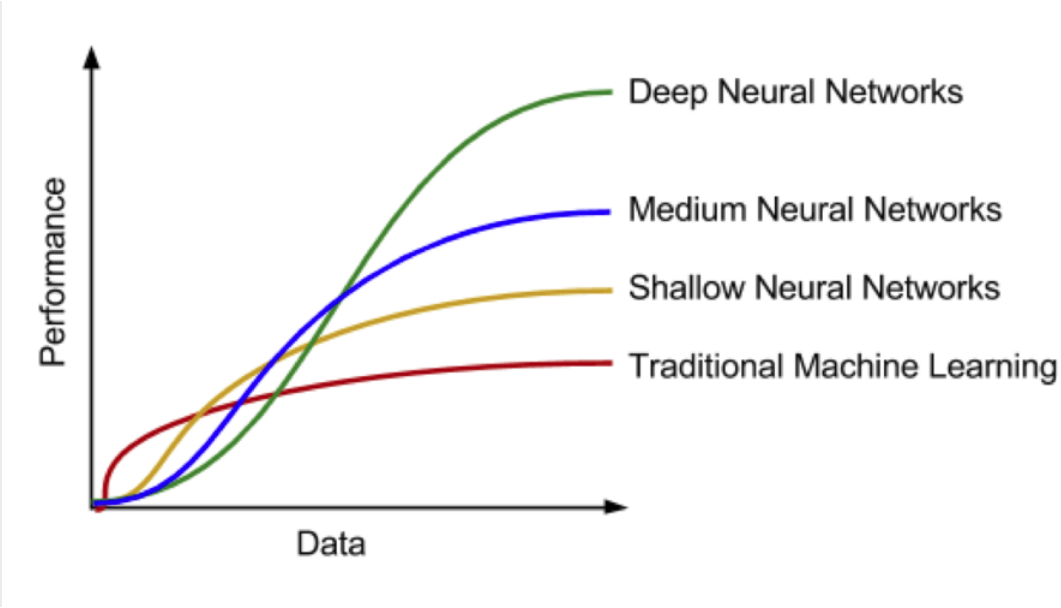


Figure 1.1: Performance Comparison of Deep Learning vs Traditional Algorithms [1]

An outlier is defined as an observation that deviates so significantly from other observations as to raise suspicion that it was generated by a different mechanism [7]. Outliers can be caused by errors in the data, but sometimes they are indicative of a new, previously unknown, underlying process.

Deep neural networks have become increasingly common in the larger area of machine learning in recent years. From figure 1.1 we cobserver that Deep learning algorithms have demonstrated comparable performance to other machine learning techniques.

Deep learning, a subset of machine learning, learns to represent the input as a deep hierarchy of concepts within layers of the neural network. This allows it to perform well and be flexible. As the size of the data grows, deep learning beats regular machine learning, as seen in Figure 1.3 and Figure 1.2 represents how deep learning-based anomaly detection algorithms have been used for a variety of tasks in recent years. Research demonstrates that deep learning significantly outperforms conventional techniques. [2, 8]. Over the years, researchers conduct several research

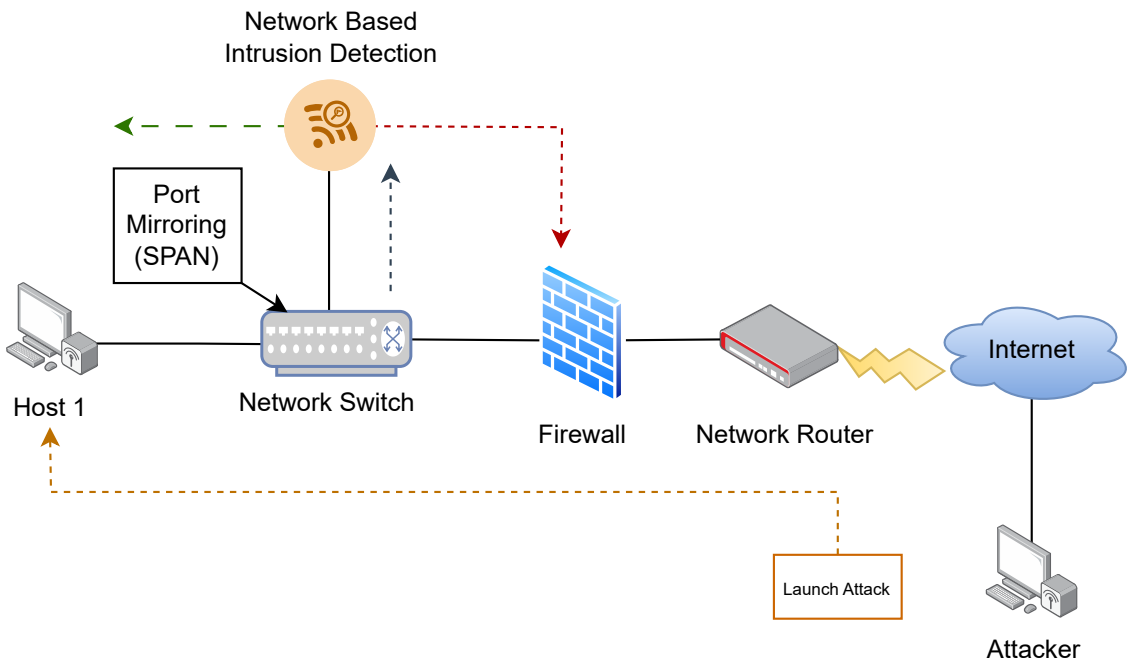


Figure 1.2: Cyber Network Intrusion Detection System [2]

works to address this challenge and develop effective anomaly detection models.

1.3 Anomaly Detection in IIoT

In the context of edge IoT, an anomaly refers to any unexpected or abnormal behavior or pattern observed in the data [9] collected from edge devices. Figure 1.4 represents a general anomaly detection scenario in IIoT. Anomalies can be deviations from the normal expected behavior, outliers, or unusual patterns that may indicate a malfunction, security breach, or other irregularities in the edge IoT system.

Detecting anomalies in edge IoT data is crucial for maintaining the reliability, security, and optimal performance of the system. By monitoring and analyzing the data collected from edge devices, anomalies can be identified and flagged for further investigation or action. This can involve comparing the incoming data to predefined thresholds, statistical models, machine learning algorithms, or domain-specific rules to identify deviations from the expected behavior.

(a) Illegal Traffic Flow detection

(b) Detecting Retinal Damage

Figure 1.3: (a) Video Surveillance, Image Analysis: Illegal Traffic detection [3], (b) Health-care: Detecting Retinal Damage [4]

Anomaly detection in edge IoT plays a significant role in various applications, such as predictive maintenance, security monitoring, fault detection, and quality control. By detecting anomalies in real-time or near real-time, appropriate actions can be taken to mitigate risks, optimize operations, and ensure the overall integrity and efficiency of the edge IoT system.

1.4 Challenges in IIoT Anomaly Detection

The development of anomaly detection schemes in the IIoT environment is challenging due to several factors such as (1) scarcity of IoT resources; (2) profiling normal behaviors; (3) the dimensionality of data; (4) context information; and (5) the lack of resilient machine learning models [10]. These factors will be explained in this section [11].

1.4.1 Scarcity of IIoT Resources

The leverage of device-level IIoT anomaly detection can be hindered by the constraints in storage, processing, communication, and power resources. To compensate for this, the cloud can be adopted as a data collection, storage, and processing platform. However, the remoteness of the cloud can introduce high latency due to resource scheduling and round trip time. This delay may not be acceptable for real-time requirements of IoT suspicious events [10]. It is also evident that the

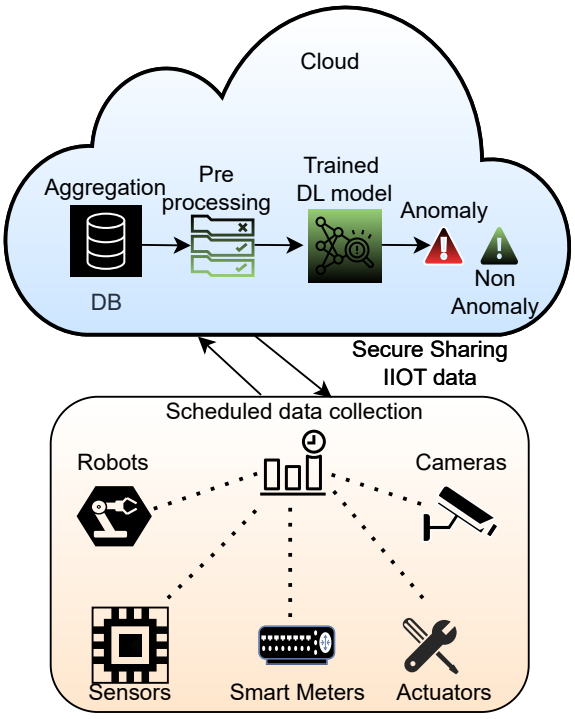


Figure 1.4: Anomaly Detection in IIoT Application

scale of traffic in the IIoT may degrade the detection performance of the anomaly detection system if it exceeds the capacity of the devices. A better solution is to offload certain storage and computations from devices to edge nodes or to send aggregated data to the cloud. Sliding window techniques can also offer reduced storage benefits by withholding only certain data points, though the anomaly detection system may require patterns/trends [12].

1.4.2 Profiling Normal Behaviours

The success of an anomaly detection system depends on gathering sufficient data about normal behaviors; however, defining normal activities is challenging. Due to their rare occurrence, anomalous behaviors might be collected within normal behaviors. There is a lack of datasets representing both IIoT normal and abnormal data, making supervised learning impractical, specifically for massively deployed IoT devices. This drives the need to model IIoT anomaly detection systems in

unsupervised or semi-supervised schemes, where data deviating from those collected in normal operations are taken as anomalous [13].

1.4.3 Dimensionality of Data

IIoT data can be univariate as key-value x_t, or multivariate as temporally correlated univariate $x_t = [x_{t1}, \ldots, x_{tn}]$. The IoT anomaly detection using univariate series compares current data against historical time series. In contrast, multivariate-based detection provides historical stream relationships and relationships among attributes at a given time. Thus, choosing a specific anomaly detection mechanism in IoT applications depends on data dimensionality due to associated overheads in processing [13, 14]. Furthermore, multivariate data introduces the complexity of processing for models, which needs dimension reduction techniques using principal components analysis (PCA) and autoencoders (AE). On the other hand, univariate data may not represent finding patterns and correlations that enhance machine learning performance.

1.4.4 Context Information

The distributed nature of IoT devices caters to context information for anomaly detection. However, the challenge is to capture the temporal input at a time t_1 is related to input at a time t_n and spatial contexts in large IoT deployments where some IIoT devices are mobile in their operations. This means that introducing context enriches anomaly detection systems but increases complexity if the right context is not captured [13].

1.4.5 Lack of Machine Learning Models Resiliency against Adversarial Attacks

The lack of a low false-positive rate of existing machine learning models and the vulnerability to adversarial attacks during training and detection call for both accurate algorithms and resilient models.

On the other hand, the massive deployment of IoT devices could be leveraged for collective anomaly detection as most of the devices in the network exhibit similar characteristics. This large number of devices helps to utilize the power of cooperation against cyber-attacks such as malware [15]. Model poisoning and evasion can decrease the utility of machine learning models as adversaries can introduce fake data to train or tamper with the model.The upcoming sections will outline the research objectives, methodology, and contributions in detail.

1.5 Objectives of the Research

The objectives of this research are to:

1. Investigate the effectiveness of various machine learning models for anomaly detection in IIoT systems.
2. Assess the performance of Convolutional Neural Networks (CNN), Gated Recurrent Units (GRU), GRU+CNN hybrid, Long Short-Term Memory (LSTM), Autoencoder + GRU, Autoencoder + CNN, Autoencoder + LSTM, and XGBoost models.
3. Compare the accuracy, recall, precision, F1-score, and false alarm rate (FAR) of these models.
4. Explore the potential for improving the security and reliability of IIoT systems

through effective anomaly detection.

1.6 Research Methodology

This research holds significant importance in the field of IIoT security and anomaly detection. By evaluating and comparing multiple machine learning models, this study aims to identify the most effective approaches for detecting anomalies in IIoT systems. The findings of this research can contribute to enhancing the security posture of critical infrastructure, reducing the impact of potential attacks, and ensuring the smooth operation of IIoT networks. Additionally, the insights 4gained from this research can guide the development of advanced hybrid architectures and optimization techniques for future anomaly detection systems. The research methodology employed in this study includes the following steps:

- **Data collection:** A representative dataset of encrypted IIOT traffic, including both normal and anomalous instances, was gathered.
- **Preprocessing:** Conducted necessary preprocessing tasks such as feature selection, dimensionality reduction, and data normalization.
- **Model selection and implementation:** Evaluated and implemented various machine learning models, including CNN, GRU, hybrid models, LSTM, Autoencoder-based models, and XGBoost.
- **Performance evaluation:** Assessed the performance of each model using metrics such as accuracy, recall, precision, F1-score, and false alarm rate (FAR).
- **Comparative analysis:** Conducted a comprehensive analysis to compare the performance of the different models and identify the top-performing approaches.

1.7 Contribution of this Thesis

The contributions of this research can be summarized as follows:

- A Hybrid model for anomaly detection is proposed to secure IIoT network traffic by leveraging multiple deep learning architectures such as CNN, GRU, LSTM, Autoencoders and a gradient boosting algorithm XGBoost.
- Significant improvement in anomaly detection accuracy is achieved by utilizing multiple deep learning algorithms. These algorithms effectively extract intricate features and patterns from encrypted traffic, resulting in a substantial improvement compared to existing methods.
- The current research also addresses class imbalance by employing oversampling techniques, resulting in improved performance for detecting anomalies across different attack types.
- The proposed hybrid model exhibits enhanced performance by combining CNN and GRU for classifying encrypted attack traffic, leveraging the strengths of both architectures to capture spatial and temporal characteristics.

In the next chapter, we will discuss the related work in the field of anomaly detection for IIoT and explore the application of various deep learning models in enhancing IIoT security. We will examine different research works that have leveraged deep learning techniques to tackle the challenge of anomaly detection in IIoT systems. Additionally, we will provide a overview of models and datasets utilized in other studies to gain insights into their effectiveness and applicability in the context of IIoT security.

2 Related Work

Anomaly detection in edge IIoT systems has attracted significant attention from researchers, given the growing importance of ensuring the reliability and security of these systems. Various machine learning and deep learning techniques have been explored for this purpose. In this section, we review the literature on anomaly detection in edge IIoT systems, focusing on the models evaluated in this paper: CNN, GRU, CNN+GRU, LSTM, Autoencoder+CNN, Autoencoder+GRU, Autoencoder+LSTM, XGBoost.

Convolutional Neural Networks (CNN) have been widely applied for anomaly detection in IIoT systems due to their ability to capture spatial features in data. [16] presents a data-driven fault diagnosis method based on Convolutional Neural Networks (CNNs). The authors showcase that CNNs are effective in capturing local and global patterns in time series data, which can be useful for detecting anomalies in different applications.

A Long Short-Term Memory (LSTM) network is introduced to address the vanishing gradient problem that plagued traditional Recurrent Neural Networks (RNNs) during the training process [17]. In their experiments, LSTM networks outperformed other recurrent network algorithms, such as real-time recurrent learning, backpropagation through time, and recurrent cascade correlation, in learning complex, artificial long-time-lag tasks. [18] presents a Long Short-Term Memory (LSTM) based Encoder-Decoder architecture for anomaly detection in time series data. Although the authors focus on LSTM, the approach could be applied to GRUs as well. They

demonstrate that their model is capable of detecting anomalies in multivariate time series data with various types of anomalies. [19] proposed an anomaly detection technique for Electrocardiogram (ECG) time signals using deep Long Short-Term Memory (LSTM) networks. Their work demonstrates the capability of LSTM networks in identifying anomalies in time series data, which can be adapted for different domains and datasets.

A robust deep autoencoder approach for anomaly detection is proposed by [20]. They introduce a method that incorporates robust principal component analysis (RPCA) into the training of deep autoencoders, which makes the model more resistant to outliers and improves its performance in detecting anomalies. This paper introduces a method for multivariate anomaly detection using Generative Adversarial Networks (GANs) and GRUs. [21] proposed the MAD-GAN framework that combines the strengths of GANs and GRUs to learn the underlying structure of the time series data and detect anomalies effectively.

[22] propose an unsupervised method for anomaly detection and diagnosis in multivariate time series data using deep neural networks, including CNNs and autoencoders. They demonstrate the effectiveness of their approach on various real-world datasets, showcasing the potential of combining CNNs and autoencoders for anomaly detection tasks. An XGBoost-based network intrusion detection system for imbalanced data is proposed in [23]. The authors implement a method that combines XGBoost with a weighted loss function to handle imbalanced data effectively. Although their focus is on network intrusion detection, the approach could be adapted for other anomaly detection tasks. Lastly, [24]provides a comprehensive survey of various network anomaly detection techniques, including machine learning-based methods. The authors discuss and compare different machine learning algorithms and their performance in the context of anomaly detection.

Anomaly detection in encrypted Internet traffic is a significant area of research, given the increasing reliance on encrypted services for consumer privacy. A recent study that aligns closely with our work proposed the use of hybrid deep learning techniques for detecting anomalies in encrypted network traffic [25].

The study utilized multiple deep learning models, including Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Recurrent Neural Networks (RNNs), across three public datasets: NSL-KDD, UNSW-NB15, and CIC-IDS-2017. Their findings highlighted a hybrid model combining CNN and Gated Recurrent Unit (GRU) as the most effective solution. While the approach was valid, the use of older datasets may not reflect the intricacies of contemporary cyber threats. In contrast, our work leverages a more recent dataset, providing a more accurate and practical solution for detecting current cybersecurity threats. In the next section we discuss about the previous works on applications of various deep learning models on IIoT security.

2.1 Deep Learning in IIoT security

Deep learning has emerged as a prominent area of research in IoT systems [26, 27, 28, 5]. Its effectiveness in handling large datasets is a major advantage over classical machine learning methods. Given the vast amounts of data generated by IoT systems, deep learning techniques are well-suited for analysis. These approaches dynamically generate representations of dynamic data [29] and can be seamlessly integrated with the IoT ecosystem [30]. For instance, in a smart home, IoT devices can autonomously communicate with each other, enabling a fully intelligent home [26].

Deep learning employs a computational paradigm with multiple layers that learn different levels of abstraction in data structures. This enables significant advancements over traditional machine learning techniques [31, 32]. Deep learning is a subfield of

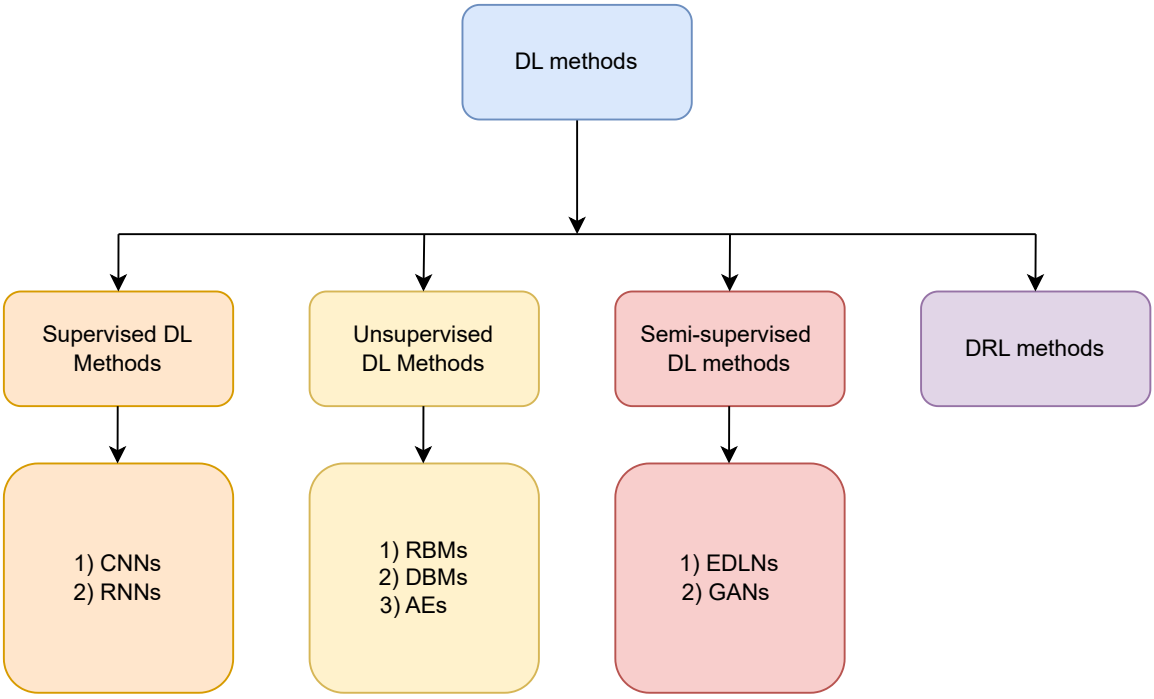


Figure 2.1: Classification of DL Algorithms [5]

machine learning that utilizes nonlinear layers of computation to extract discriminative or generative patterns. With their ability to capture hierarchical features, deep learning methods are often referred to as hierarchical learning approaches. They draw inspiration from how human neurons and the brain interpret stimuli.

Deep networks encompass both unsupervised and supervised learning, as well as hybrid learning approaches that combine both forms. In this section, we discuss the commonly used deep learning algorithms. Figure 2.1 illustrates various deep learning classifiers for IoT security.

2.1.1 Supervised Deep Learning

The most popular controlled DL techniques are covered in this section. There are two different categories of discriminative DL algorithms: recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

A. Convolutional Neural Networks (CNNs):

Convolutional Neural Networks (CNNs) are widely used in deep learning for their ability to reduce the number of data parameters in artificial neural networks (ANNs) [33, 34]. CNNs employ sparse relationships, parameter sharing, and fair distribution to minimize data parameters. They consist of convolutional layers that apply filters (kernels) to combine data parameters and pooling layers that reduce the size of subsequent layers through average or peak pooling [35, 36, 37].

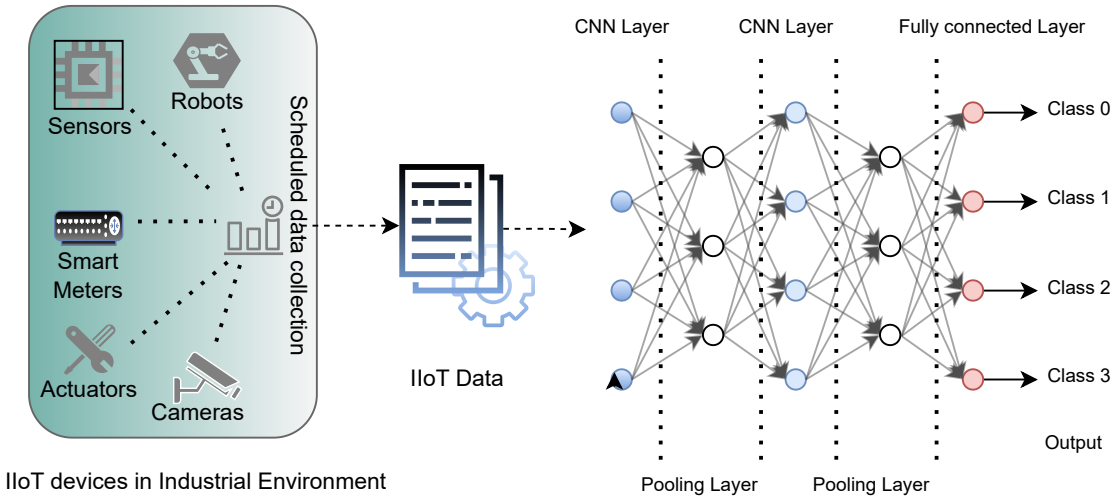


Figure 2.2: Feeding IIoT data through CNN layers

Figure 2.2 depicts how CNN works when IoT protection is extended. The activation device, such as ReLU, introduces non-linear activation to the feature space [38].

While CNNs excel in image detection and classification CNN development is mostly focused on image detection. Due to their widespread application, CNNs are utilized to develop precise and reliable models of picture ID and categorization for big public image datasets like ImageNet [39, 40], however, their high computational cost poses challenges for resource-constrained IIoT devices. Distributed architectures and lightweight deep neural networks (DNNs) have been proposed to address this issue

[41]. CNNs have also found applications in malware identification, such as developing a CNN-based framework for Android IoT protection [42]. However, it's important to consider that CNNs' powerful learning capacity can also be exploited by attackers, as demonstrated in breaking cryptographic applications [43].

In summary, CNNs offer effective simultaneous learning and classification, eliminating the need for separate feature extraction steps [42]. However, their resource requirements and potential vulnerabilities must be carefully considered in IoT security applications.

B. Recurrent Neural Networks (RNNs):

One essential class of deep learning methods is recurrent neural networks (RNNs). RNNs are particularly useful for handling transient outcomes and making predictions based on both recent and previous inputs. They are commonly used in applications that require consecutive inputs, such as sensor, text, and speech data. RNNs have a sequential data storage layer and hidden components that are regularly updated to reflect the current state of the network. However, RNNs suffer from the issue of gradients fading or exploding, which limits their effectiveness in certain scenarios [44].

RNNs are widely used in various applications, including speech recognition, translation, and network security. In speech-related tasks, RNNs and their variants have shown superior performance [45, 46, 47]. For network security, RNNs can be employed to detect potential network attacks by analyzing large volumes of sequential data, such as network traffic patterns. Previous research has demonstrated the effectiveness of RNN analysis in accurately classifying network traffic and detecting malicious behavior [48]. RNNs offer practical solutions for enhancing the security of IoT systems, particularly in mitigating serial-based attacks.

In practical applications, RNNs play a crucial role in safeguarding IoT devices and networks. IoT networks face the challenge of detecting a wide range of potential network attacks, requiring the analysis of significant amounts of sequential data from various sources. RNNs enable accurate detection of malicious activity by leveraging their ability to recognize patterns in sequential data. By studying RNNs and their variations, it becomes possible to enhance the overall security of IoT systems, addressing the evolving threats and ensuring robust protection against serial-based attacks. against serial-based assaults.

2.1.2 Unsupervised Deep Learning

In this part, we go through the most popular unsupervised DL methods, such as deep restricted Boltzmann machines (RBMs), deep belief networks (DBNs), and autoencoders (AEs).

A. Restricted Boltzmann Machines (RBMs):

Uncontrolled Restricted Boltzmann Machines (RBMs) are deep generative models that operate without any directional constraints among nodes, allowing for flexible information flow [49]. RBMs consist of visible and hidden layers, with the visible layer receiving input data and the hidden layer representing latent variables at different levels. RBMs capture data features hierarchically, with the hidden layer serving as a record of features from the previous layer [50]. An RBM-based model was proposed for network anomaly detection, addressing challenges such as the lack of labeled data for effective testing and the evolving nature of irregular behavior over time [50]. The researchers employed a discriminative RBM model, which combines generative and classification capabilities to detect network anomalies even with incomplete training data. However, their experiments revealed reduced classification accuracy when

applied to different network datasets. Further research is needed to extend the classifier's performance across various network contexts. To enhance the capabilities of a single RBM, multiple RBMs can be stacked to form a Deep Belief Network (DBN), which will be discussed in detail in the following section.

B. Deep Belief Networks (DBNs):

Generative approaches in Deep Belief Networks (DBNs) have been explored [51]. DBNs consist of stacked, unsupervised Restricted Boltzmann Machines (RBMs) trained layer by layer [52]. The initial layer-specific pretraining phase utilizes a greedy unsupervised learning technique to learn the initial features, while the top layer is fine-tuned using a softmax layer [48]. DBNs have been effectively applied to malware detection, surpassing traditional feature engineering methods [53]. Recent studies have combined Evolutionary Algorithms (EA) with DBNs for malware detection, employing non-linear projection techniques to reduce data dimensionality [54]. However, the resource-intensive nature of DBNs makes them less suitable for resource-constrained devices.

C. Deep Autoencoders (AEs):

Deep Autoencoders (AEs) are unsupervised learning neural networks designed to replicate their input to output [33]. They consist of an encoder function and a decoder function, with the encoder transforming the input into a code and the decoder reconstructing the original input [33]. AEs aim to minimize the reconstruction error during the learning phase [55, 56]. However, AEs can only provide an approximate reproduction and prioritize which input characteristics to copy [33]. While AEs are effective for feature extraction, they require significant computing time and may

struggle if the training data does not match the test data [57]. AEs have been used for ransomware recognition, outperforming traditional machine learning algorithms in terms of detection efficiency [57]. In another study, a Deep Belief Network (DBN) was combined with AEs for malware detection and dimensionality reduction [54]. The DBN learned to identify malicious code based on the learned latent representations [54].

2.1.3 Semi-Supervised or Hybrid DL

The most traditional deep hybrid learning strategies are covered in this section. Generative adversarial networks (GANs) and network communities (EDLNs) are examples of hybrid deep learning techniques.

A. Generative Adversarial Networks (GANs):

Generative Adversarial Networks (GANs) are a powerful framework for deep learning [58]. From figure 2.3 we can understand that GANs consist of two models, a generative model, and a discriminatory model, trained simultaneously using an opposed mechanism [58]. The generative model learns the data distribution and aims to fool the discriminator, while the discriminator predicts the probability of evaluation results [58, 59]. GANs have been applied in IoT security, demonstrating efficacy in detecting suspicious system activity [60]. GANs are capable of learning various attack scenarios and generating samples that resemble zero-day attacks [60]. They are well-suited for semi-supervised classification and can produce samples faster than fully transparent Deep Belief Networks (DBNs) [58, 61]. However, GAN training can be challenging, and GANs may struggle to generate diverse data such as text [58, 61].

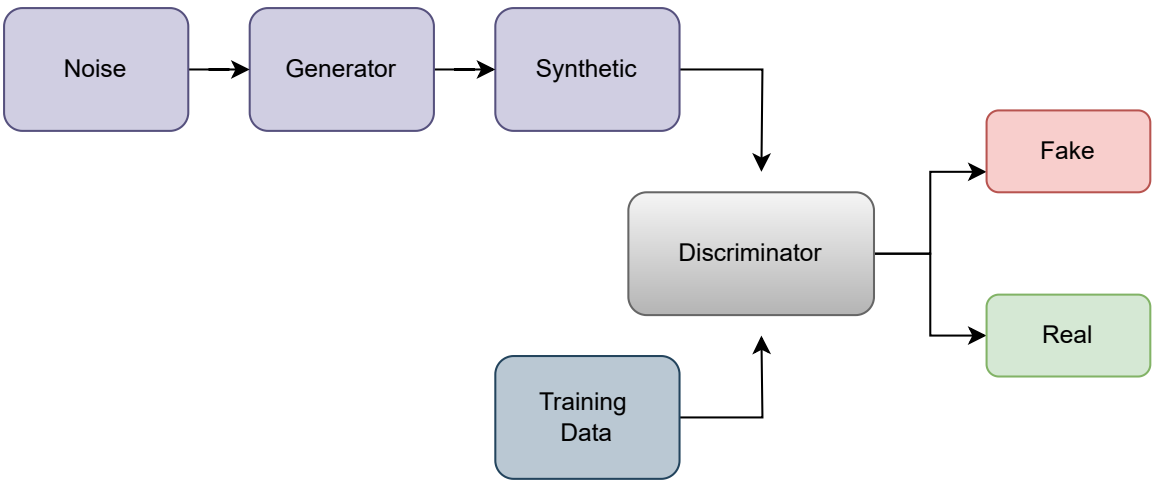


Figure 2.3: Functional Architecture of Generative Adversarial Networks (GANs)[5]

B. Ensemble of DL Networks (EDLNs):

Hybrid EDLNs, combining generative and discriminatory models, can be created using ensemble learning (EL) techniques [32, 62, 63, 64]. EL employs multiple classification approaches to improve performance and is commonly used in anomaly detection, virus detection, and intrusion detection [65, 66, 67]. However, EDLNs may increase temporal complexity in the system. Assembling DNNs as explicit/implicit ensembles offers an alternative to mitigate the computing cost associated with training numerous neural networks [68, 69]. Dropout and DropConnect introduce sparsity and prevent overfitting, while stochastic depth and Swapout enhance network depth during training [70, 71, 72]. EDLNs, consisting of heterogeneous or homogeneous classifiers, are effective in addressing complex problems and have shown promising results in various applications [73, 74]. Further research is needed to deploy lightweight classifiers in distributed IoT settings to enhance accuracy and efficiency for IoT security systems.

2.1.4 Deep Reinforcement Learning (DRL)

Reinforcement learning (RL) is an ML technique that improves a learning agent's methods by evaluating and iterating on actions to achieve long-term goals [75]. RL allows agents to learn from trial and error, calculating rewards and transitioning to the next state [76]. RL is particularly useful for complex problems with long-term outcomes, assuming Markovian models where actions and rewards are based on the environment's dynamics [75].

Deep reinforcement learning (DRL) combines RL with deep learning, such as the deep Q-network (DQN), to solve high-dimensional problems with scalability and performance [77]. However, the curse of dimensionality and computational requirements limit its applicability in real-world systems. To address these limitations, various enhancements have been proposed, including dual Q-learning, continuous monitoring, and priority replay of knowledge [78, 79, 80]. DRL is often used in conjunction with other ML approaches to overcome its drawbacks and achieve better results.

In the context of IoT networks, researchers have explored the integration of DRL with access management, download optimization, and cyber security mechanisms [77, 80]. These investigations aim to leverage DRL's capabilities for improved application schemes and protection against cyber threats. Further research in applying DRL within the IoT context can contribute to advancements in IoT security and optimization.

Table 2.1 provides an overview of various works related to our research, highlighting their models and datasets.

Table 2.1: Related Works in Anomaly Detection for IIoT

Work	**Models Covered**	**Dataset**
A deep hybrid learning model for detection of cyber attacks in industrial IoT devices	Attention-based Long Short Term Memory (ALSTM), Fully Convolutional Neural Network (FCN) + XGBoost , Fully Convolutional Neural Network (FCN)+ AdaBoost	TON_IoT
IoT Anomaly Detection Using a Multitude of Machine Learning Algorithms	Extreme Gradient Boosting (XGBoost), Support Vector Machines (SVM) , Deep Convolutional Neural Networks (DCNN)	IoT-23
A Comparative Analysis of Machine Learning Techniques for IoT Intrusion Detection	Support Vector Machine (SVM), XGBoost, Light Gradient Boosting Machine (LightGBM), Isolation Forest (iForest), Local Outlier Factor (LOF), Q-Network (DDQIN) based Deep Reinforcement Learning (DRL)	IoT-23
Anomaly Detection in Encrypted Internet Traffic Using Hybrid Deep Learning	Convolutional Neural Network (CNN), Long Short-term Memory (LSTM), Gated Recurrent Unit (GRU) Convolutional Neural Network (CNN) +Gated Recurrent Unit (GRU)	NSL-KDD UNSW-NB15 CIC-IDS-2017
An improved anomaly detection model for IoT security using decision tree and gradient boosting	Gradient boosting (GB) and decision tree (DT) through the open-source Catboost	NSL-KDD IoT-23 BoT-IoT Edge-IIoT
Secure and Efficient Anomaly Detection Using Hybrid Deep Learning Approach (Current Work)	Convolutional Neural Network (CNN) Long Short-term Memory (LSTM) Gated Recurrent Unit (GRU) Convolutional Neural Network (CNN) +Gated Recurrent Unit (GRU) Autoencoder + Convolutional Neural Network (CNN) Autoencoder + Long Short-term Memory (LSTM) Autoencoder + Gated Recurrent Unit (GRU) Extreme Gradient Boosting (XGBoost)	Edge-IIoTset

3 Theoretical Foundations

3.1 Threat Model

In this section, we present the threat model for our anomaly detection system. We outline various potential attacks that the system may encounter and analyze their impact on the system and overall performance. We also introduce the list of various Deep learning models implemented in this study.

3.1.1 Scope of the Threat Model

The scope of this threat model encompasses the detection and mitigation of various types of network attacks in the context of anomaly detection using machine learning algorithms. The focus is on analyzing encrypted IoT network traffic and identifying patterns indicative of malicious activities. The threat model considers attacks such as DDoS (UDP, ICMP, TCP, and HTTP), SQL injection, vulnerability scanning, password attacks, uploading of malicious files, backdoors, port scanning, XSS, ransomware, fingerprinting, and MITM attacks. The goal is to develop effective anomaly detection mechanisms that can detect and classify these attacks accurately, thereby enhancing the security of IoT systems and networks. The threat model takes into account the specific characteristics of encrypted traffic and aims to address the evolving challenges posed by sophisticated attack techniques.

3.1.2 Attack Types

- **Normal:** Represents normal or legitimate network traffic without any malicious intent. This category serves as the baseline for comparison against other attack types.
- **DDoS UDP (Distributed Denial of Service User Datagram Protocol):** Involves overwhelming a target system with a large volume of UDP packets, causing it to become unresponsive or unavailable to legitimate users.
- **DDoS ICMP (Distributed Denial of Service Internet Control Message Protocol):** Aims to disrupt network connectivity by flooding the target system with ICMP packets, resulting in network congestion or resource exhaustion.
- **SQL Injection:** Exploits vulnerabilities in web applications that interact with databases. Attackers inject malicious SQL statements through user input fields to manipulate or extract sensitive data from the database.
- **DDoS TCP (Distributed Denial of Service Transmission Control Protocol):** Targets the TCP protocol to exhaust system resources or disrupt network communication by flooding the target system with an overwhelming number of TCP connection requests.
- **Vulnerability Scanner:** Automated tools or scripts used to identify potential weaknesses or vulnerabilities in a target system or network. Scans for misconfigurations, outdated software, or known security flaws that can be exploited by attackers.
- **Password:** Involves various techniques such as brute-forcing, dictionary attacks, or phishing to gain unauthorized access by compromising user passwords.

- **DDoS HTTP (Distributed Denial of Service - Hypertext Transfer Protocol):** Focuses on overwhelming web servers or applications by flooding them with a high volume of HTTP requests, causing service disruptions or resource depletion.

- **Uploading:** Refers to unauthorized file uploads or malicious file uploads to a web server or application. Attackers exploit vulnerabilities to upload and execute malicious scripts or malware.

- **Backdoor:** Involves unauthorized access or the creation of a hidden entry point in a system, allowing attackers to gain persistent access and control over the compromised system.

- **Port Scanning:** The process of systematically scanning a target system or network to identify open ports and services. Attackers use port scanning to discover potential entry points and vulnerabilities for further exploitation.

- **XSS (Cross-Site Scripting):** Exploits vulnerabilities in web applications to inject malicious scripts into web pages viewed by other users. Attackers can steal sensitive information, manipulate website content, or perform phishing attacks through XSS vulnerabilities.

- **Ransomware:** Malware that encrypts files on a victim's system, rendering them inaccessible until a ransom is paid. Attackers demand payment in exchange for providing decryption keys or restoring access to the encrypted data.

- **Fingerprinting:** Involves gathering information about a target system or network to identify its characteristics, vulnerabilities, or potential attack vectors. Attackers perform fingerprinting to gain insights and tailor their attacks accordingly.

- **MITM (Man-in-the-Middle):** Attackers intercept and manipulate communication between two parties without their knowledge. Allows attackers to eavesdrop, modify, or inject malicious content into the communication stream.

3.1.3 Analysis of Existing Security Systems:

In the field of anomaly detection for network attacks, several existing security systems have been developed to mitigate risks and protect network infrastructure. These systems utilize a combination of traditional rule-based methods, statistical analysis, and machine-learning techniques. Here is a brief analysis of some prominent security systems:

- **Intrusion Detection Systems (IDS):** IDSs monitor network traffic and identify suspicious activities based on predefined rules or signatures. They can detect known attack patterns but may struggle with detecting novel attacks or zero-day exploits.
- **Intrusion Prevention Systems (IPS):** IPSs build upon IDSs by not only detecting but also actively blocking or mitigating malicious activities. They often use rule-based approaches to identify and respond to attacks in real time.
- **Firewall Systems:** Firewalls act as a barrier between internal and external networks, enforcing access control policies. While they are effective at filtering traffic based on predetermined rules, they may not be equipped to detect or prevent sophisticated attacks.
- **Machine Learning-Based Anomaly Detection Systems:** These systems leverage machine learning algorithms to learn patterns of normal network behavior and detect deviations that may indicate attacks. They can identify previously unseen attacks and adapt to changing attack strategies.

- **Deep Learning-Based Anomaly Detection Systems:** Deep learning approaches, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and autoencoders, have shown promise in detecting anomalies in network traffic. They can capture complex patterns and dependencies in data, enabling effective anomaly detection.

While existing security systems provide valuable defense mechanisms, they often face challenges in detecting evolving and sophisticated attacks. Traditional rule-based systems may struggle with zero-day attacks, while machine learning-based systems require large amounts of labeled training data and may be prone to false positives.

To address these limitations, this research aims to develop an effective anomaly detection system by incorporating deep learning algorithms (CNN, GRU, LSTM, autoencoders) and XGBoost, a gradient boosting algorithm, this research aims to develop an advanced anomaly detection system that can effectively detect and classify network attacks. The combination of these algorithms offers the merits of accurate pattern recognition, adaptability to changing attack strategies, and improved overall system performance. Through this research, we aim to enhance the security of IIoT systems and networks by leveraging the strengths of deep learning and gradient-boosting algorithms.

3.1.4 Assumptions

In the context of our anomaly detection research using various machine learning algorithms, we make the following assumptions:

- **Stationarity of Data:** We assume that the underlying data distribution remains stationary over time, meaning that the statistical properties of normal and abnormal behavior remain relatively constant during the period of analysis.

- **Representative Training Data:** We assume that the training data used to train the machine learning algorithms is representative of the real-world data distribution. The training data encompasses a diverse range of normal and abnormal instances to enable the algorithms to learn meaningful patterns and generalize well to unseen data.

- **Independence of Instances:** We assume that instances in the dataset are independent and identically distributed (i.i.d.). This assumption simplifies the modeling and analysis process and allows for the application of standard machine-learning algorithms. Deviations from independence may arise in scenarios where instances exhibit temporal dependencies or when the presence of one anomaly influences the likelihood of subsequent anomalies.

- **Feature Relevance:** We assume that the features used in the anomaly detection process are relevant and informative for distinguishing between normal and abnormal instances. Feature engineering and selection techniques are employed to identify the most relevant features that capture the discriminatory power necessary for accurate anomaly detection.

- **Generalization to Unseen Data:** We assume that the trained models will generalize well to unseen data, including both normal and abnormal instances. The models capture the underlying patterns and characteristics of the data distribution, allowing them to accurately detect anomalies in new instances.

These assumptions form the basis of our research and guide the design and evaluation of our anomaly detection framework.

3.1.5 Classification models

In this work, we implement individual and combination of these models to evaluate their performance on the edge IIoT dataset. Below are the models we used:

1. Convolutional Neural Networks (CNN): Convolutional Neural Networks (CNNs) have shown remarkable success in various computer vision tasks, including image classification and object detection. In our work, we employed CNNs for anomaly detection in Industrial Internet of Things (IIoT) systems. The main advantage of CNNs lies in their ability to automatically learn hierarchical representations from raw data. The network consists of multiple convolutional layers that capture local patterns and features, followed by pooling layers that reduce spatial dimensions. By stacking these layers, CNN can learn increasingly complex representations and identify anomalies in the data.

2. Gated Recurrent Units (GRU): Gated Recurrent Units (GRUs) are a type of recurrent neural network (RNN) that excels in capturing sequential dependencies in data. In our work, we leveraged GRUs for anomaly detection in IIoT systems, as they are well-suited for processing time-series data. GRUs utilize gating mechanisms to control the flow of information within the network, allowing them to retain important information over longer sequences. This enables the model to effectively capture temporal patterns and identify anomalies in the data.

3. Long Short-Term Memory (LSTM): LSTM is another type of recurrent neural network that addresses the vanishing gradient problem faced by traditional RNNs. LSTMs are capable of learning long-term dependencies in sequential data and have been widely used in various applications, including natural language processing and time-series analysis. In our work, we applied LSTM models for anomaly detection in IIoT systems. The LSTM's memory cell and gating mechanism enable it to

retain and update information over extended sequences, making it a powerful tool for capturing complex temporal patterns and detecting anomalies.

4. CNN+GRU Hybrid Model: To further enhance the anomaly detection performance, we explored a hybrid model that combines the strengths of both CNN and GRU. The CNN+GRU model takes advantage of the CNN's ability to extract local spatial features and the GRU's proficiency in capturing sequential dependencies. By combining these two architectures, the hybrid model achieves a comprehensive understanding of the data, enabling it to effectively identify anomalies in IIoT systems.

5. Autoencoder-based Models: In addition to the deep learning models, we also employed autoencoder-based models for anomaly detection in IIoT systems. Autoencoders are unsupervised learning models that learn to reconstruct the input data, thereby capturing its underlying structure. We explored different combinations of autoencoders with CNN, GRU, and LSTM architectures. By training the models to reconstruct normal data accurately, we can identify deviations from the learned patterns as anomalies.

6. XGBoost: While deep learning models have shown great promise in anomaly detection, we also considered traditional machine learning approaches. XGBoost, an ensemble learning algorithm based on decision trees, was included in our analysis. XGBoost utilizes gradient boosting to combine the predictions of multiple decision trees, resulting in an accurate and robust model. We explored the effectiveness of XGBoost for anomaly detection in IIoT systems, leveraging its ability to handle both numerical and categorical features. We aimed to comprehensively address the challenges of anomaly detection in IIoT systems by utilizing a combination of models with different strengths in capturing spatial, temporal, and structural patterns in the data.

4 Proposed Technique

Anomaly detection in Industrial Internet of Things (IIoT) systems is a complex and critical task. The interconnected and dynamic nature of IIoT devices and networks makes it challenging to identify abnormal behavior and potential security threats. Traditional anomaly detection methods often struggle to cope with the scale and variability of IIoT data, calling for more advanced approaches.

In our research, we have developed and evaluated several deep learning models for anomaly detection in IIoT systems. Among these models, XGBoost emerges as the top performer, consistently achieving exceptional accuracy, precision, recall, and F1 score. Leveraging the power of gradient boosting, XGBoost effectively captures intricate patterns and dependencies in the data, enabling it to accurately detect anomalies in IIoT systems. The results validate its effectiveness and highlight its potential as a reliable solution for anomaly detection in complex IIoT environments.

Additionally, the CNN+GRU model demonstrates significant promise as the second-best performer in our study. By combining the spatial feature extraction capabilities of Convolutional Neural Networks (CNNs) with the temporal sequence modeling abilities of Gated Recurrent Units (GRUs), the CNN+GRU model captures both local patterns and long-term dependencies in the IIoT data. This hybrid model showcases commendable accuracy, precision, recall, and F1 score, further emphasizing its effectiveness in anomaly detection.

In the following section, we will provide a comprehensive explanation of the techniques employed in our work, focusing on two top-performing models: XGBoost, a gradient boosting algorithm renowned for its scalability and efficiency, and CNN+GRU, a hybrid model that combines the strengths of convolutional neural networks (CNNs) and gated recurrent units (GRUs) to effectively capture spatial and temporal patterns in the data.

4.1 Extreme Gradient Boost(XGBoost):

XGBoost, short for Extreme Gradient Boosting, has demonstrated exceptional performance in various machine learning tasks, including anomaly detection in the context of our work. This boosting algorithm is known for its ability to handle large-scale datasets and complex feature spaces effectively. One of the key factors contributing to the success of XGBoost is its ensemble learning approach, which combines the predictions of multiple weak learners (decision trees) to form a robust and accurate final model.

One notable aspect of XGBoost is its optimization objective, which incorporates both the loss function and a regularization term to prevent overfitting. By employing a gradient-based approach, XGBoost optimizes the model's parameters in an iterative manner, continually improving its predictive power. Furthermore, XGBoost employs a unique tree construction algorithm that intelligently selects the optimal splitting points, resulting in highly informative and expressive decision trees.

In terms of technical capabilities, XGBoost provides a range of features that enhance its performance and versatility. These include built-in handling of missing values, support for parallel computing, and the ability to handle sparse data efficiently. XGBoost also offers flexible parameters for fine-tuning the model, allowing users to control factors such as learning rate, maximum tree depth, and regularization

strength. Its ability to handle imbalanced datasets is particularly valuable in anomaly detection, as it helps mitigate the challenges posed by rare and infrequent events. The below pseudo code gives an overview of the XGBoost classifier we employed which shows its simplicity. Below pseudo code provides the simple implementation of XGBoost algorithm

Psudo code:

Algorithm 1 XGBoost Model Training

```
Import the necessary libraries (XGBoost)
Initialize the XGBoost classifier model
Train the model using the training data
Specify the evaluation set for monitoring performance
Set early stopping rounds to prevent overfitting
Obtain the training history and evaluation results
```

4.2 CNN+GRU Hybrid model:

The CNN+GRU model combines the strengths of Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs) to achieve high performance in anomaly detection. This architecture effectively captures both spatial and temporal information present in the data, making it well-suited for analyzing complex sequences such as those encountered in industrial IoT systems.

The CNN component of the model uses convolutional layers to extract spatial features from the input data. The convolutional layers apply filters to capture patterns and structures in the data, allowing the model to learn meaningful representations. By stacking multiple convolutional layers with increasing filter sizes and pooling layers, the CNN model can effectively capture hierarchical representations of the input.

On the other hand, the GRU component focuses on modeling the temporal dependencies in the data. GRUs are a type of recurrent neural network (RNN)

that have gating mechanisms to selectively update and reset their internal states. This enables the model to retain and propagate important information across time steps, capturing long-term dependencies in the sequence. The GRU layer in the model utilizes these gating mechanisms to learn and represent temporal patterns in the data.

The combination of the CNN and GRU models through the concatenation of their output layers allows for the fusion of both spatial and temporal features. This fusion provides a comprehensive understanding of the data, enabling the model to make accurate predictions. By leveraging the complementary strengths of CNNs and GRUs, the CNN+GRU architecture achieves a balance between capturing local spatial features and modeling temporal dynamics.

The advantages of the CNN+GRU model lie in its ability to effectively capture complex patterns in industrial IoT data. The CNN component excels at extracting spatial features, detecting local anomalies, and recognizing spatial patterns. The GRU component, on the other hand, captures temporal dependencies, enabling the model to understand the temporal behavior of the data and detect anomalies that occur over time. By combining these two architectures, the CNN+GRU model can effectively identify anomalies that exhibit both spatial and temporal characteristics, providing a comprehensive solution for anomaly detection in industrial IoT systems.

Through extensive experimentation and evaluation, the CNN+GRU model has demonstrated its effectiveness in our work. It has achieved high accuracy, precision, recall, and F1 score in detecting anomalies in the industrial IoT dataset. The model's ability to capture intricate spatial and temporal patterns allows it to accurately identify anomalous instances, enabling proactive security measures in industrial IoT systems. Below pseudo code and figure 4.1 represent the architecture of the Hybrid model.

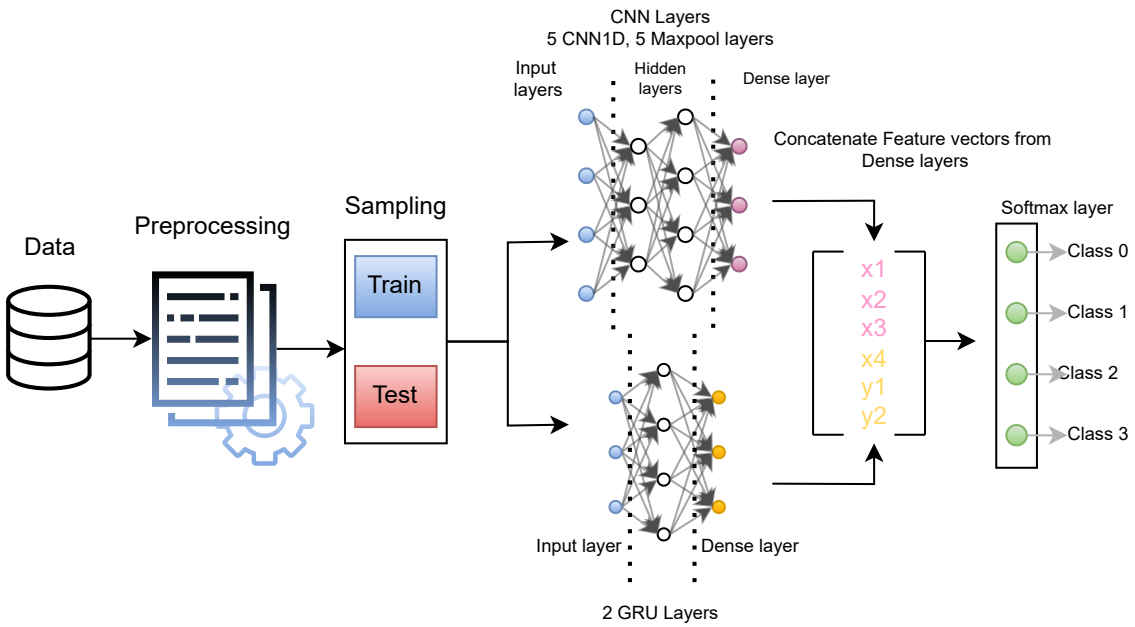


Figure 4.1: Hybrid CNN GRU model

Psudo code:

Algorithm 2 CNN+GRU Model

Define the CNN model
$cnn\_input \leftarrow Input(shape = input\_shape)$
$cnn\_layer \leftarrow Conv1D(64, 3, activation =' relu')(cnn\_input)$
$cnn\_layer \leftarrow MaxPooling1D(2)(cnn\_layer)$ ▷ CNN Layers
Define the GRU model
$gru\_input \leftarrow Input(shape = input\_shape)$
$gru\_layer \leftarrow GRU(units = 32, activation =' tanh')(gru\_input)$ ▷ GRU layers
Concatenate the outputs from the CNN and GRU models
$concat\_layer \leftarrow concatenate([cnn\_layer, gru\_layer])$
Output layer for classifying anomalies
$output\_layer \leftarrow Dense(num\_classes, activation =' softmax')(concat\_layer)$
Create the CNN+GRU model ▷ CNN+GRU Model
$model \leftarrow Model(inputs = [cnn\_input, gru\_input], outputs = output\_layer)$

5 Experimental Details

Below, we present the experimental details and in the next we present chapter results of our work, providing a comprehensive analysis of the performance and effectiveness of various models employed in the experimental process. These models, including CNN, GRU, LSTM, CNN+GRU hybrid model, XGBoost and Autoencoder based models, were carefully selected for their ability to effectively extract features and patterns from encrypted IIoT network traffic. Each model brings unique strengths and capabilities to the table, allowing for a comprehensive approach to anomaly detection. The hardware setup will now be presented, outlining the necessary components and configurations to support the experimental implementation.

5.1 Hardware setup

The proposed models including CNN, GRU, LSTM, hybrid CNN+GRU, and Autoencoder-based models, were executed on a system with 24GB of RAM and an NVIDIA RTX 3060 Ti graphics card. The system is powered by a Ryzen 5600G processor, which provides efficient computational capabilities for deep learning tasks. The GPU acceleration offered by the RTX 3060 Ti enables faster model training and evaluation, enhancing the overall efficiency of the experiments.

Additionally, the XGBoost model, which is a gradient-boosting algorithm, was run on the CPU. While neural network models benefit from GPU acceleration, XGBoost is primarily designed to utilize the computational power of the CPU. The CPU in the system provides sufficient processing capacity for running XGBoost and

leveraging its capabilities for anomaly detection in edge IIoT systems.

By utilizing this hardware setup, the experiments were conducted with the necessary computational resources to ensure accurate evaluations of the models' performance in anomaly detection. The combination of the powerful GPU and CPU allows for efficient training, evaluation, and comparison of the neural network models and XGBoost, enabling robust insights and findings in the thesis.

5.2 Dataset

In this study, we utilized a comprehensive dataset [81] specifically designed for anomaly detection in Industrial Internet of Things (IIoT) networks. The dataset comprises a diverse range of network traffic data, including normal traffic and various types of attack scenarios such as Port Scanning, XSS, Ransomware, Fingerprinting, and MITM. To ensure the representativeness and reliability of the data, it was collected from a real-world industrial setting, consisting of multiple devices and communication protocols commonly found in IIoT networks. By leveraging this rich dataset, we were able to effectively assess the performance of different machine learning and deep learning models in detecting and mitigating cybersecurity threats in the IIoT domain.

The dataset comprises 2,219,201 instances and 63 features, which have been collected to study and analyze cybersecurity threats in the context of edge computing for Industrial Internet of Things (IIoT) systems. The dataset includes information such as network traffic attributes, protocol-specific parameters, and attack types. The features in the dataset encompass various data types, including numerical (float64) and categorical (object). Some of the key features include network communication parameters like IP addresses (ip.src_host, ip.dst_host), ARP protocol details (arp.opcode, arp.hw.size), ICMP protocol attributes (icmp.checksum, icmp.seq_le),

HTTP protocol fields (http.content_length, http.request.method, http.referer), and TCP/UDP protocol properties (tcp.flags, tcp.len, udp.port, udp.stream). The dataset also contains features related to domain name system (DNS) queries (dns.qry.name, dns.qry.type) and Message Queuing Telemetry Transport (MQTT) protocol (mqtt.conack.flags, mqtt.hdrflags, mqtt.topic).

Data Preprocessing

In the process of preparing the dataset for analysis, it was necessary to perform various preprocessing steps to ensure the quality and relevance of the data. One crucial step involved the removal of certain columns that were deemed either redundant or irrelevant to the specific analysis goals. The decision to exclude these columns was based on their limited contribution to the analysis or the availability of more informative features. The following columns were removed from the dataset: `frame.time`, `ip.src_host`, `ip.dst_host`, `arp.src.proto_ipv4`, `arp.dst.proto_ipv4`, `http.file_data`, `http.request.full_uri`, `icmp.transmit_timestamp`, `http.request.uri.query`, `tcp.options`, `tcp.payload`, `tcp.srcport`, `tcp.dstport`, `udp.port`, and `mqtt.msg`.

The rationale behind the removal of these columns was as follows:

1. `frame.time`: This column contained the timestamp of each network frame. However, for the purpose of our analysis, the timestamp was not deemed essential to the specific research questions and thus was excluded.

2. `ip.src_host` and `ip.dst_host`: These columns represented the source and destination IP addresses of network packets. Since our analysis focused on higher-level protocols and features derived from packet content, the inclusion

of IP addresses was considered unnecessary and did not contribute significantly to the research objectives.

3. `arp.src.proto_ipv4` and `arp.dst.proto_ipv4`: These columns contained the source and destination IPv4 addresses in Address Resolution Protocol (ARP) packets. Given that our analysis did not involve ARP-specific features, these columns were removed to streamline the dataset and eliminate irrelevant information.

4. `http.file_data` and `http.request.full_uri`: These columns included file data and the full URIs (Uniform Resource Identifiers) extracted from HTTP requests. Since our analysis did not require deep inspection of HTTP payloads or specific file-related features, these columns were excluded to reduce the dimensionality of the data and focus on more relevant attributes.

5. `icmp.transmit_timestamp`: This column represented the transmit timestamp in ICMP (Internet Control Message Protocol) packets. Since our analysis did not specifically focus on ICMP-related features, this column was removed as it did not contribute significantly to the research goals.

6. `http.request.uri.query`: This column contained the query string part of the URI in HTTP requests. Since our analysis did not involve query-specific features, such as parameter extraction or query pattern analysis, this column was excluded to simplify the dataset representation and avoid unnecessary complexity.

7. `tcp.options`, `tcp.payload`, `tcp.srcport`, `tcp.dstport`, and `udp.port`: These columns contained TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) packet-specific attributes. Since our analysis did not require detailed inspection of these protocol-specific features, these columns were

removed to focus on more informative attributes and reduce the dimensionality of the dataset.

8. `mqtt.msg`: This column represented the MQTT (Message Queuing Telemetry Transport) message. As our analysis did not specifically revolve around MQTT-related

Oversampling data:

The target variable of the dataset is the "Attack_type", which is a categorical variable representing 15 different classes of cybersecurity threats, such as Distributed Denial of Service (DDoS), ransomware, man-in-the-middle (MITM) attacks, and port scanning, among others. The dataset is further processed by dropping unnecessary columns, handling missing and duplicate values, and shuffling the data. Categorical variables are one-hot encoded, and the target variable is label-encoded. To address class imbalance, RandomOverSampler is used for oversampling minority classes. The dataset is then split into train and test sets. Finally, feature scaling is applied using MinMaxScaler, and input data and target variables are reshaped to fit the requirements of deep learning models. To address the class imbalance issue in the dataset, random oversampling techniques were employed. The objective was to increase the number of samples for the minority classes using the `RandomOverSampler` from the `imblearn` library. Figure 5.1 shows the number of samples for each class after the data preprocessing step.

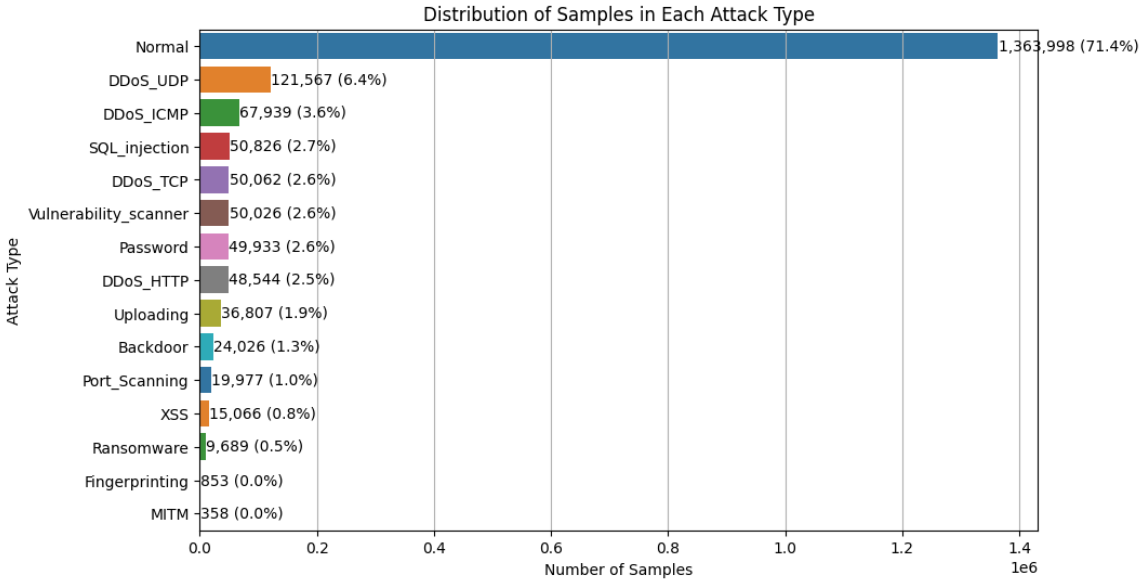


Figure 5.1: Distribution of samples after preprocessing dataset

The random oversampling technique involves generating synthetic instances for the minority classes to match the number of samples in the majority class. By creating additional instances, the dataset becomes more balanced, allowing the machine learning models to learn from a more diverse range of instances. The logic behind random oversampling is to provide the model with a better representation of the minority classes, enabling it to learn and detect anomalies from these underrepresented classes more effectively. By introducing synthetic instances, the model can capture the patterns and characteristics specific to the minority classes, improving its overall performance and accuracy. It is important to note that oversampling techniques, including random oversampling, should be carefully evaluated to avoid potential issues such as overfitting or introducing biases. Additionally, alternative approaches may need to be considered based on the specific characteristics of the dataset and the research objectives.

The application of random oversampling in this study involved utilizing the `RandomOverSampler` from the `imblearn` library. The minority classes were

oversampled to match the number of samples in the majority class, resulting in a more balanced dataset. This oversampled dataset served as a valuable resource for training and evaluating the machine learning models, ultimately enhancing their ability to detect anomalies.

5.3 Architecture of the models

5.3.1 CNN model overview:

The proposed 1D-CNN model in this paper is built using the Keras Sequential API. It consists of multiple layers designed to extract useful features from the input data and make predictions. The model includes an input layer that takes data in the shape corresponding to the dimensions of the dataset. Five Conv1D layers are added after it, each with a different number of filters and a ReLU activation function. These layers extract local features from the data using filters. After each convolutional layer, a MaxPooling1D layer with a pool size of 2 is applied to downsample the data and capture important features, followed by a flatten layer that converts the feature maps into a one-dimensional vector. The model also includes two dense layers. The first dense layer with 64 neurons and a ReLU activation function learn global features, while the second dense layer with the same number of neurons as the number of classes in the dataset uses a softmax activation function to provide probabilities for each class, enabling the final prediction.

5.3.2 GRU model

The proposed model is a Gated Recurrent Unit (recurrentGRUunit)-based neural network implemented using the gatedKeras Sequential API. It consists of two GRU layers, with the first layer having 32 units and the second layer having 64 units. The GRU layers use a combination of tanh and sigmoid activation functions for

the update and reset gates. The model also includes two dense layers with ReLU activation functions, with 32 and 16 units, respectively. The final output layer has a softmax activation function with a number of units corresponding to the number of classes in the dataset. The GRU-based model effectively captures both short-term and long-term dependencies in sequential data, making it suitable for various classification tasks.

5.3.3 LSTM model overview

The proposed model is an LSTM classifier designed for time series classification tasks. The model consists of a series of LSTM layers followed by dense layers for classification. The model's architecture is as follows:

1. *LSTM Layer 1*: The first LSTM layer forms the input layer of the model. It requires the input shape to be specified and comprises 128 units. The 'tanh' activation function is used for the transformations within the LSTM units. This layer is configured to return sequences, meaning it outputs a sequence of the same length for the next layer to use, instead of just the output of the last timestep.

2. *LSTM Layer 2*: The second LSTM layer consists of 256 units and uses a 'tanh' activation function. Unlike the previous layer, it doesn't return sequences, hence it only outputs the last output of the LSTM sequence. This allows us to connect this layer to a traditional dense layer.

3. *Dense Layer (Output Layer)*: The final layer in the model is a dense layer with a number of units equal to the number of classes in the classification task (num_classes). This layer uses a softmax activation function to produce a probability distribution over the classes, making it suitable for multi-class classification tasks.

This model leverages the ability of LSTM layers to process and learn from sequence data, creating an effective and efficient model for time-series classification tasks.

5.3.4 CNN+GRU model overview:

The proposed model is a hybrid CNN-GRU neural network that combines the strengths of convolutional neural networks (CNNs) and gated recurrent units (GRUs) to process and learn from sequential data efficiently. The model is designed using the Keras Functional API, and it consists of two separate branches: the CNN branch and the GRU branch. The model's architecture is as follows:

CNN Branch:

1. *Input Layer:* The model takes as input a shape corresponding to the dimensions of the dataset.

2. *Convolutional Layers:* The CNN branch includes two convolutional layers with 64 and 128 filters, respectively, each with a kernel size of 3. Both layers use the ReLU activation function.

3. *MaxPooling Layers:* In between the convolutional layers, max-pooling layers with a pool size of 2 are used to reduce the spatial dimensions and improve computational efficiency.

4. *Flatten Layer:* After the final max-pooling layer, a flatten layer is used to convert the 3D output of the convolutional layers into a 1D vector.

5. *Dense Layer:* The last layer in the CNN branch is a dense layer with 64 units and a ReLU activation function. This layer helps the model learn higher-level features from the extracted spatial information.

GRU Branch:

1. *Input Layer:* Similar to the CNN branch, the input layer of the GRU branch accepts an input of the same shape.

2. *GRU Layer:* The GRU layer has 32 units and uses a 'tanh' activation function for the gate updates and a 'sigmoid' activation function for the reset gates. Recurrent dropout is disabled (set to 0), and the layer does not unroll the recurrent loop for efficiency. Bias terms are included in the computations of the update and reset gates, and the hidden states are reset after each sequence.

3. *Dense Layer:* Following the GRU layer, a dense layer with 32 units and a 'tanh' activation function is added. This layer helps the model learn more complex patterns and features from the extracted temporal information.

Combining the branches: After processing the input data through both the CNN and GRU branches, the outputs from these branches are concatenated using the concatenate layer. This combined representation incorporates the spatial and temporal features learned from both branches for the final decision-making process.

Output Layer: The final layer is a dense layer with (num_classes) units and a softmax activation function. The softmax function provides a probability distribution across the classes, enabling the model to make a final prediction based on the highest probability. The structure of the combined CNN+GRU model is depicted in Figure1.

5.3.5 XGBoost classifier overview

The XGBoost classifier model is trained using the XGBoost library, which provides an efficient implementation of the gradient boosting algorithm. The XGBoost algorithm constructs an ensemble of decision tree models iteratively to optimize a specific

objective function.

Model Initialization

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier()
```

The XGBoost classifier model is initialized by creating an instance of the `XGBClassifier` class from the `XGBoost` library.

Model Training

```
xgb_model.fit(X_train, y_train,
              eval_set=[(X_train, y_train), (X_val, y_val)],
              early_stopping_rounds=5)
```

The model is trained using the `fit()` method, which takes the following arguments:

- `X_train`: The feature matrix of the training data.
- `y_train`: The target labels corresponding to the training data.
- `eval_set`: A list of evaluation datasets, including both the training set and the validation set.
- `early_stopping_rounds`: An optional parameter that specifies the number of consecutive rounds with no improvement on the validation set's performance before stopping the training process.

During the training process, the XGBoost classifier iteratively builds an ensemble of decision tree models, optimizing a specific objective function to minimize the loss. The model learns to make predictions based on the input features and their corresponding labels.

Evaluation and Early Stopping

The `eval_set` parameter allows the model to monitor its performance on both the training set and the validation set during training. The `early_stopping_rounds` parameter helps prevent overfitting by stopping the training if the performance on the validation set does not improve for a certain number of rounds.

5.3.6 Autoencoder Models Overview

The proposed models are a combination of an autoencoder with CNN, LSTM, and GRU networks that leverages the benefits of both techniques to process and learn from input data efficiently. Each of these models is designed using the Keras Functional API and consists of two main parts: the encoder-decoder (autoencoder) component and the CNN, LSTM and GRU components. The type of autoencoder used in the above models is a basic convolutional autoencoder. A convolutional autoencoder utilizes convolutional layers for both encoding and decoding. These

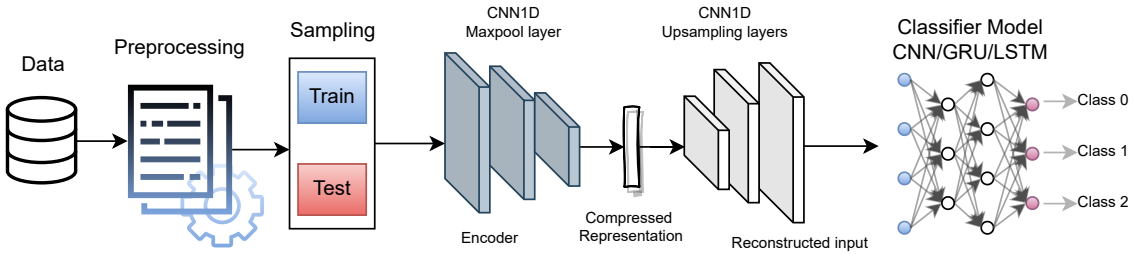


Figure 5.2: Autoencoder-based Models Architecture

layers are effective in capturing spatial patterns and features in the input data. The

encoder part of the autoencoder uses convolutional layers with decreasing filters to extract the essential features from the input data and reduce its dimensionality. The decoder part uses upsampling layers followed by convolutional layers to reconstruct the original input from the encoded representation. Fig 4.2 shows the overview of the autoencoder models, the pseudo code also gives a better understanding of the model.

Model 1:

Encoder:

1. *Input Layer:* The model takes as input a shape corresponding to the dimensions of the dataset.
2. *Convolutional Layers:* The encoder includes three convolutional layers with 32, 64, and 128 filters, respectively, each with a kernel size of 3. All layers use the ReLU activation function.
3. *MaxPooling Layers:* In between the convolutional layers, max-pooling layers with a pool size of 2 are used to reduce the spatial dimensions and improve computational efficiency.
4. *Output Layer:* The encoder's output is obtained from the last max-pooling layer.

Decoder:

1. *Convolutional Layers:* The decoder consists of three convolutional layers with 128, 64, and 32 filters, respectively, each with a kernel size of 3. All layers use the ReLU activation function.
2. *UpSampling Layers:* After each convolutional layer, upsampling layers with a size of 2 are used to restore the spatial dimensions.

3. *Output Layer:* The decoder's output is obtained from the last convolutional layer.

Autoencoder: The autoencoder is formed by combining the encoder and decoder models. The input to the autoencoder is the same as the input to the encoder, and the output is obtained from the decoder.

Classifier:

1. The autoencoder is added as the first layer of the classifier model.
2. Additional Conv1D and MaxPooling1D layers are used for feature extraction and dimensionality reduction.
3. The output from the last MaxPooling1D layer is flattened.
4. Dense layers with ReLU activation are added for further processing of the features.
5. The final Dense layer with softmax activation provides the probability distribution across the classes for classification.

Model 2:

The architecture is similar to Model 1, with the following modifications:

Classifier:

1. Instead of Conv1D and MaxPooling1D layers, LSTM layers are used for feature extraction and sequence processing.
2. The LSTM layers return sequences and extract features from the sequences.

3. A Dense layer with softmax activation is added for classification.

Model 3:

The architecture is similar to Model 1, with the following modifications:

Classifier:

1. Instead of Conv1D and MaxPooling1D layers, a GRU (Gated Recurrent Unit) layer is used for sequence processing.
2. The GRU layer has 32 units and uses the 'tanh' activation function for the gate updates and 'sigmoid' activation function for the reset gates.
3. A Dense layer with 'tanh' activation is added for further processing of the features.
4. The final Dense layer with softmax activation provides the probability distribution across the classes for classification.

Pseudo code:

Algorithm 3 Autoencoder + GRU Model

Define the Autoencoder model
$encoder\_inputs \leftarrow Input(shape = input\_shape)$
$encoder\_outputs \leftarrow ...$ ▷ Encoder architecture
Define the Decoder model
$decoder\_inputs \leftarrow Input(shape = ...)$
$decoder\_outputs \leftarrow ...$ ▷ Decoder architecture
Combine the Encoder and Decoder models to form the Autoencoder
$autoencoder \leftarrow Model(inputs = encoder\_inputs, outputs = decoder\_outputs)$
Define the GRU model
$gru\_inputs \leftarrow Input(shape = input\_shape)$
$gru\_outputs \leftarrow GRU(units = ...,')(gru\_inputs)$
Concatenate the outputs from the Autoencoder and GRU models
$concat\_outputs \leftarrow concatenate([autoencoder\_outputs, gru\_outputs])$
Output layer for classification
$output\_layer \leftarrow Dense(num\_classes, activation =' softmax')(concat\_outputs)$
Create the Autoencoder + GRU model
$model \leftarrow Model(inputs = [encoder\_inputs, gru\_inputs], outputs = output\_layer)$
Compile the model with appropriate optimizer, loss function and trainig data

5.4 Callback Functions

In machine learning and deep learning, a callback is a function or set of functions that can be applied during the training process of a model. Callbacks provide a way to customize and control the training behavior by performing specific actions at various stages of training, such as at the end of each epoch or when a certain condition is met. They are useful for monitoring the model's performance, adjusting parameters, implementing early stopping, saving model checkpoints, and more.

5.4.1 Callbacks Used

EarlyStopping: The EarlyStopping callback is used to monitor a specified metric, such as the validation loss, and stop the training process if the monitored metric does not improve for a certain number of epochs. In this work, the EarlyStopping callback is set to monitor the validation loss (`val_loss`) and stop

the training if it does not improve for 5 consecutive epochs. This helps prevent overfitting and saves computational resources by stopping the training early when further training is unlikely to improve the model's performance.

ReduceLROnPlateau: The ReduceLROnPlateau callback is used to reduce the learning rate when the monitored metric (e.g., validation loss) stops improving. This helps fine-tune the learning process by dynamically adjusting the learning rate during training. In your case, the learning rate is reduced by a factor of 0.2 whenever the validation loss does not improve for 5 consecutive epochs. This can help the model to converge faster or escape from a local minimum.

6 Results and discussion

This section provides a comprehensive evaluation of the performance of the different deep learning models used for anomaly detection in encrypted IoT traffic. It includes a detailed discussion of the results, highlighting the strengths and weaknesses of each model and comparing their performance metrics such as accuracy, precision, recall, and F1-score. The section also explores the convergence behavior of the models and discusses any notable observations or patterns.

6.1 Metrics

Metrics in machine learning are quantitative measures used to evaluate the performance of a model. These metrics provide insights into how well the model is performing in terms of accuracy, precision, recall, F1-score, and other evaluation criteria. Metrics are essential in assessing the effectiveness and reliability of a model, as they allow comparisons between different models or variations of the same model. The metrics used in this study are explained below.

6.1.1 Categorical Cross-Entropy Loss

Categorical cross-entropy is a commonly used loss function in classification tasks where the output is a categorical variable with multiple classes. It measures the dissimilarity between the predicted class probabilities and the true class probabilities.

Mathematically, the categorical cross-entropy loss is calculated as follows:

Let's assume we have n samples and m classes. The predicted class probabilities for the i-th sample are denoted as p_i and the true class probabilities as t_i.

The categorical cross-entropy loss is given by:

$$L_{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} t_{ij}\log(p_{ij})$$

where t_{ij} is the true probability of the i-th sample belonging to class j and p_{ij} is the predicted probability of the i-th sample belonging to class j.

The inner summation is calculated across all classes for each sample, and the outer summation is calculated across all samples.

This loss function penalizes large differences between the predicted probabilities and the true probabilities, encouraging the model to make more accurate predictions.

In summary, the categorical cross-entropy loss measures the dissimilarity between the predicted class probabilities and the true class probabilities, and it is widely used in multiclass classification tasks.

6.1.2 Accuracy

Accuracy is a commonly used metric to evaluate the performance of classification models. It measures the proportion of correctly predicted samples out of the total number of samples.

Mathematically, accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{Number of correctly predicted samples}}{\text{Total number of samples}}$$

6.1.3 Recall

Recall, also known as the true positive rate or sensitivity, measures the proportion of correctly predicted positive samples (anomalies) out of all actual positive samples.

Mathematically, recall is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

6.1.4 Precision

Precision measures the proportion of correctly predicted positive samples (anomalies) out of all predicted positive samples.

Mathematically, precision is calculated as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

6.1.5 F1-score

The F1-score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance. It combines both precision and recall into a single metric.

Mathematically, the F1-score is calculated as:

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The function calculates the following components:

1. True Positives: The number of correctly predicted positive samples.

2. Possible Positives: The total number of positive samples in the dataset.

3. Predicted Positives: The total number of samples predicted as positive by the model.

The function then calculates the precision and recall as follows:

Precision: The ratio of true positives to the total predicted positives. It measures how many of the predicted positive samples are actually positive.

Recall: The ratio of true positives to the total possible positives. It measures how many of the actual positive samples are correctly identified by the model.

Finally, the F1 score is calculated as the harmonic mean of precision and recall, which provides a balanced measure of both metrics. The `K.epsilon()` term is added to avoid division by zero errors.

The averaging technique applied in this function is the micro-averaging. Micro-averaging aggregates the true positives, possible positives, and predicted positives over all classes in the dataset and then calculates precision, recall, and F1 score based on the aggregated values.

The F1-score ranges between 0 and 1, with 1 indicating a perfect model and 0 indicating poor performance.

6.1.6 False Alarm Rate (FAR)

The false alarm rate is a performance metric that measures the proportion of falsely predicted positive instances (false alarms) out of all negative instances. It is calculated using the following equation:

$$\text{FAR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

where False Positives represent the number of negative instances incorrectly predicted as positive, and True Negatives represent the number of correctly predicted negative instances.

These metrics provide valuable insights into the performance of the anomaly detection models, allowing for a comprehensive evaluation of their effectiveness.

6.2 Model Performance

In this section, we present the performance evaluation of various machine learning models for anomaly detection in encrypted IoT traffic. We assess the effectiveness of deep learning models including CNN, GRU, LSTM, and autoencoder-based models, as well as the XGBoost algorithm. The evaluation is based on above metrics shedding light on the models' capabilities in accurately detecting anomalies and minimizing false positives.

Table 6.1: Performance of the models

Model	Loss	Accuracy	Recall	Precision	F1-Score	FAR	Training Time (min)	Training Epochs
CNN	0.10211	0.94839	0.92303	0.98384	0.95196	0.00108	18	23
GRU	0.12445	0.93981	0.91766	0.97027	0.9428	0.002	35	17
GRU+CNN	0.09985	0.9494	0.92288	0.98494	0.95239	0.001	97	73
LSTM	0.12607	0.93939	0.91288	0.97455	0.94219	0.0017	36	14
Autoencoder+GRU	1.26009	0.71425	0.71425	0.71425	0.71425	0.02041	12	7
Autoencoder+CNN	0.11195	0.94695	0.92194	0.98104	0.95009	0.00127	32	30
Autoencoder+LSTM	0.23309	0.91507	0.87954	0.97054	0.92207	0.0019	37	12
XGBoost	0.0672	0.9641	0.965	0.9857	0.9603	0.0023	16	29

From extensive expiremetns the XGBoost model emerges as the best performer in terms of accuracy, recall, precision, and F1-score. It achieves an accuracy of 96.41%, indicating its ability to correctly classify the majority of samples. The recall of 96.50% highlights its effectiveness in identifying true positive instances, while the precision of 98.57% signifies its low false positive rate. The high F1-score of 96.03% indicates a good balance between precision and recall, reflecting the model's overall performance. In the research work that implemented the Catboost model for intrusion detection in IoT systems [82], the reported results showed an accuracy of 100% during training and a validation accuracy of 99.27%. Additionally, they achieved high values for precision(98.42%) and recall(98.78%), indicating excellent performance in classifying intrusions. Comparing the two models, it is evident that both Catboost and XGBoost achieved high accuracy and performed well in classifying intrusions. The Catboost model reported slightly higher accuracy during training, but both models demonstrated strong precision and recall scores.it's essential to consider various evaluation metrics, check for overfitting, and analyze the context of the problem before concluding that a high accuracy indicates a better model.

Moving on to the average performers, we have the CNN, GRU, GRU+CNN, and LSTM models. These models consistently achieve high accuracy values ranging

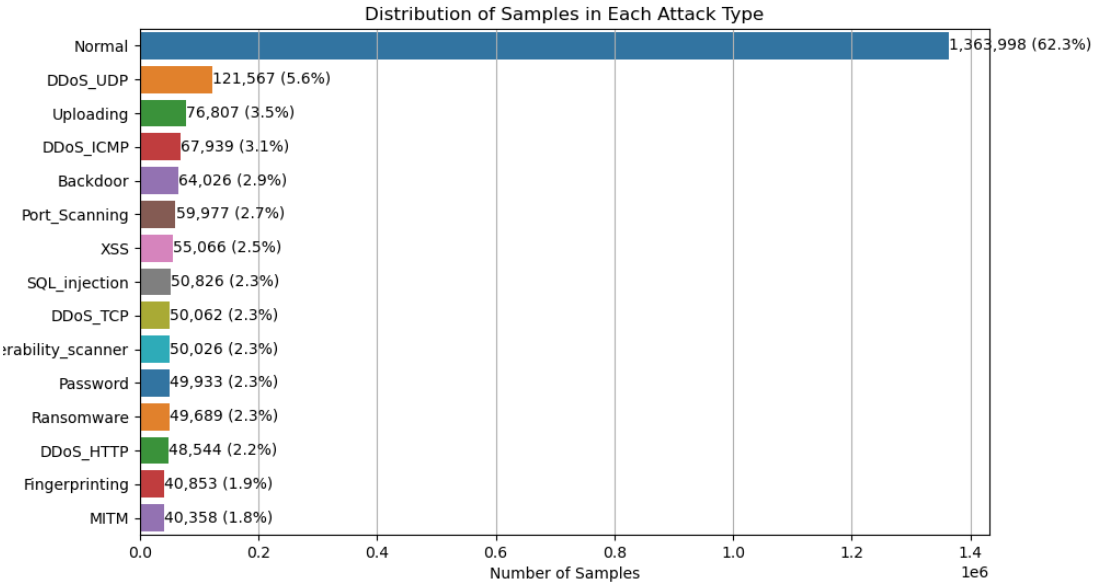


Figure 6.1: Distribution of samples for XGBoost model

from 93.94% to 94.94%. While they may not surpass the performance of XGBoost, their accuracy rates demonstrate their ability to accurately classify the majority of samples. The recall values ranging from 91.29% to 92.29% indicate their effectiveness in capturing true positive instances, while the precision values ranging from 97.03% to 98.49% reveal their ability to minimize false positive instances. The F1-scores ranging from 94.22% to 95.24% further emphasize the models' overall performance in achieving a balance between precision and recall. On the other hand, the Autoencoder + GRU model showcases the lowest performance among the models across the metrics with an accuracy score of 71.43%. These metrics indicate a significant gap compared to the other models, suggesting lower effectiveness in classifying the different attack types. Further analysis may be required to investigate the reasons behind this underperformance and explore ways to enhance the model's capabilities.

It is important to consider the strengths and limitations of each model. XGBoost, with its gradient boosting algorithm, demonstrates strong performance across all

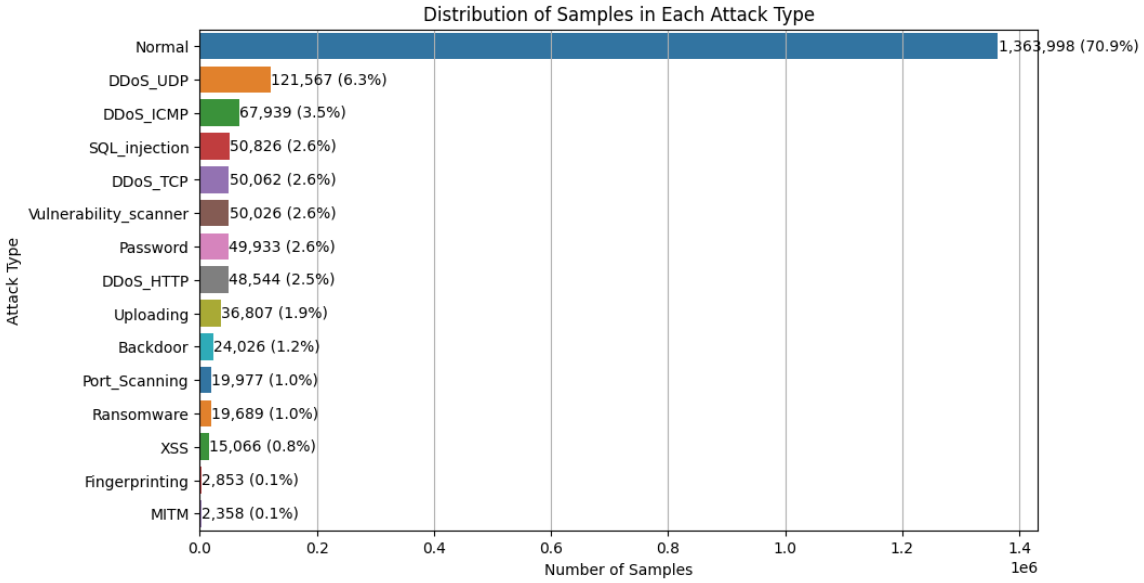


Figure 6.2: Distribution of samples for remaining models

metrics and is largely benefited by oversampling the data for minor classes, making it a suitable choice for anomaly detection, figure 6.1 represents the number of samples for each attack type used for the XGboost model. The deep learning models (CNN, GRU, GRU+CNN, and LSTM) showcase commendable accuracy rates and exhibit the ability to capture anomalies effectively.

However, it is important to note that the performance of CNN was particularly good on the original number of samples after data processing step. while the performance of the other models improved when the sample count for smaller classes was increased using oversampling techniques, figrue 6.2 shows the number of samples used for each class other than CNN and XGboost models. This indicates that the CNN model's performance degraded when faced with an increased sample count for smaller classes. The Autoencoder + GRU model, although underperforming in comparison, still provides insights into the potential of combining autoencoder-based feature extraction with the GRU architecture.

In conclusion, the XGBoost model stands out as the best performer, while the

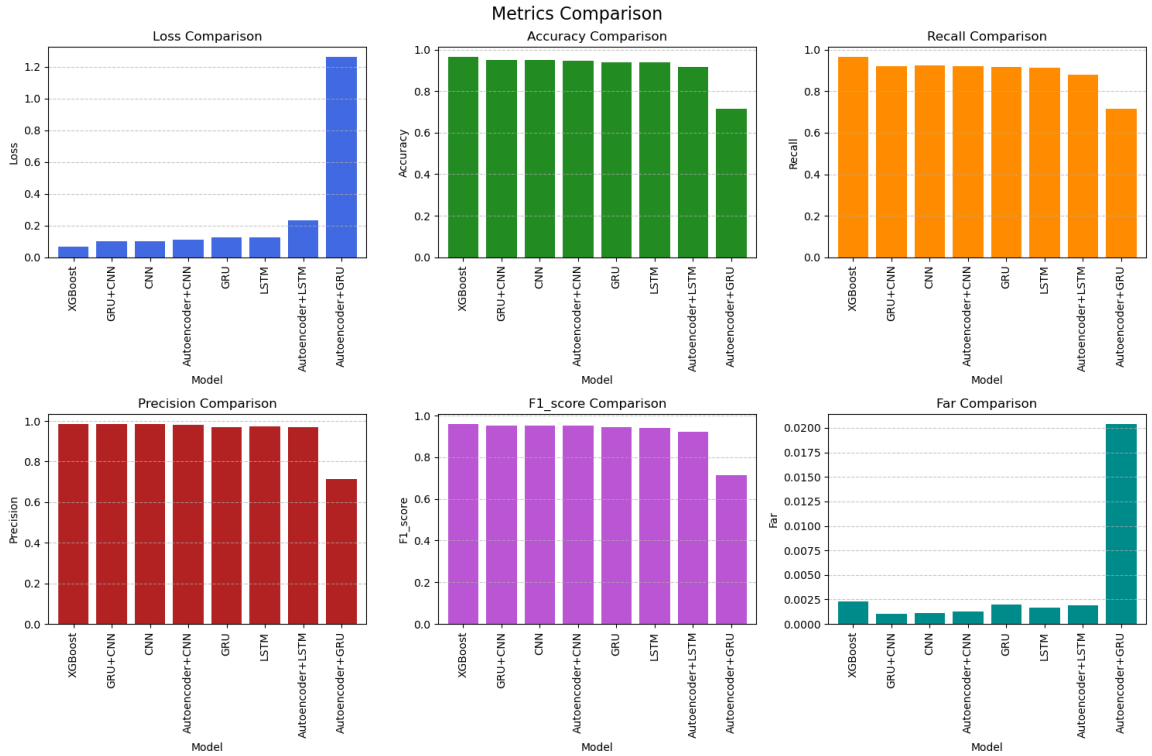


Figure 6.3: Performance Comparison

CNN, GRU, GRU+CNN, and LSTM models exhibit strong performance in detecting anomalies Table 6.1 and figure 6.3 provides an overview of the performance metrics of each model. The Autoencoder + GRU model shows room for improvement and warrants further investigation. These findings contribute to the understanding of the performance of different models and can guide the selection of appropriate models for anomaly detection in encrypted IoT traffic. Figure 6.4 shows the confusion matrix of the XGBoost model, the top performer in our study, provides valuable insights into its classification capabilities for different attack types. Each row in the matrix represents the actual labels, while each column represents the predicted labels. The numbers within the matrix indicate the count of samples that fall into each category.

The XGBoost model showcases outstanding accuracy and precision in classifying various attack types. It successfully identifies a substantial number of samples belonging to the DDoS_HTTP, DDoS_ICMP, DDoS_TCP, DDoS_UDP, Fingerprinting,

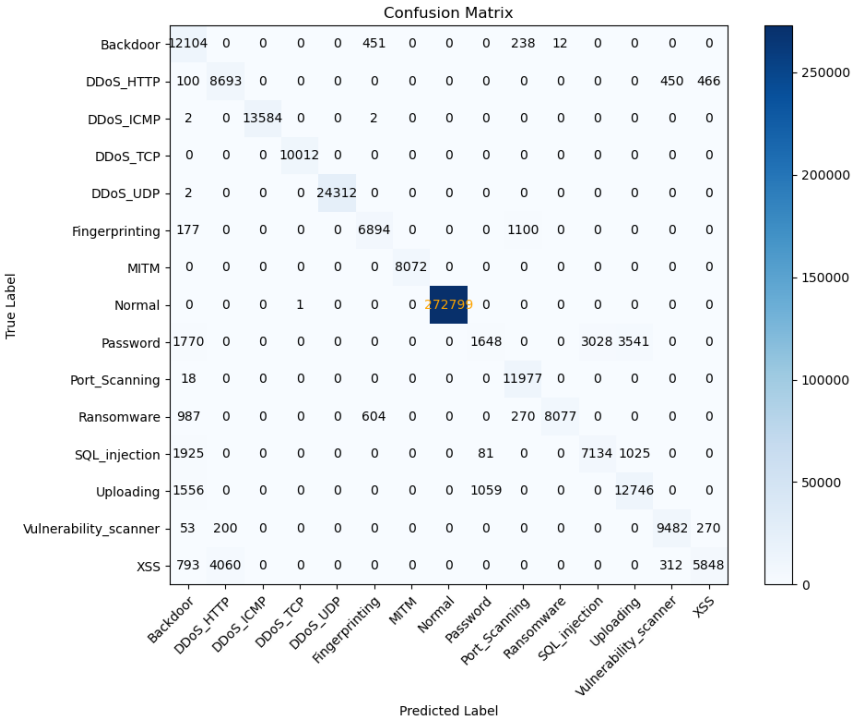


Figure 6.4: XGboost model Confusion Matrix

MITM, Password, Port Scanning, Ransomware, SQL Injection, Uploading, Vulnerability Scanner, and XSS categories.

However, there are instances of misclassifications within the confusion matrix. For instance, the model may encounter challenges in accurately distinguishing samples in the Backdoor category, resulting in a few misclassified instances. Additionally, there are a small number of misclassified samples in the Normal category, where they are mistakenly labeled as other attack types.

Overall, the XGBoost model demonstrates superior performance by accurately classifying a wide range of attack types with exceptional precision and recall. Its low false alarm rate indicates its ability to effectively avoid false positives, ensuring a high level of accuracy in detecting anomalies within encrypted IoT traffic. The confusion matrix of the CNN and GRU hybrid model is shown in Figure 6.5, which is the second-best performer after XGboost.

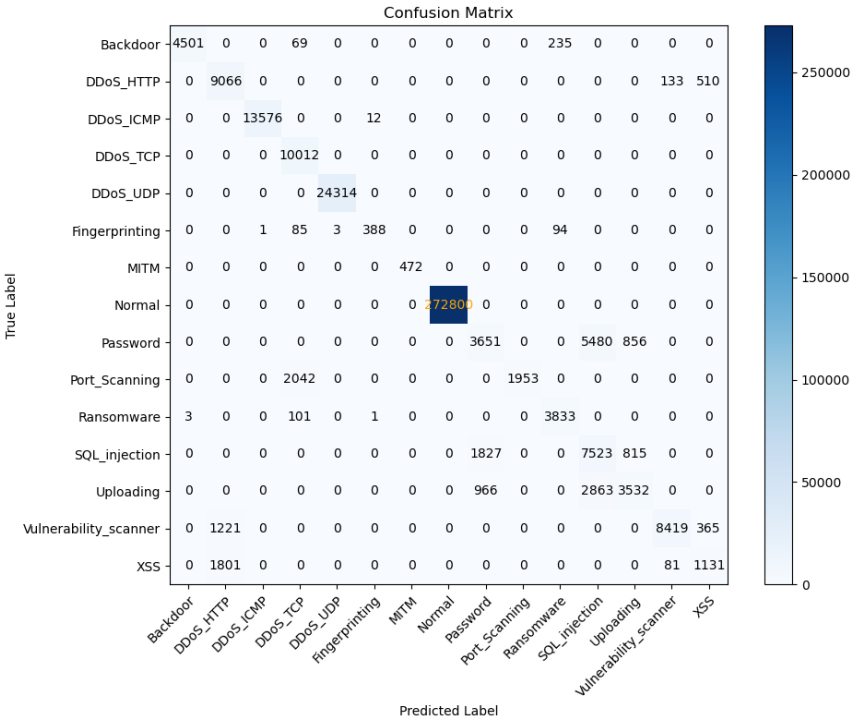


Figure 6.5: CNN+GRU Confusion Matrix

The confusion matrix demonstrates that the CNN+GRU model has distinguished the majority of attack types with a high degree of accuracy. For instance, a sizable portion of samples from the DDoS HTTP, DDoS ICMP, DDoS TCP, DDoS UDP, Fingerprinting, MITM, Password, Port Scanning, Ransomware, SQL injection, Uploading, Vulnerability scanning, and XSS categories are accurately identified. It's important to note that the model has some misclassifications as well. For instance, there are a few misclassified samples in the Backdoor category, suggesting that the model struggles to distinguish them accurately. Similarly, there are a small number of misclassified samples in the Normal category, which are mistakenly labeled as other attack types.

Based on the provided results for the CNN+GRU and XGBoost models, it's evident that both models have achieved relatively high performance across multiple metrics. Let's analyze how the other metrics influenced the low false alarm rate of

the models:

1. Accuracy: Both models have high accuracy values, indicating that they are overall effective in correctly classifying samples, including both normal and abnormal instances. High accuracy suggests that the models are making correct predictions, which is a contributing factor to a low false alarm rate.

2. Recall: The recall values are reasonably high (0.92288 and 0.965, respectively). High recall means that the models are effectively capturing a significant number of true positive anomalies. This sensitivity to detecting anomalies helps in maintaining a low false alarm rate.

3. Precision: Both models exhibit high precision values (0.98494 for CNN+GRU and 0.9857 for XGBoost). High precision indicates that the models are not generating many false positives, which is essential for maintaining a low false alarm rate.

4. F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of the model's performance in terms of both false alarms and false negatives. For both models, the F1-scores are high (0.95239 for CNN+GRU and 0.9603 for XGBoost), suggesting a good balance between precision and recall.

5. FAR (False Alarm Rate): The FAR is specifically related to false alarms, representing the proportion of negative predictions (normal samples) that are incorrectly classified as positive (anomalies). Both models achieved low FAR values (0.001 for CNN+GRU and 0.0023 for XGBoost). The low FAR indicates that the models are making minimal false positive predictions, which is crucial for a low false alarm rate.

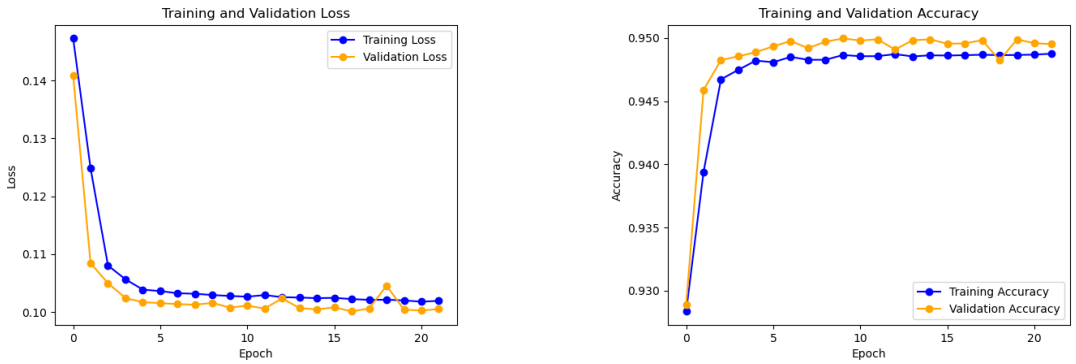
In summary, the high accuracy, recall, precision, F1-score, and the low FAR for both the CNN+GRU and XGBoost models demonstrate their effectiveness in accurately detecting anomalies with minimal false positives. The combination of

these performance metrics has influenced the low false alarm rate of the models, making them valuable tools for anomaly detection tasks.

6.3 Convergence of the Models

The convergence behavior of the models is an essential aspect to consider in the evaluation of their performance. It provides insights into the training process, including the convergence time, stability, and overall efficiency. In this study, we examined the convergence of the CNN, GRU, LSTM, CNN+GRU, Autoencoder+CNN, Autoencoder+GRU, Autoencoder+LSTM, and XGBoost models.

Figures 6.6(a)–6.12 provide valuable insights into the training progress of each model, depicting the trends in loss and accuracy across different epochs. Let's delve into each model's performance individually:

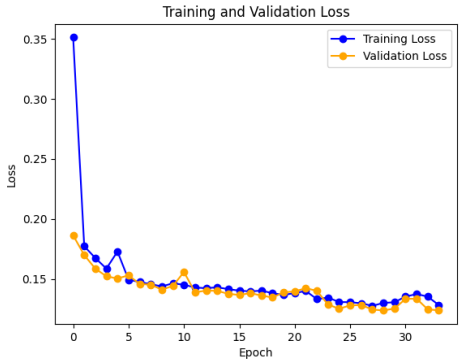


(a) CNN Loss

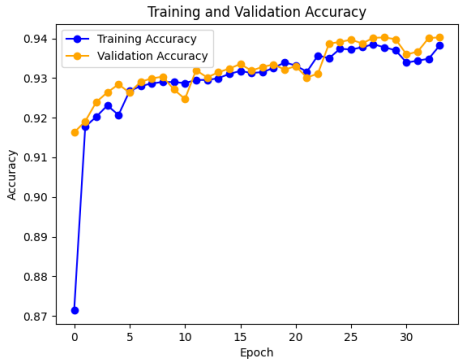
(b) CNN Accuracy

Figure 6.6: CNN Loss and Accuracy

For the CNN model (Figure 6.6(a) and Figure 6.6(b)), we observe a gradual decrease in loss and a simultaneous increase in accuracy as the number of epochs increases. This trend suggests that the model effectively minimizes the discrepancy between predicted and actual values. The model demonstrated relatively fast



(a) GRU Loss



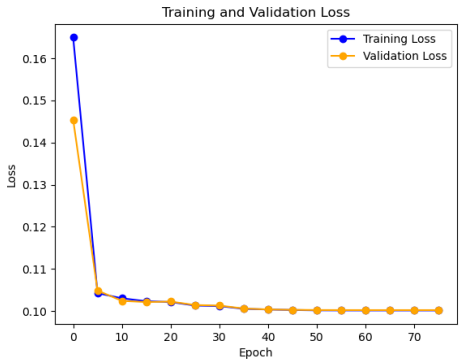
(b) GRU Accuracy

Figure 6.7: GRU Loss and Accuracy

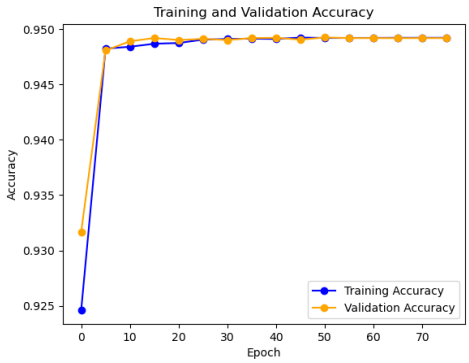
convergence, with a per-epoch runtime of 17.5 minutes and a total of 23 training epochs. This suggests that the CNN architecture efficiently learned the features from the dataset and achieved convergence in a relatively short time. The model's ability to capture spatial information using convolutional layers and its simplicity contributed to its faster convergence.

Similarly, the GRU model (Figure 6.7(a) and Figure 6.7(b)) demonstrates a decreasing loss trend overall. However, it is worth noting that some fluctuations in training and validation accuracy are observed during the training process, the GRU model showed efficient convergence, with a per-epoch runtime of 35 minutes and a total of 17 training epochs. The GRU architecture, being a type of recurrent neural network (RNN), excels at capturing temporal dependencies in sequential data. The model successfully learned the temporal patterns present in the encrypted IoT traffic, leading to its convergence within a reasonable number of epochs. These fluctuations may indicate the model's sensitivity to certain patterns in the data.

The CNN+GRU model (Figure 6.8(a) and Figure 6.8(b)) exhibits a consistently decreasing loss curve and a corresponding increase in accuracy. This behavior suggests a steady convergence towards the optimal solution, emphasizing the effectiveness of

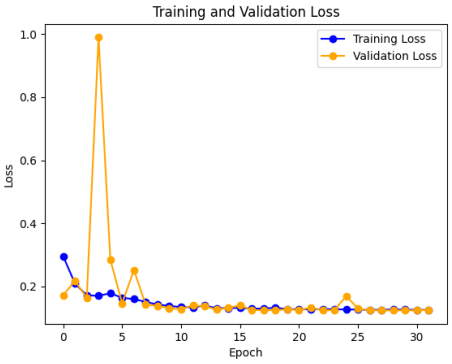


(a) CNU+GRU Loss

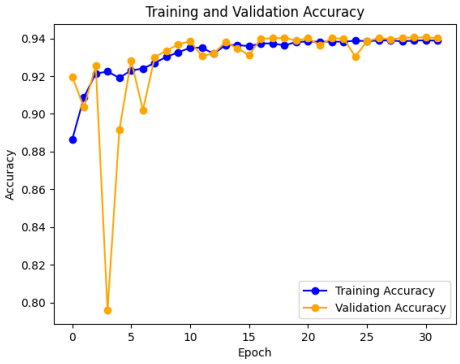


(b) CNN+GRU Accuracy

Figure 6.8: CNN+GRU Loss and Accuracy



(a) LSTM Loss

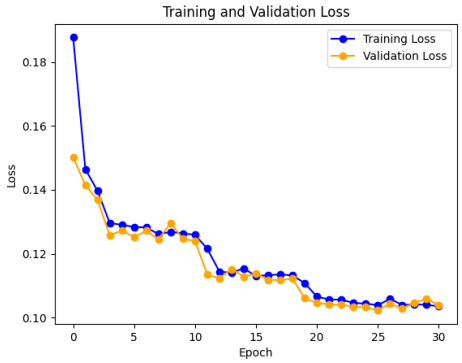


(b) LSTM Accuracy

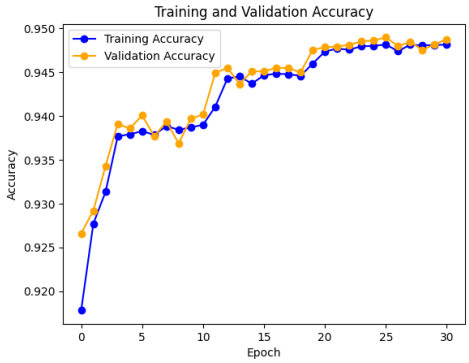
Figure 6.9: LSTM Loss and Accuracy

combining both CNN and GRU architectures for anomaly detection. The CNN+GRU hybrid model, combining the strengths of both architectures, demonstrated a longer convergence time compared to the individual models. It took 97 minutes and a total of 73 training epochs to converge. The longer convergence time can be attributed to the combined complexity of the two architectures and the need for the model to extract both spatial and temporal features simultaneously. However, despite the longer convergence time, the hybrid model achieved superior performance, indicating the effectiveness of integrating both CNN and GRU.

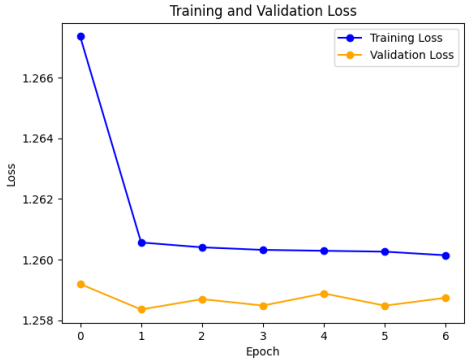
On the other hand, the LSTM model (Figure 6.9(a) and Figure 6.9(b)) shows



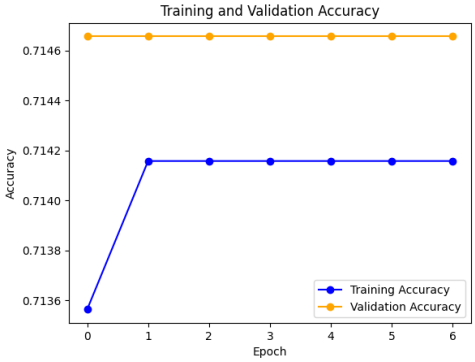
(a) Autoencoder+CNN Loss



(b) Autoencoder+CNN Accuracy



(c) Autoencoder+GRU Loss

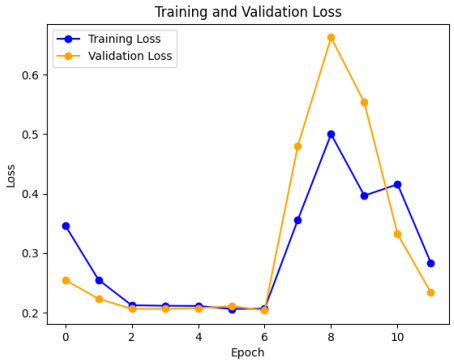


(d) Autoencoder+GRU Accuracy

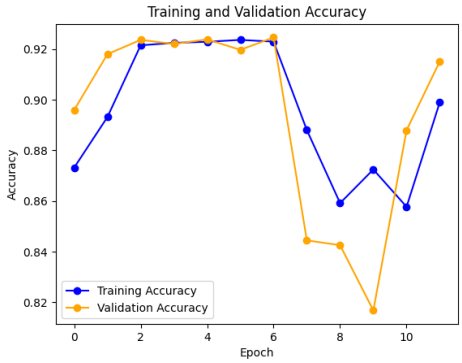
Figure 6.10: Autoencoder+CNN and Autoencoder+GRU Loss and Accuracy

a different pattern. While the loss decreases over time, there is a significant dip in accuracy during the initial epochs. This behavior may indicate a slower convergence rate or challenges in capturing temporal dependencies in the data. The model exhibited slightly slower convergence compared to CNN and GRU. It required a per-epoch runtime of 36 minutes and a total of 14 training epochs to achieve convergence. The LSTM's ability to capture long-term dependencies makes it well-suited for modeling complex sequential data. However, the additional complexity of the LSTM architecture and the longer sequence length in the encrypted IoT traffic likely contributed to the longer convergence time.

The Autoencoder-based models, Autoencoder+CNN, Autoencoder+GRU, and



(a) Autoencoder+LSTM Loss



(b) Autoencoder+LSTM Accuracy

Figure 6.11: Autoencoder+LSTM Loss and Accuracy

Autoencoder+LSTM models exhibited distinct patterns in their training curves. Figure 6.10(a) and Figure 6.11(b) demonstrate that the Autoencoder+CNN model initially showed slow reduction in loss for a few epochs, followed by a steep dive, and then gradually decreased until convergence. On the other hand, Figure 6.10(c) and Figure 6.10(d) illustrate that the Autoencoder+GRU model maintained a consistent loss curve without significant variations. Interestingly, similar to the Autoencoder+GRU model the Autoencoder+LSTM model from 6.11(a) and Figure 6.11(b) exhibited quick convergence within six epochs, however, it is followed by a notable spike in loss values, indicating overfitting. The models showed varying convergence behaviors. These models aim to learn a compact representation of the input data through an unsupervised learning process. The convergence time and number of epochs varied depending on the complexity of the autoencoder architecture and the reconstruction error optimization. In general, the convergence time was relatively shorter compared to the deep learning models, with per-epoch runtimes ranging from 12 to 37 minutes and a total of 7 to 30 training epochs.

In contrast to the Autoencoder-based models, the XGBoost model displayed a smooth convergence pattern, as evident from Figure 6.12. The loss value consistently decreased over the epochs, resembling the behavior observed in the CNN+GRU

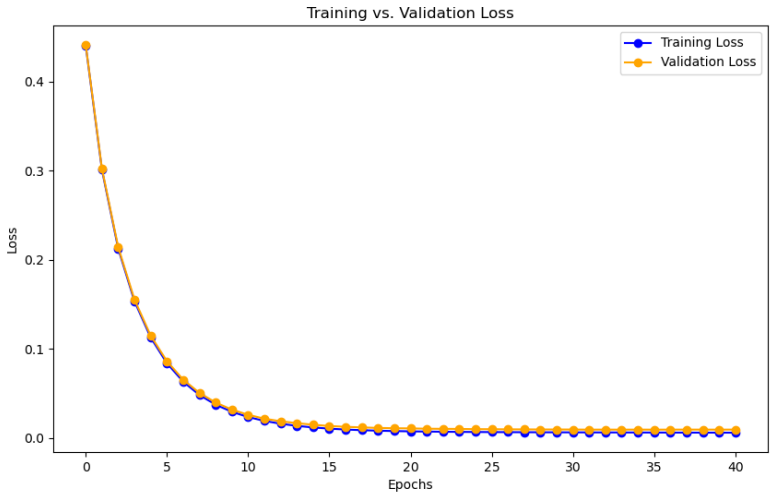


Figure 6.12: XGBoost Loss

model. This highlights the efficacy of the XGBoost algorithm in achieving a gradual reduction in loss and optimizing the model's performance. The XGBoost model, a gradient boosting algorithm, demonstrated efficient convergence with a per-epoch runtime of 16 minutes and a total of 29 training epochs. XGBoost is a powerful algorithm known for its capability to handle tabular data and perform well in classification tasks. It optimizes the objective function using gradient boosting, which leads to fast convergence and high predictive performance.

Overall, when comparing the training curves of the models, it is evident that the CNN+GRU and XGBoost models demonstrate relatively stable and favorable convergence patterns. The CNN+GRU model exhibits a steady reduction in loss, while the XGBoost model showcases a smooth and consistent decrease in loss values. These models can be considered the top performers in terms of convergence and optimization. On the other hand, the Autoencoder-based models exhibit varying convergence behaviors and may require further investigation to enhance their performance and mitigate overfitting.

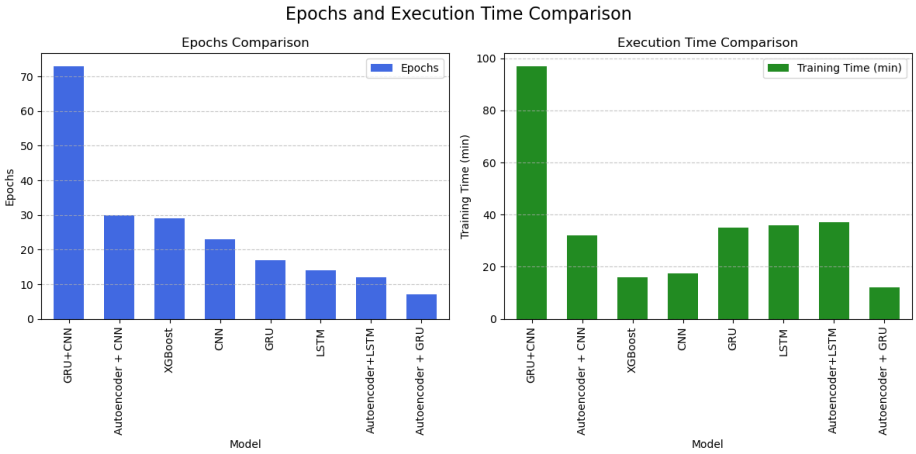


Figure 6.13: epochs and training time comparison

Figure 6.13 shows the training time and number of epochs taken by each model which reveals the computational efficiency of the models. Models with shorter training times are generally more efficient and require less computational resources. A shorter training time can be advantageous when dealing with large datasets or when performing hyperparameter tuning.

In summary, the training curves provide valuable insights into the convergence patterns of the models, the convergence behaviors of the models varied depending on their architectures and the complexity of the problem at hand. The CNN and GRU models showed relatively faster convergence, while the LSTM, the hybrid CNN+GRU model, and XGboost models required more time to achieve convergence however, the CNN+GRU and XGBoost models exhibit desirable convergence behaviors, while the Autoencoder-based models show distinctive patterns that require additional attention. Understanding these convergence characteristics helps in assessing the models' training stability and identifying areas for potential improvement.

7 Conclusion and future work

7.1 Conclusion

In this thesis, we explored and evaluated various deep-learning models for anomaly detection in edge IIoT systems. Through extensive experimentation and analysis, we have gained valuable insights into the performance and capabilities of these models. The results demonstrate the effectiveness of different neural network architectures, including CNN, GRU, CNN+GRU, LSTM, XGBoost, and Autoencoder-based models in detecting anomalies in edge IIoT data.

Among the models evaluated, the XGBoost model emerges as the top performer, consistently achieving high accuracy (96.41%), precision (98.57%), recall (96.50%), and F1 score (96.03%). CNN+GRU Hybrid model follows XGBoost with 99.94% accuracy, it combines the spatial feature extraction capabilities of CNNs with the temporal sequence modeling capabilities of GRUs, enabling it to capture both local patterns and long-term dependencies in the data. The hybrid model's superior performance highlights the importance of leveraging multiple neural network architectures for enhanced anomaly detection in edge IIoT systems. When compared the CNN+GRU models' performance by including normal data samples in the training data of the CNN+GRU model has proven to be highly beneficial. The model's performance significantly improved, achieving a higher accuracy of 94.94% (compared to 82.44% without normal samples), a recall of 92.29% (compared to 73.52%), precision of 98.49% (compared to 93.63%), F1-Score of 95.24% (compared to 82.18%), and a

reduced false alarm rate (FAR) of 0.001 (compared to 0.0038). By learning from normal data patterns, the model becomes more effective in distinguishing abnormal patterns, leading to better anomaly detection. The results highlight the importance of using a balanced dataset that includes both normal and abnormal samples to enhance the model's performance and real-world applicability in detecting anomalies accurately and efficiently.

However, it is worth noting that not all models perform equally well. The Autoencoder-based models, while exhibiting reasonable accuracy (71.43%), fall short in terms of precision (71.43%), recall (71.43%), and F1 score (71.43%). This suggests that the unsupervised nature of Autoencoders may limit their ability to accurately identify anomalies in complex edge IIoT data. Further investigation and improvement are needed to enhance the performance of these models.

Furthermore, computational efficiency is a critical factor in real-world implementation. The CNN model stands out as the most efficient, with faster convergence and lower training times. On the other hand, the hybrid CNN+GRU model requires significantly more computational resources but achieves the highest detection performance. This trade-off between performance and computational efficiency needs to be carefully considered when deploying these models in resource-constrained edge IIoT environments.

7.2 Future Work

This research opens up several avenues for future work in the field of anomaly detection in edge IIoT systems. Some potential areas of exploration include:

1. **Model Optimization**: Further optimization techniques can be applied to improve the performance and efficiency of the hybrid CNN+GRU model. This can

involve architecture refinement, hyperparameter tuning, and regularization methods to reduce overfitting and enhance generalization.

2. **Data Augmentation**: Investigate the effectiveness of data augmentation techniques in augmenting the limited labeled dataset for training deep learning models. Techniques such as synthetic data generation, oversampling, and augmentation through transformations can help address the data scarcity challenge.

3. **Transfer Learning**: Explore the applicability of transfer learning by leveraging pre-trained models on large-scale datasets for anomaly detection in edge IIoT systems. This approach can potentially enhance the performance of models by leveraging knowledge learned from related domains.

4. **Ensemble Methods**: Investigate the potential of ensemble methods by combining multiple models to achieve higher accuracy and robustness. Techniques such as model averaging, stacking, and boosting can be explored to harness the collective intelligence of diverse models.

5. **Real-time Anomaly Detection**: Develop real-time anomaly detection systems that can process and analyze streaming data from edge IIoT devices in real time. This would involve designing efficient data ingestion, processing, and prediction pipelines to enable timely detection and response to anomalies.

6. **Interpretability and Explainability**: Focus on enhancing the interpretability and explainability of deep learning models for anomaly detection. Techniques such as attention mechanisms, visualization methods, and rule-based post-processing can provide insights into the decision-making process of the models.

In conclusion, this thesis contributes to the field of anomaly detection in edge IIoT systems by evaluating and comparing various deep learning models. The findings

shed light on the performance, strengths, and limitations of these models, guiding future research and development efforts. By addressing the challenges of anomaly detection, we can enhance the reliability, security, and efficiency of edge IIoT systems, paving the way for their widespread adoption and deployment.

REFERENCES

[1] A. C. Bahnsen, "Building ai applications using deep learning," *Blog Total Fraud Protection*, 2016.

[2] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, 2016, pp. 21–26.

[3] X. Xie, C. Wang, S. Chen, G. Shi, and Z. Zhao, "Real-time illegal parking detection system based on deep learning," in *Proceedings of the 2017 international conference on deep learning technologies*, 2017, pp. 23–27.

[4] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *Information Processing in Medical Imaging: 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings*. Springer, 2017, pp. 146–157.

[5] J. Xiong, S. Bharati, and P. Podder, "Machine and deep learning for iot security and privacy: Applications, challenges, and future directions," *Security and Communication Networks*, vol. 2022, p. 8951961, 2022. [Online]. Available: https://doi.org/10.1155/2022/8951961

[6] V. Chandola, A. Banerjee, and V. Kumar, "Outlier detection: A survey," *ACM Computing Surveys*, vol. 14, p. 15, 2007.

[7] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11.

[8] H.-K. Peng and R. Marculescu, "Multi-scale compositionality: identifying the compositional structures of social dynamics using deep learning," *PloS one*, vol. 10, no. 4, p. e0118309, 2015.

[9] A. Diro, N. Chilamkurti, V.-D. Nguyen, and W. Heyne, "A comprehensive study of anomaly detection schemes in iot networks using machine learning algorithms," *Sensors*, vol. 21, no. 24, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/24/8320

[10] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17308488

[11] A. Diro, N. Chilamkurti, V.-D. Nguyen, and W. Heyne, "A comprehensive study of anomaly detection schemes in iot networks using machine learning algorithms," *Sensors*, vol. 21, no. 24, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/24/8320

[12] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏot: A federated self-learning anomaly detection system for iot," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 756–767.

[13] A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly detection for iot time-series data: A survey," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2020.

[14] Y. Himeur, K. Ghanem, A. Alsalemi, F. Bensaali, and A. Amira, "Artificial intelligence based anomaly detection of energy consumption

in buildings: A review, current trends and new perspectives," *Applied Energy*, vol. 287, p. 116601, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261921001409

[15] N. Alexopoulos, E. Vasilomanolakis, N. R. Ivánkó, and M. Mühlhäuser, "Towards blockchain-based collaborative intrusion detection systems," in *Critical Information Infrastructures Security*, G. D'Agostino and A. Scala, Eds. Cham: Springer International Publishing, 2018, pp. 107–118.

[16] L. Wen, X. Li, L. Gao, and Y. Zhang, "A new convolutional neural network-based data-driven fault diagnosis method," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5990–5998, 2018.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=6795963isnumber=6795273

[18] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," 2016.

[19] S. Chauhan and L. Vig, "Anomaly detection in ecg time signals via deep long short-term memory networks," in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2015, pp. 1–7.

[20] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 665–674. [Online]. Available: https://doi.org/10.1145/3097983.3098052

[21] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S. Ng, "Madgan: Multivariate anomaly detection for time series data with generative adversarial networks: 703–716," 2019.

[22] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 1409–1416, Jul. 2019. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/3942

[23] T.-T.-H. Le, Y. E. Oktian, and H. Kim, "Xgboost for imbalanced multiclass classification-based industrial internet of things intrusion detection systems," *Sustainability*, vol. 14, no. 14, 2022. [Online]. Available: https://www.mdpi.com/2071-1050/14/14/8707

[24] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515002891

[25] K. Hayat, T. Bakhshi, and B. Ghita, "Anomaly detection in encrypted internet traffic using hybrid deep learning," *Security and Communication Networks*, vol. 2021, p. 5363750, 2021. [Online]. Available: https://doi.org/10.1155/2021/5363750

[26] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[27] S. Shadroo, A. M. Rahmani, and A. Rezaee, "The two-phase scheduling based on deep learning in the internet of things," *Computer Networks*, vol. 185, p. 107684, 2021.

[28] M. A. Rahman and M. S. Hossain, "An internet-of-medical-things-enabled edge computing framework for tackling covid-19," *IEEE Internet of Things Journal*, vol. 8, no. 21, pp. 15 847–15 854, 2021.

[29] F. Liang, W. Yu, X. Liu, D. Griffith, and N. Golmie, "Toward edge-based deep learning in industrial internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4329–4341, 2020.

[30] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[31] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[32] S. Bharati, P. Podder, and M. R. H. Mondal, "Hybrid deep learning for detecting lung diseases from x-ray images," *Informatics in Medicine Unlocked*, vol. 20, p. 100391, 2020.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[34] S. Tofigh, M. O. Ahmad, and M. Swamy, "A low-complexity modified thinet algorithm for pruning convolutional neural networks," *IEEE Signal Processing Letters*, vol. 29, pp. 1012–1016, 2022.

[35] X.-W. Chen and X. Lin, "Big data deep learning: challenges and perspectives," *IEEE access*, vol. 2, pp. 514–525, 2014.

[36] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010: 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III 20*. Springer, 2010, pp. 92–101.

[37] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-second international joint conference on artificial intelligence*. Citeseer, 2011.

[38] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[40] L. Zhang, L. Zhang, and B. Du, "Deep learning for remote sensing data: A technical tutorial on the state of the art," *IEEE Geoscience and remote sensing magazine*, vol. 4, no. 2, pp. 22–40, 2016.

[41] E. De Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens, P. Demeester, and B. Dhoedt, "Distributed neural networks for internet of things: The big-little approach," in *Internet of Things. IoT Infrastructures: Second International Summit, IoT 360° 2015, Rome, Italy, October 27-29, 2015, Revised Selected Papers, Part II*. Springer, 2016, pp. 484–492.

[42] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé *et al.*, "Deep android malware

detection," in *Proceedings of the seventh ACM on conference on data and application security and privacy*, 2017, pp. 301–308.

[43] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*. Springer, 2016, pp. 3–26.

[44] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[45] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.

[46] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A survey on deep learning for big data," *Information Fusion*, vol. 42, pp. 146–157, 2018.

[47] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[48] P. Torres, C. Catania, S. Garcia, and C. G. Garino, "An analysis of recurrent neural networks for botnet detection behavior," in *2016 IEEE biennial congress of Argentina (ARGENCON)*. IEEE, 2016, pp. 1–6.

[49] G. E. Hinton, "A practical guide to training restricted boltzmann machines," *Neural Networks: Tricks of the Trade: Second Edition*, pp. 599–619, 2012.

[50] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, "Network anomaly detection with the restricted boltzmann machine," *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[51] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[52] H. F. Nweke, Y. W. Teh, M. A. Al-Garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Systems with Applications*, vol. 105, pp. 233–261, 2018.

[53] Y. Chen, Y. Zhang, S. Maharjan, M. Alam, and T. Wu, "Deep learning for secure mobile edge computing in cyber-physical transportation systems," *IEEE Network*, vol. 33, no. 4, pp. 36–41, 2019.

[54] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," *International Journal of Security and Its Applications*, vol. 9, no. 5, pp. 205–216, 2015.

[55] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[56] S. Harush, Y. Meidan, and A. Shabtai, "Deepstream: autoencoder-based stream temporal clustering and anomaly detection," *Computers & Security*, vol. 106, p. 102276, 2021.

[57] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 3854–3861.

[58] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[59] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[60] R. E. Hiromoto, M. Haney, and A. Vakanski, "A secure architecture for iot with supply chain risk management," in *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 1. IEEE, 2017, pp. 431–435.

[61] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[62] A. Mohebbi, H. Abdzadeh-Ziabari, W.-P. Zhu, and M. O. Ahmad, "Doubly selective channel estimation algorithms for millimeter wave hybrid mimo systems," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 12 821–12 835, 2021.

[63] A. Esmaeilzehi, M. O. Ahmad, and M. Swamy, "Srnharb: A deep light-weight image super resolution network using hybrid activation residual blocks," *Signal Processing: Image Communication*, vol. 99, p. 116509, 2021.

[64] M. R. H. Mondal, S. Bharati, and P. Podder, "Co-irv2: Optimized inceptionresnetv2 for covid-19 detection from chest ct images," *PloS one*, vol. 16, no. 10, p. e0259179, 2021.

[65] R. R. Reddy, Y. Ramadevi, and K. Sunitha, "Enhanced anomaly detection using ensemble support vector machine," in *2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC)*. IEEE, 2017, pp. 107–111.

[66] H. H. Bosman, G. Iacca, A. Tejada, H. J. Wörtche, and A. Liotta, "Ensembles of incremental learners to detect anomalies in ad hoc sensor networks," *Ad Hoc Networks*, vol. 35, pp. 14–36, 2015, special Issue on Big Data Inspired Data Sensing, Processing and Networking Technologies. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870515001481

[67] F. Amiri, M. Rezaei Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1184–1199, 2011, advanced Topics in Cloud Computing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804511000038

[68] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[69] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International conference on machine learning*. PMLR, 2013, pp. 1058–1066.

[70] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 646–661.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[72] S. Singh, D. Hoiem, and D. Forsyth, "Swapout: Learning an ensemble of deep architectures," *Advances in neural information processing systems*, vol. 29, 2016.

[73] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2014.

[74] M. E. Aminanto and K. Kim, "Detecting active attacks in wi-fi network by semi-supervised deep learning," in *Conference on Information Security and Cryptography*, 2017.

[75] I.-S. Comşa, S. Zhang, M. E. Aydin, P. Kuonen, Y. Lu, R. Trestian, and G. Ghinea, "Towards 5g: A reinforcement learning-based scheduling solution for data traffic management," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1661–1675, 2018.

[76] F. Hussain, A. Anpalagan, A. S. Khwaja, and M. Naeem, "Resource allocation and congestion control in clustered m2m communication using q-learning," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 4, p. e3039, 2017.

[77] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Secure computation offloading in blockchain based iot networks with deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 4, pp. 3192–3208, 2021.

[78] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[79] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[80] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[81] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning," *IEEE Access*, vol. 10, pp. 40 281–40 306, 2022.

[82] M. Douiba, S. Benkirane, A. Guezzaz, and M. Azrour, "An improved anomaly detection model for iot security using decision tree and gradient boosting," *The Journal of Supercomputing*, vol. 79, no. 3, pp. 3392–3411, 2023.

8 Appendix

In this chpater, additional information and supporting materials are provided to further enhance the understanding of the research presented in this thesis. The appendix includes supplementary figures, tables, code snippets, or any other relevant details that provide additional context or evidence for the findings and conclusions presented in the main body of the thesis. These additional resources aim to provide a more comprehensive view of the research process and analysis conducted throughout the study.

8.1 Code

8.1.1 Datapreprocessing

Datset: https://www.kaggle.com/datasets/mohamedamineferrag/edgeiiotset-cyber-security-dataset-of-iot-iiot

```
import os
import numpy as np
import pandas as pd
import tensorflow as tf

df = pd.read_csv('DNN-EdgeIIoT-dataset.csv', low_memory=False)

# Data preprocessing steps...
```

```
from sklearn.utils import shuffle
drop_columns = ["frame.time", "ip.src_host", "ip.dst_host",
                "arp.src.proto_ipv4","arp.dst.proto_ipv4",
                "http.file_data","http.request.full_uri",
                "icmp.transmit_timestamp",
                "http.request.uri.query", "tcp.options",
                "tcp.payload","tcp.srcport",
                "tcp.dstport", "udp.port", "mqtt.msg"]

df.drop(drop_columns, axis=1, inplace=True)
df.dropna(axis=0, how='any', inplace=True)
df.drop_duplicates(subset=None, keep="first", inplace=True)
df = shuffle(df)
df.isna().sum()
print(df['Attack_type'].value_counts())

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)
```

```
encode_text_dummy(df,'http.request.method')
encode_text_dummy(df,'http.referer')
encode_text_dummy(df,"http.request.version")
encode_text_dummy(df,"dns.qry.name.len")
encode_text_dummy(df,"mqtt.conack.flags")
encode_text_dummy(df,"mqtt.protoname")
encode_text_dummy(df,"mqtt.topic")

empty_cols = [col for col in df.columns if df[col].isnull().all()]
empty_cols

skip_list = ["icmp.unused", "http.tls_port", "dns.qry.type", "mqtt.msg_decoded_as"
df[skip_list[3]].value_counts()
X = df.drop([label_col], axis=1)
y = df[label_col]
```

8.1.2 Imbalanced Data Handling

```
from imblearn.over_sampling import RandomOverSampler
import warnings
warnings.filterwarnings("ignore") #'Port_Scanning', 'XSS',
minority_classes = [ 'Ransomware', 'Fingerprinting', 'MITM']
desired_samples = {
    # 'Port_Scanning': 20000,
    # 'XSS': 20000,
    'Ransomware': 10000,
```

```
    'Fingerprinting': 2000,
    'MITM': 2000
}
mask = df[label_col].isin(minority_classes)
minority_mask = df[label_col].isin(minority_classes)

X_minority = X[minority_mask]
y_minority = y[minority_mask]

# Imbalanced data handling steps...

oversample = RandomOverSampler(sampling_strategy={k: desired_samples[k] if k in de
                               random_state=42)
X_oversampled, y_oversampled = oversample.fit_resample(X[mask], y[mask])

# Concatenate the oversampled data with the original data
X_balanced = pd.concat([X, X_oversampled])
y_balanced = pd.concat([y, y_oversampled])
```

8.1.3 Data Splitting

```
from sklearn.model_selection import train_test_split

# Split into train and test sets
#X_train, X_test, y_train, y_test = train_test_split(X_balanced,
y_balanced, test_size=0.2, random_state=1, stratify=y_balanced)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1, stratify=y)
```

```
# Split train set into train and validation sets
#X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=1, stratify=y_train)

#### for XGBoost model
from sklearn.model_selection import train_test_split

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_s
```

8.1.4 Data Encoding

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

# Data encoding steps...

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train =  label_encoder.fit_transform(y_train)
y_test = label_encoder.fit_transform(y_test)
label_encoder.classes_

from sklearn.preprocessing import MinMaxScaler
```

```
min_max_scaler = MinMaxScaler()
X_train =  min_max_scaler.fit_transform(X_train)
X_test = min_max_scaler.transform(X_test)

import numpy as np
# assuming X_train and X_test are DataFrames
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

input_shape = X_train.shape[1:]
num_classes = len(np.unique(y_train))
num_classes

from  tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train, num_classes=num_classes)
y_test = to_categorical(y_test, num_classes=num_classes)

### for XGBoost model
from sklearn.preprocessing import LabelEncoder


label_encoder = LabelEncoder()
y_train =  label_encoder.fit_transform(y_train)
y_val = label_encoder.fit_transform(y_val)
y_test = label_encoder.fit_transform(y_test)
```

```
label_encoder.classes_


from sklearn.preprocessing import MinMaxScaler


min_max_scaler = MinMaxScaler()
X_train =  min_max_scaler.fit_transform(X_train)
X_val = min_max_scaler.fit_transform(X_val)
X_test = min_max_scaler.transform(X_test)


input_shape = X_train.shape[1:]
num_classes = len(np.unique(y_train))
num_classes


from  tensorflow.keras.utils import to_categorical


y_train = to_categorical(y_train, num_classes=num_classes)
y_val = to_categorical(y_val, num_classes=num_classes)
y_test = to_categorical(y_test, num_classes=num_classes)
```

8.1.5 Metrics:

```
import keras.backend as K
import tensorflow as tf
def f1_score(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
```

```
    recall = true_positives / (possible_positives + K.epsilon())



    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val

def false_alarm_rate(y_true, y_pred):
    true_negatives = K.sum(K.round(K.clip((1 - y_true) * (1 - y_pred), 0, 1)))
    false_positives = K.sum(K.round(K.clip((1 - y_true) * y_pred, 0, 1)))
    return false_positives / (false_positives + true_negatives)

# Define custom metric precision
def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision
```

8.1.6 Models

CNN Model:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
Dropout, BatchNormalization

model = Sequential()
model.add(Conv1D(64, 3, activation='relu', input_shape=(input_shape)))
```

```
model.add(MaxPooling1D(2))
model.add(Conv1D(128, 3, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Conv1D(256, 3, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Conv1D(512, 3, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Conv1D(256, 3, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes,activation='softmax'))
```

GRU Model:

```
model = Sequential()
model.add(GRU(units=32, input_shape=input_shape, implementation=2,
activation='tanh', recurrent_activation='sigmoid', recurrent_dropout=0,
unroll=False, use_bias=True, reset_after=True, return_sequences=True))
model.add(GRU(units=64, implementation=2, activation='tanh',
recurrent_activation='sigmoid', recurrent_dropout=0, unroll=False,
use_bias=True, reset_after=True))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=16, activation='relu'))
model.add(Dense(units=num_classes, activation='softmax'))
```

LSTM Model:

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.layers import GlobalAveragePooling1D


# model = Sequential()
# model.add(LSTM(units=128, activation = 'tanh', input_shape=input_shape, return_s
# recurrent_activation = 'sigmoid',
# recurrent_dropout = 0,
# unroll = False,
# use_bias = True))
# model.add(LSTM(256, activation = 'tanh'))
# model.add(LSTM(512, activation = 'tanh'))
# model.add(Dense(units=128, activation='tanh'))
# model.add(Dense(num_classes, activation='softmax'))

model = Sequential()
model.add(LSTM(units=128, activation = 'tanh', input_shape=input_shape, return_seq
model.add(LSTM(256, activation = 'tanh'))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

CNN+GRU modell:

```
# CNN model
cnn_input = Input(shape=input_shape)
cnn_layer = Conv1D(64, 3, activation='relu')(cnn_input)
cnn_layer = MaxPooling1D(2)(cnn_layer)
cnn_layer = Conv1D(128, 3, activation='relu')(cnn_layer)
cnn_layer = MaxPooling1D(2)(cnn_layer)
cnn_layer = Conv1D(256, 3, activation='relu')(cnn_layer)
cnn_layer = MaxPooling1D(2)(cnn_layer)
# cnn_layer = Conv1D(512, 3, activation='relu')(cnn_layer)
# cnn_layer = MaxPooling1D(2)(cnn_layer)
cnn_layer = Conv1D(256, 3, activation='relu')(cnn_layer)
cnn_layer = MaxPooling1D(2)(cnn_layer)
cnn_layer = Flatten()(cnn_layer)
cnn_layer = Dense(64, activation='relu')(cnn_layer)
cnn_layer = Dense(64, activation='relu')(cnn_layer)
gru_input = Input(shape=input_shape)
gru_layer = GRU(units=32, input_shape=input_shape, implementation=2,
activation='tanh', recurrent_activation='sigmoid',
recurrent_dropout=0, unroll=False, use_bias=True,
# Inputs, if masked, are strictly right-padded
reset_after=True)(gru_input)#return_sequences = True
#gru_layer = GRU(units=64, activation='tanh')(gru_layer)
#gru_layer = Dense(units=32, activation='tanh')(gru_layer)
gru_layer = Dense(units=32, activation='tanh')(gru_layer)
# Concatenate the output from the CNN and GRU models
concat_layer = concatenate([cnn_layer, gru_layer])
```

```
# Output layer
output_layer = Dense(num_classes, activation='softmax')(concat_layer)

# Define the model
model = Model(inputs=[cnn_input, gru_input], outputs=output_layer)
```

Autoencoder+CNN:

```
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D,
Flatten, UpSampling1D, Dense
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras import regularizers
# Define the encoder model
encoder_inputs = Input(shape=input_shape)
x = Conv1D(32, 3, activation='relu', padding='same')(encoder_inputs)
x = MaxPooling1D(2, padding='same')(x)
x = Conv1D(64, 3, activation='relu', padding='same')(x)
x = MaxPooling1D(2, padding='same')(x)
x = Conv1D(128, 3, activation='relu', padding='same')(x)
encoder_outputs = MaxPooling1D(2, padding='same')(x)

# Define the decoder model
x = Conv1D(128, 3, activation='relu', padding='same')(encoder_outputs)
x = UpSampling1D(2)(x)
x = Conv1D(64, 3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)
x = Conv1D(32, 3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)
```

```
decoder_outputs = Conv1D(1, 3, activation='sigmoid', padding='same')(x)

# Combine the encoder and decoder models to form the autoencoder
encoder = Model(inputs=encoder_inputs, outputs=encoder_outputs)
decoder = Model(inputs=encoder_outputs, outputs=decoder_outputs)
autoencoder = Model(inputs=encoder_inputs, outputs=decoder_outputs)
# Define the classifier model
model = Sequential()
model.add(autoencoder)
model.add(Conv1D(64, 3, activation='relu', padding='same'))
model.add(MaxPooling1D(2, padding='same'))
model.add(Conv1D(128, 3, activation='relu', padding='same'))
model.add(MaxPooling1D(2, padding='same'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

Autoencoder+LSTM:

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D,
Flatten, Reshape, UpSampling1D, GRU, Dense
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
```

```
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras import regularizers
# Define the encoder model
encoder_inputs = Input(shape=input_shape)
x = Conv1D(32, 3, activation='relu', padding='same')(encoder_inputs)
x = MaxPooling1D(2, padding='same')(x)
x = Conv1D(64, 3, activation='relu', padding='same')(x)
x = MaxPooling1D(2, padding='same')(x)
x = Conv1D(128, 3, activation='relu', padding='same')(x)
encoder_outputs = MaxPooling1D(2, padding='same')(x)

# Define the decoder model
x = Conv1D(128, 3, activation='relu', padding='same')(encoder_outputs)
x = UpSampling1D(2)(x)
x = Conv1D(64, 3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)
x = Conv1D(32, 3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)
decoder_outputs = Conv1D(1, 3, activation='sigmoid', padding='same')(x)

# Combine the encoder and decoder models to form the autoencoder
encoder = Model(inputs=encoder_inputs, outputs=encoder_outputs)
decoder = Model(inputs=encoder_outputs, outputs=decoder_outputs)
autoencoder = Model(inputs=encoder_inputs, outputs=decoder_outputs)

model = Sequential()
model.add(autoencoder)
```

```
model.add(LSTM(units=64, activation = 'tanh',
input_shape= autoencoder.output_shape[1:],
return_sequences=True))
model.add(LSTM(128, activation = 'tanh'))
model.add(Dense(num_classes, activation='softmax',
kernel_regularizer=regularizers.l2( l2=0.01)))
model.summary()
```

Autoencoder+GRU:

```
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D,
Flatten, Reshape, UpSampling1D, GRU, Dense
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import categorical_crossentropy
# Define the encoder model
encoder_inputs = Input(shape=input_shape)
x = Conv1D(32, 3, activation='relu', padding='same')(encoder_inputs)
x = MaxPooling1D(2, padding='same')(x)
x = Conv1D(64, 3, activation='relu', padding='same')(x)
x = MaxPooling1D(2, padding='same')(x)
x = Conv1D(128, 3, activation='relu', padding='same')(x)
encoder_outputs = MaxPooling1D(2, padding='same')(x)

# Define the decoder model
x = Conv1D(128, 3, activation='relu', padding='same')(encoder_outputs)
x = UpSampling1D(2)(x)
```

```
x = Conv1D(64, 3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)
x = Conv1D(32, 3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)
decoder_outputs = Conv1D(1, 3, activation='sigmoid', padding='same')(x)

# Combine the encoder and decoder models to form the autoencoder
encoder = Model(inputs=encoder_inputs, outputs=encoder_outputs)
decoder = Model(inputs=encoder_outputs, outputs=decoder_outputs)
autoencoder = Model(inputs=encoder_inputs, outputs=decoder_outputs)

model = Sequential()
model.add(autoencoder)
model.add(GRU(units=32, input_shape= autoencoder.output_shape[1:],
implementation=2, activation = 'tanh',
recurrent_activation = 'sigmoid',
recurrent_dropout = 0,
unroll = False,
use_bias = True,
#Inputs, if masked, are strictly right-padded
reset_after = True)) #for GRU only
# Add additional layers as needed
model.add(Dense(units=32, activation='tanh'))
#model.add(Dense(units=64, activation='tanh'))

model.add(Dense(units=num_classes, activation='softmax'))
```

XGBoost:

```
xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train, y_train, eval_set=[(X_train, y_train),
(X_val, y_val)], early_stopping_rounds=5)
```

8.1.7 Model Compilation

```
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau
from keras.metrics import Recall, Precision
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.callbacks import TensorBoard
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss= 'categorical_crossentropy',
                  metrics=['accuracy', Recall(), Precision(),
                  f1_score,false_alarm_rate])

early_stopping = EarlyStopping(monitor="val_loss", mode="min",
verbose=1, patience=5)
lr_reduce = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=5, mode="min", verbose=1, min_lr=0)
# Define the TensorBoard callback
tensorboard_callback = TensorBoard(log_dir='./logs')
call_backs = [early_stopping,tensorboard_callback]#,lr_reduce
```

8.1.8 Training models

```
### for CNN+GRU model
EPOCHS = 100
BATCH_SIZE = 256
# Train the model
history = model.fit([X_train, X_train],y_train,
                    validation_split=0.2,
                    epochs=EPOCHS,
                    batch_size=BATCH_SIZE,
                    callbacks=call_backs,
                    verbose=1)


### for CNN, LSTM, GRU and autoencoder models
=

EPOCHS = 50
BATCH_SIZE = 256

history = model.fit(X_train, y_train,
                        #validation_data=(X_val, y_val),
                        validation_split=0.2,
                        epochs=EPOCHS,
                        batch_size=BATCH_SIZE,
                        callbacks=call_backs,
```

```
                        #class_weight=class_weights,
                        verbose=1)


### Train the XGBoost model
xgb_model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_val, y_val)], ear
```

8.1.9 Evaluation:

Evaluation for Neural network models:

```
# Make predictions on the test set
predictions = model.predict(X_test)
y_pred = np.argmax(predictions, axis=1)
y_true = np.argmax(y_test, axis=1)

loss,accuracy,recall, precission, f1_score, far =
model.evaluate(X_test, y_test, verbose=0)
#CNN +GRU loss,accuracy,recall, precission, f1_score, far =
model.evaluate([X_test,X_test], y_test, verbose=0)
print('loss: ', loss)
print('accuracy: ', accuracy)
print('recall: ', recall)
print('precission: ', precission)
print('f1_score: ', f1_score)
print('far: ',far)
```

Evaluation for XGBoost model:

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
from sklearn.metrics import log_loss, ndcg_score, average_precision_score


y_pred = xgb_model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
auc = roc_auc_score(y_test, y_pred, average='weighted')
y_pred_proba = xgb_model.predict_proba(X_test)
logloss = log_loss(y_test, y_pred_proba)
print("Multi-class log loss:", logloss)
```