King Saud University

College of Computer & Information Sciences

Cybersecurity Master Program

Machine Learning-Based Anomaly Detection for Enhanced Cybersecurity in Wind Farms

SEC 598/599

By

Amani Al-Ghwainm

444203326

Under the supervision of

Dr. Anmar Arif

Submitted in partial fulfillment of the requirements for the Degree of Master in Cybersecurity at the College of Computer and Information Sciences

King Saud University

1445-1446 (2023-2024)

Dedication

This research is dedicated to all the dedicated professionals and researchers working tirelessly to enhance the security and integrity of renewable energy infrastructure. Their commitment and relentless efforts in developing advanced anomaly detection methodologies for wind farms inspire us to strive for a safer and more sustainable future.

Declaration

We hereby declare that we are the sole authors of this report. We authorize King Saud University to lend this report to other institutions or individuals for the purpose of scholarly research.

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, Dr. Anmar Arif, for their invaluable guidance and support throughout this research project. Their expertise and mentorship have been instrumental in shaping this study. I also extend my appreciation to the examinators for their time and consideration in evaluating this work.

Abstract

As wind energy adoption grows, securing wind farms against cyberattacks becomes crucial. This research project proposes a framework for detecting and mitigating cyberattacks in wind farms using data analytics and machine learning. SCADA data from wind turbines is preprocessed and GAMs establish relationships between wind speed, power output, and rotor speed. Four types of cyberattacks are simulated: single parameter manipulation, multiple parameter manipulation, data repetition, and simulated faults. These simulations generate labeled training data for an LSTM neural network model, which is trained to identify anomalous patterns indicating cyberattacks. The framework's performance is evaluated, and insights are derived for refinement. This project contributes to advancing cybersecurity in the renewable energy sector, enhancing the security and resilience of wind farms against evolving cyber threats.

Table of Contents

Table of Figures

List of Tables

List of Abbreviation

CNN	Convolutional Neural Network
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
ML	Machine Learning
SCADA	Supervisory Control and Data Acquisition
RTU	Remote Terminal Unit
GAM	Generalized Additive Model
PCA	Principal Component Analysis
SVM	Support Vector Machines
TN	True Negatives
TP	True Positives
FP	False Positives
FN	False Negatives

1. Introduction

In recent years, the increasing reliance on renewable energy sources, particularly wind power, has brought attention to the critical importance of securing wind turbine systems against potential cyberattacks. Wind farms, which consist of interconnected wind turbines, are vulnerable to malicious actors seeking to disrupt energy production, cause equipment damage, or manipulate data for nefarious purposes. Ensuring the resilience and reliability of wind farms in the face of cyberattacks is crucial for maintaining a stable and sustainable energy supply.

1.1. Problem Statement

Renewable energy sources, such as wind farms, have gained increasing importance in the global energy landscape due to their potential to provide sustainable alternatives to traditional fossil fuels [1]. Wind farms harness the power of wind to generate electricity, contributing to a cleaner and greener energy mix. However, as wind farms have embraced digitalization and connectivity, integrating advanced digital systems and communication networks, they have inadvertently introduced new vulnerabilities to cyber threats [2]. Attackers can exploit vulnerabilities in the SCADA systems, sensors, or communication networks to gain unauthorized access, manipulate data, or disrupt operations. Such attacks can lead to significant consequences, including:

- Compromised energy production and grid stability.
- Physical damage to wind turbines and equipment.
- Financial losses for energy companies and consumers.
- Potential environmental impacts. Reputational damage and loss of public trust.

Detecting and mitigating cyberattacks in wind farms is a complex challenge due to the diverse range of attacks, the distributed nature of wind turbine systems, and the need for real-time monitoring and response.

While the existing body of literature on wind farm anomaly detection has primarily focused on identifying and addressing mechanical failures within wind turbines, there is a critical gap in proactively detecting and mitigating cybersecurity breaches [3][4]. This gap leaves wind farms susceptible to various forms of cyber threats. By failing to address these cybersecurity risks, wind farms are exposed to

potential financial losses resulting from compromised operations, equipment damage, and the potential for prolonged disruptions in energy supply.

To address these challenges, it is imperative to bridge the gap in proactive identification and mitigation of cybersecurity breaches in wind farms. By leveraging machine learning techniques and advanced data analysis, it becomes possible to develop robust anomaly detection systems that can identify and respond to potential cyber threats. By strengthening the cybersecurity posture of wind farms, we can safeguard their economic viability, protect the environment, and contribute to a more resilient and sustainable energy infrastructure.

1.2. Suggested Solution

To address the problem of cyberattacks in wind farms, this project proposes an approach that combines advanced data analytics, machine learning techniques, and domain knowledge of wind turbine operations. The suggested solution utilizes the Kelmarsh wind farm dataset, which contains SCADA data from wind turbines, to develop and validate the proposed framework. The solution involves the following key components:

1. **Data Collection and Preprocessing**: Collect SCADA data from the Kelmarsh wind farm dataset [5][6], remove faulty data points, and select relevant features such as wind speed, power output, and rotor speed.
2. **Data Fitting using GAM**: Establish relationships between wind speed and power output, as well as wind speed and rotor speed, using GAMs to capture the underlying patterns and dependencies within the Kelmarsh wind farm data.
3. **Cyberattack Simulation**: Simulate four types of cyberattacks (single parameter manipulation, multiple parameter manipulation, data repetition, and simulated faults) on the Kelmarsh wind farm data to generate labeled training data for the detection model.
4. **Anomaly Detection using LSTM Model**: Train LSTM neural network model using the preprocessed Kelmarsh wind farm data and simulated attack scenarios to identify anomalous patterns indicative of cyberattacks.

5. **Model Evaluation and Identification of Refinement Areas**: Assess the performance of the trained LSTM model using appropriate evaluation metrics and identify areas for potential refinement based on insights gained from the results, ensuring its effectiveness on the Kelmarsh wind farm dataset.

By leveraging the Kelmarsh wind farm dataset and combining data analytics and machine learning techniques, this solution aims to provide a robust and automated approach for detecting and mitigating cyberattacks in wind farms. The use of real-world data from the Kelmarsh wind farm ensures the practicality and applicability of the proposed framework in enhancing the security and reliability of wind energy systems.

1.3. Research Aims and Objectives

The primary aim of this research project is to develop an effective and scalable framework for detecting and mitigating cyberattacks in wind farms using advanced data analytics and machine learning techniques. The specific objectives of the project are as follows:

1. To collect and preprocess SCADA data from wind turbines, ensuring data quality and selecting relevant features for analysis.
2. To develop GAM that accurately capture the relationships between wind speed, power output, and rotor speed, providing a foundation for anomaly detection.
3. To simulate cyberattack scenarios, including single parameter manipulation, multiple parameter manipulation, data repetition, and simulated faults, to generate labeled training data for the detection model.
4. To design, train, and evaluate an LSTM neural network model for identifying cyberattacks based on the preprocessed data and simulated attack scenarios.
5. To assess the performance of the proposed framework using appropriate evaluation metrics and validate its effectiveness in detecting and mitigating cyberattacks in wind farms.

6. To provide insights and recommendations for enhancing the security and resilience of wind energy systems based on the findings of the research project.

By achieving these objectives, this research project aims to contribute to the advancement of cybersecurity in the renewable energy sector, specifically in the context of wind farms. The developed framework and insights gained from this study can serve as a foundation for further research and practical implementations in securing wind energy systems against evolving cyber threats.

1.4. Report Outline

The remainder of this report is organized as follows: Chapter 2 provides background information on wind energy and SCADA systems, data fitting, and LSTM. In Chapter 3, a comprehensive literature review is conducted, examining the existing research and related work in the field of cybersecurity for wind farms. The review highlights th1qe current gaps and opportunities for further research. Chapter 4 presents a detailed description of the system design and implementation of the proposed framework, including the data preprocessing, GAM fitting, cyberattack simulation, and LSTM model development. The evaluation methodology used to assess the performance of the proposed framework is outlined in Chapter 5, discussing the metrics and experimental setup employed. In Chapter 6, the results obtained from the experiments are presented, followed by a discussion of the findings, insights, and implications of the research project. Finally, Chapter 7 concludes the report by summarizing the key contributions, limitations, and future directions for research in the field of cybersecurity for wind farms.

2. Background

This chapter provides an overview of the key concepts and technologies relevant to the research project. It covers wind turbines and their associated SCADA systems, the Generalized Additive Model, machine learning methods for anomaly detection, and Long Short-Term Memory neural networks.

2.1. Wind Turbines and SCADA System

Wind turbines are the primary components of wind farms, converting kinetic energy from the wind into electrical energy. Modern wind turbines are equipped with sophisticated control systems, including SCADA systems, which play a crucial role in monitoring, controlling, and optimizing the performance of wind turbines [7]. SCADA systems collect and process real-time data from various sensors and devices installed on wind turbines. The data provided by SCADA systems typically includes:

- Wind speed and direction.
- Power output.
- Rotor speed and blade pitch angle.
- Generator temperature and vibration.
- Gearbox and bearing condition.
- Electrical parameters (voltage, current, frequency).

This data is essential for monitoring the health and performance of wind turbines, detecting faults, and enabling predictive maintenance [8]. However, the reliance on SCADA systems also exposes wind farms to potential cyberattacks. Malicious actors can manipulate SCADA data to mislead operators, cause equipment damage, or disrupt energy production. Manipulating SCADA data can have severe consequences, such as:

- False alarms and unnecessary shutdowns.
- Overloading or underloading of wind turbines.
- Inaccurate forecasting and grid instability.
- Accelerated wear and tear of components.
- Safety risks for personnel and the environment.

Therefore, ensuring the integrity and security of SCADA data is crucial for the reliable and safe operation of wind farms [9].

2.2. Generalized Additive Model (GAM)

GAMs are a flexible class of statistical models that extend traditional linear models by allowing for non-linear relationships between the response variable and predictor variables [10]. GAMs are particularly useful for modeling complex, non-

linear relationships in data, making them suitable for various applications, including wind energy analysis [10].

In the context of wind turbine modeling [11], GAMs can be used to establish relationships between wind speed, power output, and rotor speed. By capturing the underlying patterns and dependencies in the data, GAMs provide a foundation for anomaly detection and condition monitoring. GAMs offer several advantages, including:

- Ability to model non-linear relationships.
- Interpretability of the model components.
- Flexibility in specifying the smoothing functions.
- Robustness to outliers and missing data.

The use of GAMs in wind turbine modeling enables accurate characterization of the expected behavior of wind turbines under normal operating conditions.

2.3. Machine Learning Methods for Anomaly Detection

Machine learning methods have gained significant attention for their ability to detect anomalies and abnormal patterns in data. Anomaly detection involves identifying instances that deviate significantly from the expected or normal behavior [12]. In the context of wind farms, anomaly detection can help identify potential cyberattacks, equipment faults, or unusual operating conditions. Various machine learning techniques have been applied for anomaly detection, including:

- Unsupervised learning methods, such as clustering and PCA [13].
- Supervised learning methods, such as support SVM and decision trees [14].
- Deep learning methods, such as LSTM and CNN [15].

These methods leverage the vast amounts of data generated by SCADA systems to learn the normal behavior of wind turbines and detect deviations from it.

2.4. LSTM Network

LSTM networks are a specialized type of recurrent neural network that excel at modeling and analyzing sequential data. They overcome the limitations of traditional feedforward neural networks by effectively capturing long-term dependencies and

addressing the vanishing gradient problem [16]. Figure 2.1 provides an overview of the structure of an LSTM cell, the fundamental building block of an LSTM network. It consists of three main components: the input gate, the forget gate, and the output gate. These gates control the flow of information within the cell and play a pivotal role in determining what information to retain and what to discard.

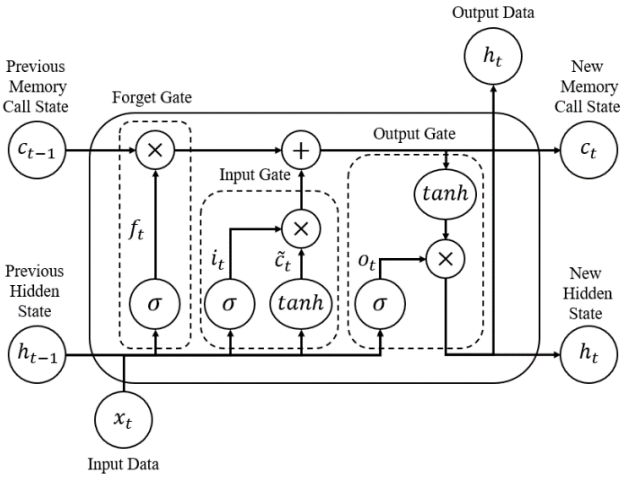


Figure 2.1 LSTM Cell Architecture

The forget gate in an LSTM network plays a crucial role in determining what information to retain or discard from the previous cell state [17]. It takes the input vector and the previous hidden state as inputs and utilizes a sigmoid activation function to produce a selector vector [18]. This selector vector acts as a gatekeeper, with elements close to 0 indicating information to forget and elements close to 1 indicating information to retain [19]. The forget gate's equation, typically represented in Equation (2.1), governs this selective memory process within the LSTM network [20]. By incorporating this mechanism, LSTM models can effectively manage and update information over time, enhancing their ability to learn and retain long-term dependencies in sequential data.

$$f_t = \sigma\left(w_f[h_{t-1}, x_t] + b_f\right) \tag{2.1}$$

Where σ is the sigmoid function, w_f is the weight matrix for the forget gate, $[h_{t-1}, x_t]$ is the concatenation of previous hidden state and current input, b_f is the bias vector for the forget gate and f_t is the vector of forget gate values for the current time step.

The input gate and candidate memory collaborate to determine new information for the cell state. They take the input vector and the previous hidden state as inputs. The input gate uses a sigmoid activation function to generate a selector vector, which determines the importance of different elements in the candidate memory [21]. The candidate memory, a separate neural network, takes the input vector and previous hidden state as inputs and produces a candidate vector using a hyperbolic tangent function. The selector vector and candidate vector are multiplied elementwise, incorporating selected information into the cell state [22]. This gating and candidate memory interaction is crucial for determining new information added to the cell state, reflecting a fundamental aspect of memory formation and retention in neural networks. The equations for the input gate are shown in Equation (2.2) and (2.3).

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{2.2}$$

$$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c) \tag{2.3}$$

Where σ is the sigmoid function, w_i and w_c are the weight matrices for the input gate, $[h_{t-1}, x_t]$ is the concatenation of previous hidden state and current input, b_i and b_c are the bias vector for the input gate, i_t is the vector of input gate values for the current time step and $\tilde{c}_t$ is the candidate cell state vector for the current time step.

The LSTM output gate determines the relevant information to output as the hidden state. It takes the input vector and the previous hidden state as inputs. Like the other gates, the output gate uses a sigmoid activation function to generate a selector vector. This selector vector determines the importance of different elements in the cell state for generating the hidden state [23]. The cell state is passed through a hyperbolic tangent function. The selector vector and the candidate vector are multiplied elementwise, producing the hidden state [24]. The hidden state is then passed to the next time step, allowing the LSTM network to process sequential data effectively [17]. The equations for the output gate are shown in Equation (2.4), (2.5), and (2.6).

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{2.4}$$

$$h_t = o_t \cdot tanh(c_t) \tag{2.5}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{2.6}$$

Where σ is the sigmoid function, w_o is the weight matrix for the output gate, $[h_{t-1}, x_t]$ is the concatenation of previous hidden state and current input, b_0 is the bias

vector for the output gate, o_t is the vector of output gate values for the current time step, and h_t is the current hidden state.

3. Literature Review

3.1. Cybersecurity Attacks and Failures in Wind Farms

The escalating threat of cyberattacks on wind energy systems has become a concerning reality, as evidenced by numerous incidents that highlight the vulnerabilities and potential consequences for the operation and security of wind farms [1]. The nature of these attacks, whether intentional or unintentional, emphasizes the imperative for effective anomaly detection methodologies to promptly identify and mitigate cyber threats within the complex digital ecosystems of wind energy installations [1].

Yan et al. [2] showed that cyberattacks can cause major problems for the electric grid, including economic losses and equipment damage. Wind farms generate vast amounts of data through SCADA systems, encompassing sensor data, equipment statuses, and performance metrics. The sheer volume and intricacy of this data require sophisticated techniques to extract valuable insights and detect deviations from expected patterns [25]. Anomaly detection plays a crucial role in handling this data. Through continuous monitoring and analysis, anomaly detection methods can identify unusual activities that could signify operational abnormalities, potential security breaches, or malicious intrusions [26].

Although the existing research contribute valuable insights in the field of anomaly detection, they do not explicitly address the direct impact of cybersecurity attacks on failures within wind farms. They primarily emphasize the development of innovative anomaly detection methods for detecting faults in wind turbines, such as gearboxes, generators, and wind speed sensors. For instance, Zeng et al. [3] developed an anomaly detection method based on a combined probability estimation model and hypothesis testing, for detecting faults gearboxes and generators of wind turbines.

Li et al. [27] introduced a novel Deep Small-World Neural Network utilizing unsupervised learning for early wind turbine failure detection, combining deep learning networks and an adaptive threshold approach. Li et al. [28] designed an anomaly detection method for wind speed sensors by employing dependency

modeling and cross-verification techniques. Zeng et al. [29] presents an enhanced wind turbine anomaly detection approach that leverages shared operational conditions among turbines within the same wind farm. Utilizing a similarity evaluation method and LSTM based estimation models, the method improves accuracy by collectively estimating monitored variables and employs a combined anomaly detection technique.

Xiang et al. [15] proposed a method for wind turbine fault detection using a combination of deep learning techniques, including CNN and LSTM, with an attention mechanism (AM). The method utilized SCADA data as input variables, employing a CNN architecture to capture dynamic data changes. The AM enhanced the importance of key information within the LSTM, improving learning accuracy. The model offered early anomaly warnings and predicted faulty components, demonstrating its effectiveness in predicting wind turbine failures.

However, existing works lack a detailed examination of the specific consequences of cyber threats on the overall integrity and operation of wind farms. This creates a gap in the literature, as the intersection between cybersecurity attacks and their potential impact on failures within wind farms remains poorly understood. Bridging this gap is crucial to develop comprehensive anomaly detection techniques that address both physical failures and cybersecurity attacks. Understanding the vulnerabilities and potential consequences of cybersecurity attacks in wind farms is vital for devising effective mitigation strategies and ensuring the reliable operation of these energy systems in the face of evolving threats.

3.2. ML-Based Cyber-Attack Detection in Power Systems

Several research studies have explored the application of machine learning techniques for anomaly detection in power system. These studies collectively strive to develop effective detection models and algorithms capable of accurately identifying, classifying, and mitigating cyber-attacks, thereby bolstering the security and resilience of power systems. Table 3.1 provides a comparison of the methodologies and key findings related to the detection of cyber-attacks using machine learning across various systems.

Table 3.1. Detection Using Machine Learning Comparison

Reference	System	Features	Attack Detection	ML method	Accuracy
[30]	Power System	Not Specified	cyber-attacks.	Restricted Boltzmann Machines-based algorithm	97.8%
[31]	Energy Systems	VCA4, VCA1, SI, SV, VCM1, and VCM2	cyber and non-cyber-attacks.	AdaBoost with Random Forest as the basic classifier	93.91%
[32]	Smart Grids	Not Specified	cyber-attacks.	Random Forest, Logistic Regression, K-Nearest Neighbor	90.56% (Random Forest)
[33]	Wind Turbines	Sensor Signals and Generation Temperature	non-cyber-attacks.	Artificial Neural Networks	99.8%
[34]	Wind Turbines	Ice Generation Related Features	non-cyber-attacks.	Deep Support Vector Data Description	91.45%
[35]	Smart Grid Networks	Not specified	cyber-attacks.	Hybrid MLP Sequential-Feedforward Neural Network	99.59%
[36]	Power Electronics-Dominated Grids	Power Electronics Failures and Anomalies Related Features	cyber and non-cyber-attacks.	LSTM-based recurrent neural networks with model predictive control	96.66%

Diaba et al. [30] proposes a Restricted Boltzmann Machine-based nature-inspired artificial root foraging optimization algorithm for identifying and classifying cyber-attacks in power systems. The proposed algorithm demonstrates superior performance compared to other deep learning algorithms such as Artificial Neural Networks, Convolutional Neural Networks, and Support Vector Machines. It achieves high accuracy rates for binary, three-class, and multi-class classification, showcasing the effectiveness of deep learning algorithms and metaheuristic optimization in bolstering cybersecurity in power systems.

In Almalaq at el. study [31], an attack detection model for energy systems based on deep learning is proposed. The model utilizes data and logs gathered through phasor measurement units and incorporates property or specification making to generate features. Various machine learning methods are employed, with random forest as the basic classifier of AdaBoost. The study highlights the vulnerability of supervisory control and data acquisition SCADA systems to cyber-attacks and emphasizes the importance of research in cyber-physical systems and the utilization of deep machine learning for intrusion detection.

Borges Hink at el. [32] proposes a machine learning-based attack detection model for smart grids. The model utilizes data and logs collected from phasor measuring devices to learn system behaviors and identify potential security boundaries. Random Forest, Logistic Regression, and K-Nearest Neighbor models are built and assessed using various performance metrics. Random Forest achieves the highest performance, with a 90.56% accuracy in detecting power system disturbances, showcasing its potential to aid operators in power system security decision-making processes.

In Amini et al. study [33], AI and ML techniques are applied for diagnosing and monitoring wind turbines using SCADA data. Eight different ANN models are evaluated for predicting system failure based on sensor signals and generation temperature. The developed ML model demonstrates an impressive 99.8% accuracy in predicting wind turbine generator temperature, enabling early fault detection and prevention.

Peng et al. [34] proposes a deep learning-based anomaly detection method, Deep Support Vector Data Description, specifically targeting ice detection on wind

turbine blades. By combining a CNN with the Support Vector Data Description detector, the method achieves a successful detection rate of 91.45% for ice formation, addressing the limitations of traditional fault detection methods and highlighting the potential of deep learning in enhancing the accuracy of wind turbine condition monitoring.

Aribisala et al [35] introduces a hybrid Multilayer Perceptron Sequential-Feedforward Neural Network model for intrusion detection and classification in smart grid networks. The model combines Deep Learning algorithms with Network Intrusion Detection Systems and Host-based Intrusion Detection Systems to accurately predict attack vectors, achieving high accuracy and showcasing the potential of using ANN and Sequential-Feedforward Neural Network model to enhance intrusion detection and classification in smart grid networks.

In Baker et al. study [36], real-time anomaly detection and classification in power electronics-dominated grids are addressed using RNN with LSTM. The proposed approach integrates LSTM networks with model predictive control to achieve real-time anomaly detection and classification. The research emphasizes the significance of distinguishing between internal failures in power electronics and potential cyber-attacks for resilient operation.

These studies collectively contribute to the development of comprehensive and effective ML-based approaches for cyber-attack detection and mitigation, improving the security and reliability of critical energy systems. Continued research in this field is essential to stay ahead of evolving cyber threats and ensure the robust operation of power systems and wind farms.

3.3. Simulation Attacks for Assessing Power Systems Vulnerabilities

In addition to anomaly detection, simulation attacks play a significant role in assessing vulnerabilities, developing mitigation strategies, and bolstering the resilience of critical systems. Simulation attacks involve creating controlled environments to mimic cyberattacks and evaluate the response of wind energy systems under different attack scenarios. Table 3.2 provides a comparison of the cyber-attack simulation approaches employed in the different studies.

Table 3.2. Cyberattack Simulation Approach Comparison

Reference	System	Data	Attack Type	Simulation Method
[37]	Industrial Power Plants	Real Time SCADA Data	Data Manipulation Attack	Remote Data Modification
[38]	Cyber-Physical Systems	Chicago Load/Weather Dataset	False Data Injection Attacks	RTU Hardware and Software Simulation
[39]	Smart Grids	Smart Grid Dataset	False Data Injection Attacks	Synthetic Injection of False Data
[40]	Power Grids	Power Grid Dataset	Various Cyber-Attacks	PowerWorld and PSCAD/EMTDC Simulations
[41]	Power Systems	Power System Dataset	Various Cyber-Attacks	Dataset Creation and Acquisition Engine Tool.

Marilena et al [37] focused on simulating and analyzing cyber-attacks in an industrial power plant, particularly targeting the SCADA system. The study identified vulnerabilities in human-machine interfaces, electrical equipment, and SCADA systems through simulations of cyber-attacks. Similarly, Ali et al [38] proposed a laboratory set-up to simulate cyber-attacks and detect false data injection attacks in a cyber-physical system. The researchers developed a framework that involved simulating different attack scenarios, including false data injection attacks, to evaluate the effectiveness of supervised learning algorithms and unsupervised scoring algorithms in detecting these attacks.

The detection of false data injection attacks in smart grids was addressed by Jin et al. [39]. The researchers simulated false data injection attacks on the IEEE-14 node test system and evaluated the effectiveness of the unscented Kalman filter state estimation method in detecting and mitigating these attacks. While Kaushik et al. [40] focused on simulating cyber-attacks on the grid and assessing the dynamic security of the system. Simulations were conducted using tools like PowerWorld and

PSCAD/EMTDC to study the impact of cyber-attacks on the grid's stability and response.

In addition, Korving and Vaarandi [41] introduced the Dataset Creation and Acquisition Engine, which provided a configurable testbed for generating datasets related to attack scenarios. By automating the data collection pipeline and leveraging virtualization technologies, Dataset Creation and Acquisition enabled the reproducible simulation of attack scenarios for dataset generation. Simulation of cyber-attacks was also used in the development of machine learning-based models for attack detection.

4. System Design and Implementation

This chapter presents the detailed design and implementation of the proposed framework for detecting cyberattacks in wind farms. The system consists of four main components: data processing, data fitting using GAMs, cyberattack simulation, and the design of the LSTM model for anomaly detection.

4.1. Data Preprocessing

The first step in the system design is to process the raw SCADA data obtained from the Kelmarsh wind farm dataset [5]. Kelmarsh Wind Farm is an onshore wind farm located near Haselbach, Northamptonshire, England. It consists of six 2.05MW Senvion MM92 turbines, with a total installed capacity of 12.3MW. The dataset consists of two main components stored in CSV files: status data and turbine data. The status data file provides information about the operational state of each individual turbine within the wind farm. The status file includes information on wind turbine faults and maintenance events. The turbine data file, on the other hand, contains more detailed information about each specific turbine within the wind farm. It contains 299 features include wind speed, direction, power production, and temperature measurements, and other relevant metrics. These features provide comprehensive insights into the performance and environmental conditions surrounding the turbine. The data processing stage involves data cleaning and feature selection.

4.1.1. Data Cleaning

The raw SCADA data may contain missing values, data points corresponding to actual wind turbine faults, and periods when the wind turbine is off. To ensure data

quality, a targeted data cleaning process is performed. Rows with missing values are identified and removed from the dataset. Additionally, data points associated with known wind turbine faults and off periods are excluded from further analysis, as they do not represent normal operating conditions. These data points are identified in the turbine data by using the status files. Figure 4.1 and Figure 4.2 show labelled data of the power and rotor speed in relation with the wind speed, respectively.

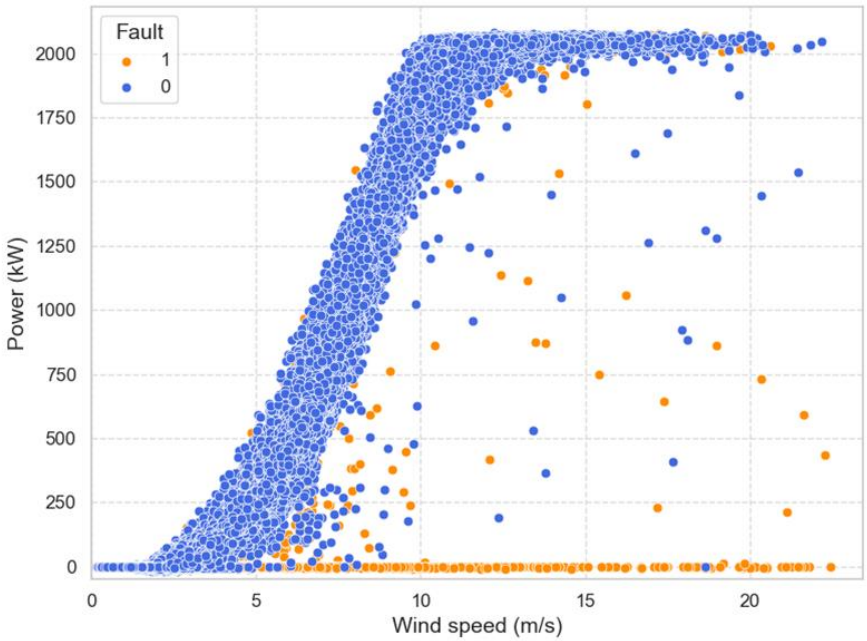


Figure 4.1. Labelled Power Data in Relation to Wind Speed

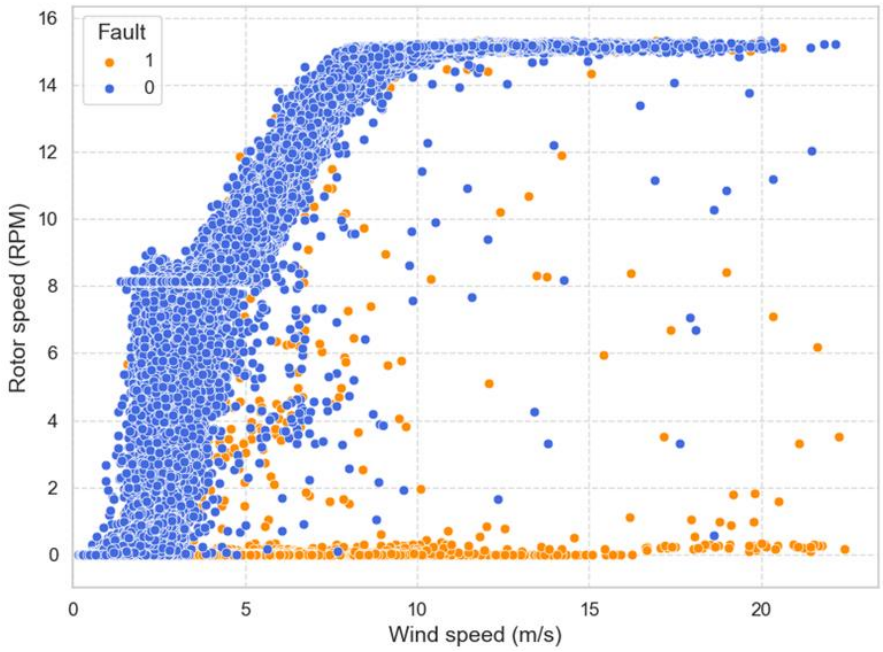


Figure 4.2. Labelled Rotor Speed Data in Relation to Wind Speed

4.1.2. Feature Selection

The SCADA system provides a wide range of measurements and parameters related to wind turbine operation. However, not all features are equally relevant for detecting cyberattacks. In this study, the selected features include wind speed, power output, and rotor speed, as they are directly related to the performance and behavior of wind turbines. These features are extracted from the cleaned SCADA data for further analysis.

4.2. Data Fitting Using GAMs

After preprocessing the SCADA data, the next step is to establish relationships between wind speed and power output, as well as wind speed and rotor speed, using GAMs. To model the relationship between wind speed and power output, a GAM is fitted using the cleaned SCADA data. The power output is treated as the response variable, and wind speed is considered as the predictor variable. Similarly, a GAM is fitted to model the relationship between wind speed and rotor speed. Figure 4.3 shows the fitted data for the power and rotor speed with the wind speed using GAM.

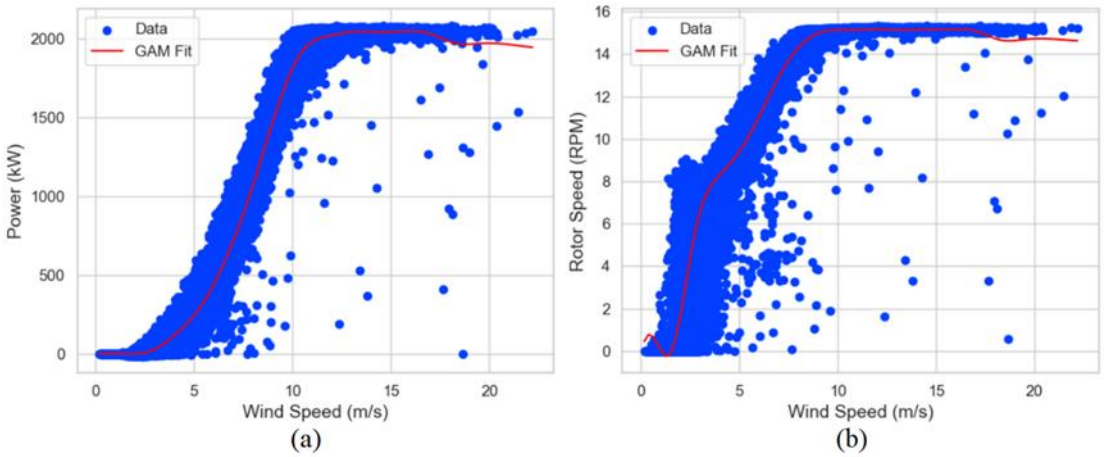


Figure 4.3. Power Output and Rotor Speed with Fitted GAM Curves

Figure 4.4 displays a histogram that represents the distribution of the absolute difference between the actual and predicted power outputs and rotor speed outputs. The x-axis of the histogram represents the range of the absolute differences, while the y-axis represents the frequency or count of occurrences for each bin. The bins divide the range of absolute differences into equal intervals, with the number of bins set to 20.

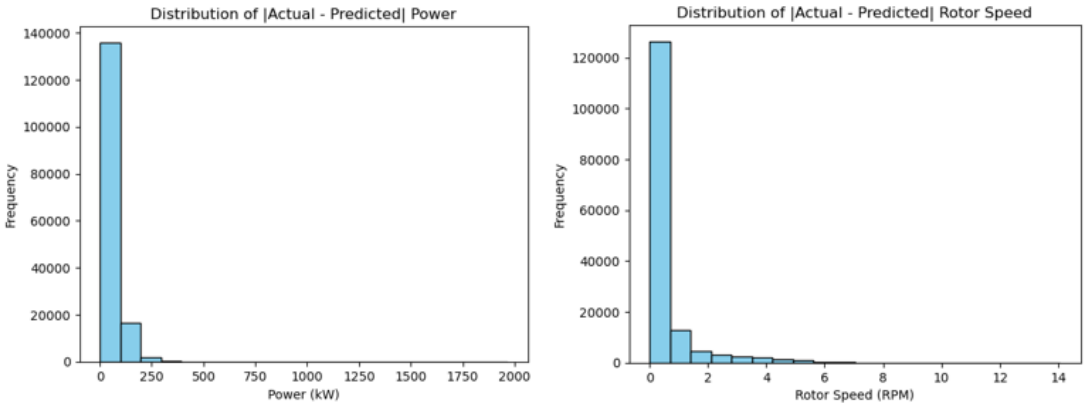


Figure 4.4. Comparative Analysis of Prediction Deviations

4.3. Cyberattack Simulation

To generate labeled training data for the cyberattack detection model, cyberattacks are simulated on the preprocessed SCADA data. Four types of cyberattacks are considered: single parameter manipulation, multiple parameter manipulation, data repetition, and simulated faults.

4.3.1. Single Parameter Manipulation

In this type of cyberattack, a single parameter (wind speed, power output, or rotor speed) is manipulated by modifying its values based on a predefined attack pattern. The manipulation is performed using a truncated normal distribution with a mean error of 70% and a range between 50%-90%. The attack is applied to a randomly selected subset of the data points, simulating a persistent sensor error or data manipulation. An example is shown in Figure 4.5.

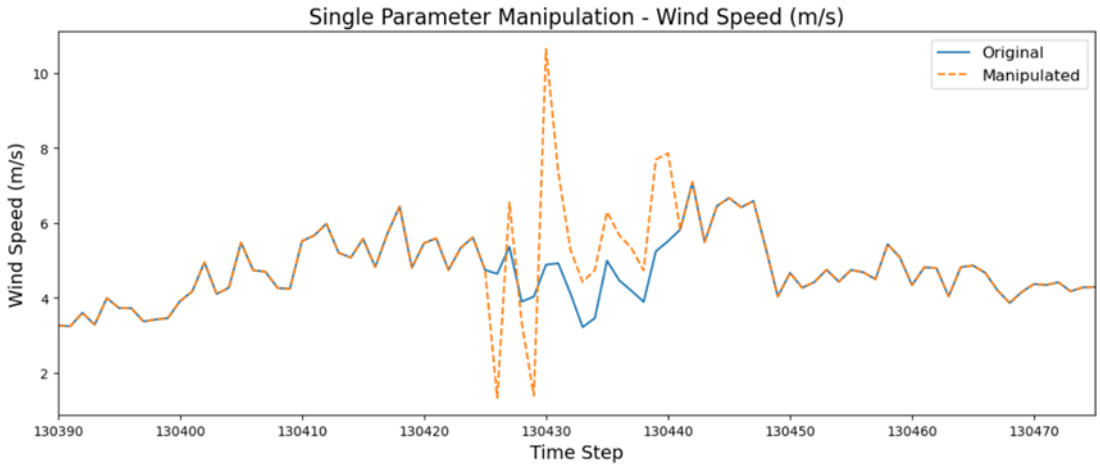


Figure 4.5. Single Parameter Manipulation Attack on Wind Speed

4.3.2. Multiple Parameter Manipulation

Multiple parameter manipulation involves the simultaneous alteration of wind speed, power output, and rotor speed values. The manipulation is performed using a similar truncated normal distribution with a mean error of 70%. This type of cyberattack represents a coordinated effort to disrupt the operation of wind turbines by tampering with multiple sensor readings. An example is shown in Figure 4.6 and 4.7.

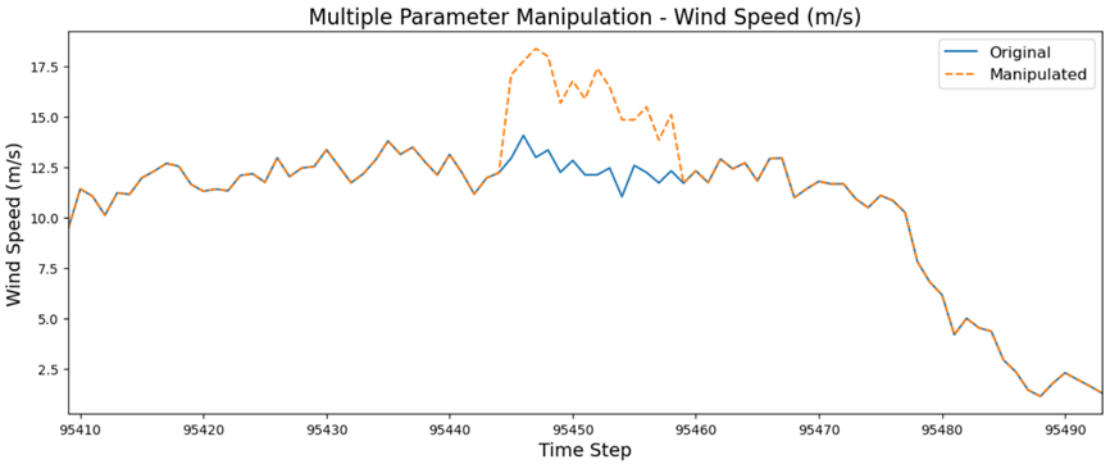


Figure 4.6. Multiple Parameter Manipulation Attack on Wind Speed

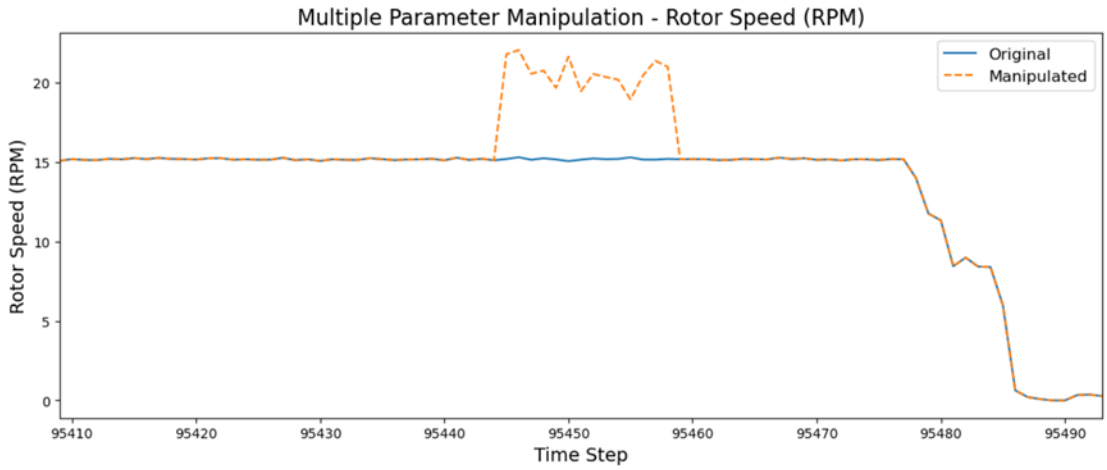


Figure 4.7. Multiple Parameter Manipulation Attack on Rotor Speed

4.3.3. Data Repetition

Data repetition attacks involve replacing a portion of the SCADA data with previously recorded values from a different time periods. This type of attack simulates a scenario where the attacker gains access to the data storage system and replaces

current measurements with historical data, making it difficult to detect the anomaly based on individual data points alone. An example is shown in Figure 4.8.

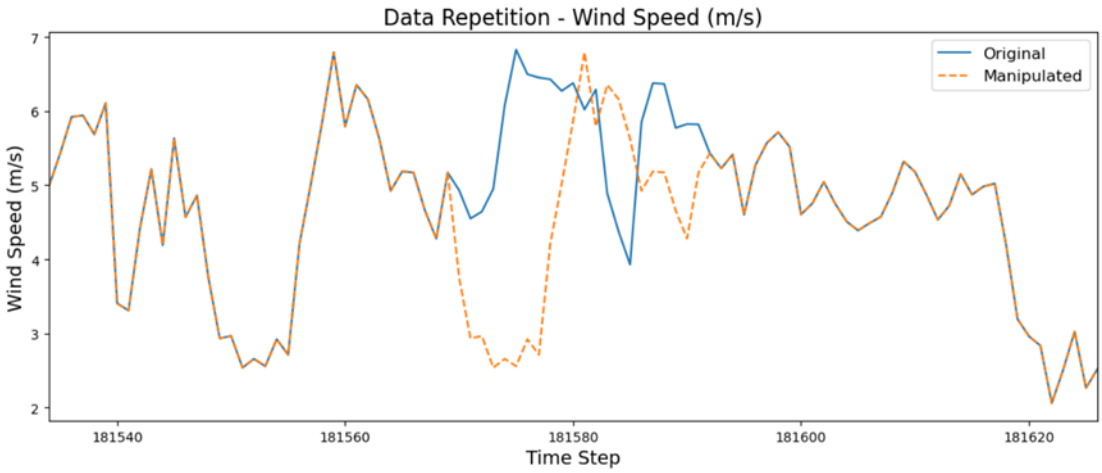


Figure 4.8. Data Repetition Attack on Wind Speed

4.3.4. Simulated Faults

Simulated faults are introduced by setting specific parameters, such as wind speed, power output, or rotor speed, to zero for a certain duration. These faults mimic sensor malfunctions or communication failures that can impact the operation of wind turbines. The duration and frequency of simulated faults are randomly determined to create a diverse set of anomalous scenarios. An example is shown in Figure 4.9.

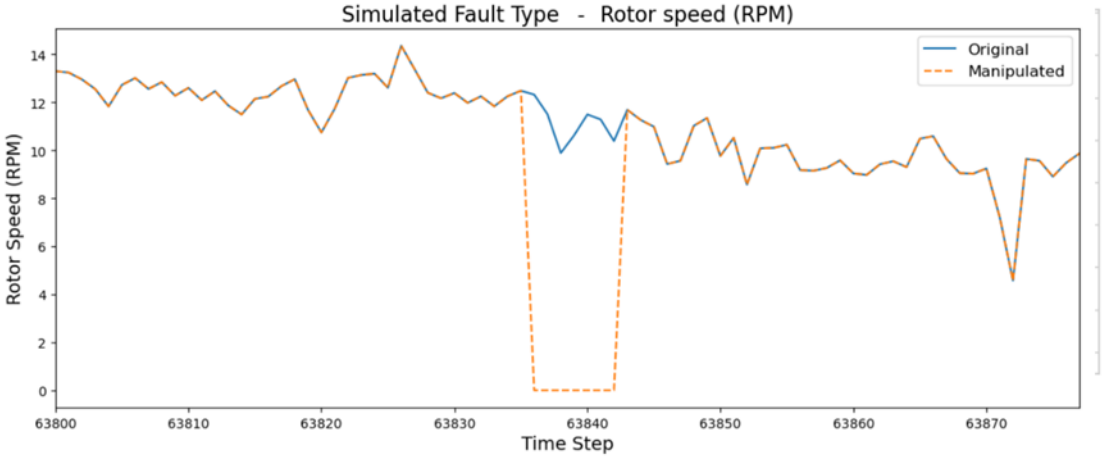


Figure 4.9. Simulated Fault Attack on Rotor Speed

4.4. LSTM Model Design and Implementation

The Long Short-Term Memory neural network is chosen as the cyberattack detection model due to its ability to capture temporal dependencies and learn complex patterns in time series data [15]. The LSTM model is designed and implemented using the Keras deep learning library with TensorFlow as the backend.

4.4.1. Data Preparation and Normalization

The preprocessed SCADA data, along with the simulated cyberattack scenarios, is split into training and testing sets. The training set is used to train the LSTM model, while the testing set is used to evaluate its performance. Before feeding the data into the LSTM model, normalization is applied to scale the features to a common range, typically between 0 and 1, or standardize them to have zero mean and unit variance [16]. This step ensures the compatibility and comparability of different features and prevents certain features from dominating others due to differences in scales.

4.4.2. LSTM Architecture

The LSTM model architecture consists of three LSTM layers followed by a dense layer for classification. The input to the model is a sequence of data points, where each sequence has a length of 10-time steps. Each time step contains three features:

1. Wind speed reading: The actual wind speed measured by the SCADA system at each time step.
2. Power difference: The difference between the actual power reading and the power estimated using GAM at each time step. This difference is calculated by subtracting the GAM-estimated power from the actual power reading.
3. Rotor speed difference: The difference between the actual rotor speed reading and the rotor speed estimated using GAM at each time step. This difference is calculated by subtracting the GAM-estimated rotor speed from the actual rotor speed reading.

These input features are selected to capture the relevant information for anomaly detection. The wind speed reading provides the actual environmental condition at each time step. The power difference and rotor speed difference features represent the deviations of the actual readings from the expected values estimated by

the GAMs. These differences can indicate potential anomalies or cyberattacks if they deviate significantly from the normal behavior captured by the GAMs.

The first LSTM layer has 128 units and receives the input sequence. It returns the full sequence of outputs, which is then passed to the second LSTM layer. The second LSTM layer has 64 units and also returns the full sequence of outputs. The third LSTM layer has 32 units and returns only the last output. Dropout regularization with a rate of 0.2 is applied after each LSTM layer to prevent overfitting [16].

The output from the last LSTM layer is then passed to a dense layer with a single unit and a sigmoid activation function. The sigmoid activation function squashes the output between 0 and 1, representing the probability of an anomaly at each time step. The architecture of the LSTM model can be summarized as follows:

- Input layer: Sequence of 10-time steps, each containing the three input features (wind speed reading, power difference, rotor speed difference).
- LSTM layer 1: 128 units, returns full sequence of outputs.
- Dropout layer 1: Dropout rate of 0.2.
- LSTM layer 2: 64 units, returns full sequence of outputs.
- Dropout layer 2: Dropout rate of 0.2.
- LSTM layer 3: 32 units, returns last output.
- Dropout layer 3: Dropout rate of 0.2.
- Dense layer: 1 unit with sigmoid activation function.

By using the wind speed reading, power difference, and rotor speed difference as inputs, the LSTM model can learn the temporal patterns and dependencies in the data, considering both the actual measurements and the deviations from the expected values estimated by the GAMs. This approach allows the model to capture the normal behavior of the wind turbine system and identify anomalies or cyberattacks based on the learned patterns.

4.4.3. Model Training

The LSTM model is trained using the prepared and normalized training data. The Adam optimizer [26] is used with binary cross-entropy as the loss function, as the task is to classify each time step as either normal or anomalous. The model is trained for 30 epochs with a batch size of 32. Early stopping is implemented to monitor the

validation loss and stop training if the loss does not improve for 5 consecutive epochs. The training data is split, with 20% used for validation during training.

4.4.4. Anomaly Detection

Once the LSTM model is trained, it can be used for anomaly detection on new, unseen SCADA data. The preprocessed and normalized data is reshaped to include the sequence length of 10-time steps. The trained model predicts the probability of each time step being anomalous. A threshold is applied to the predicted probabilities to classify each time step as either normal or anomalous.

By leveraging the power of LSTMs and the specified architecture, the proposed anomaly detection model can effectively capture the temporal dependencies and learn the normal behavior of the wind turbine system, enabling the identification of anomalous patterns indicative of cyberattacks.

4.5. Implementation Details

The proposed system is implemented using Python programming language and popular libraries such as NumPy, Pandas, and Scikit-learn for data processing and machine learning tasks. The GAMs are fitted using the PyGAM library, which provides a user-friendly interface for specifying and fitting GAMs. The LSTM model is implemented using the Keras deep learning library with TensorFlow as the backend, allowing for efficient training and deployment of the model.

The code is organized into modular functions and classes, promoting code reusability and maintainability. Proper documentation and comments are provided to enhance code readability and facilitate future updates or modifications.

In summary, this chapter presents the detailed design and implementation of the proposed system for detecting cyberattacks in wind farms. The system incorporates data processing techniques, GAMs for data fitting, cyberattack simulation, and an LSTM model for anomaly detection. The implementation details, including the choice of programming language, libraries, and evaluation metrics, are discussed. The modular and scalable design of the system ensures its applicability to real-world scenarios and facilitates future enhancements and extensions.

5. Evaluation Methodology

This chapter presents the evaluation methodology used to assess the effectiveness and performance of the proposed anomaly detection framework for identifying cyberattacks in wind turbine systems. The evaluation aims to answer key research questions, compare different input representations and machine learning models, and utilize appropriate evaluation metrics to measure the system's performance.

5.1. Research Questions

The evaluation methodology is designed to address the following research questions:

1. How effective is the proposed anomaly detection framework in identifying cyberattacks in wind turbine systems?
2. Does the use of GAM for input preprocessing improve the performance of the anomaly detection models compared to using raw input features?
3. How does the performance of the LSTM model compare to other methods like Isolation Forest model for anomaly detection in wind turbine systems?

5.2. Materials List

Hardware:

- High-performance computers for data processing and model training.
- Networking equipment to establish connectivity.

Software:

- Programming languages and open-source machine learning libraries (e.g., scikit-learn, TensorFlow, PyTorch).
- Data processing and analysis tools (e.g., pandas, NumPy, and SciPy).
- Machine learning environment (e.g., Jupyter Notebook).
- Data visualization frameworks (e.g., Matplotlib, Plotly, Tableau).

Dataset:

- The dataset will serve as the foundation for training and evaluating the machine learning models for anomaly detection in wind farms (e.g., Kelmarsh dataset [5]).

5.3. Procedure

The evaluation procedure consists of the following steps:

1. The raw SCADA data is preprocessed as described in Section 4.1, including data cleaning and feature selection.
2. Two sets of input features are prepared:
    - Raw input features: wind speed, power, and rotor speed measurements.
    - GAM-based input features: wind speed measurement, power difference (actual power - GAM-estimated power), and rotor speed difference (actual rotor speed - GAM-estimated rotor speed).
3. Model Training and Testing:
    - The preprocessed data is split into training and testing sets.
    - Four anomaly detection models are trained and evaluated: LSTM model with raw input features.
    - LSTM model with GAM-based input features.
    - Isolation Forest model with raw input features.
    - Isolation Forest model with GAM-based input features.
    - Each model is trained on the training set and evaluated on the testing set.
4. The performance of each model is evaluated using the following metrics:
    - Accuracy: The proportion of correct predictions among the total number of instances. Accuracy = (True Positives + True Negatives) / Total Instances.
    - Precision: The proportion of true positive predictions among all positive predictions made by the model. Precision = True Positives / (True Positives + False Positives).
    - Recall: The proportion of true positive predictions among all actual positive instances in the dataset. Recall = True Positives / (True Positives + False Negatives).
    - F1-score: The harmonic mean of precision and recall, providing a balanced measure of the model's performance. F1-score = 2 * (Precision * Recall) / (Precision + Recall)

These evaluation metrics provide a comprehensive assessment of the models' performance, considering different aspects such as overall correctness, ability to

identify anomalies, and the balance between precision and recall. The evaluation metrics are calculated for each model based on the predictions made on the testing set.

The performance of the LSTM model and Isolation Forest model is compared for both raw input features and GAM-based input features. The results are presented and discussed in the next chapter.

6. Results and Discussions

This section presents the findings related to the effectiveness of the proposed anomaly detection framework in identifying cyberattacks in wind turbine systems, the impact of using GAM for input preprocessing, and the comparison of the performance of the LSTM model with other methods like the Isolation Forest model for anomaly detection.

6.1. Results

Four anomaly detection models were evaluated in this study: the LSTM model with raw input features, the LSTM model with GAM-based input features, the Isolation Forest model with raw input features, and the Isolation Forest model with GAM-based input features. The performance of these models was assessed using various metrics including accuracy, precision, recall, and F1-score. These metrics provide valuable insights into the models' ability to accurately predict and capture anomalies within the dataset as shown in Table 5.

The evaluation was conducted using a dataset consisting of a training set with a size of 119,957 instances, including 895 attacks, and a testing set with a size of 20,194 instances, including 136 attacks.

Table 5.1. Performance Comparison of Anomaly Detection Models

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
LSTM	0.993	1.000	0.000	0.000
LSTM + GAM	0.996	0.787	0.654	0.714
Isolation Forest	0.869	0.043	0.867	0.081
Isolation Forest + GAM	0.854	0.039	0.889	0.074

The LSTM model with raw input features achieved an accuracy 0.993, a precision of 1.00, but it had a recall and F1-score of 0.00. Interestingly, this model did not identify any instances as attacks, resulting in zero true positives, correctly detected attacks, and zero false positives, normal instances incorrectly flagged as attacks as shown in the Figure 6.1.

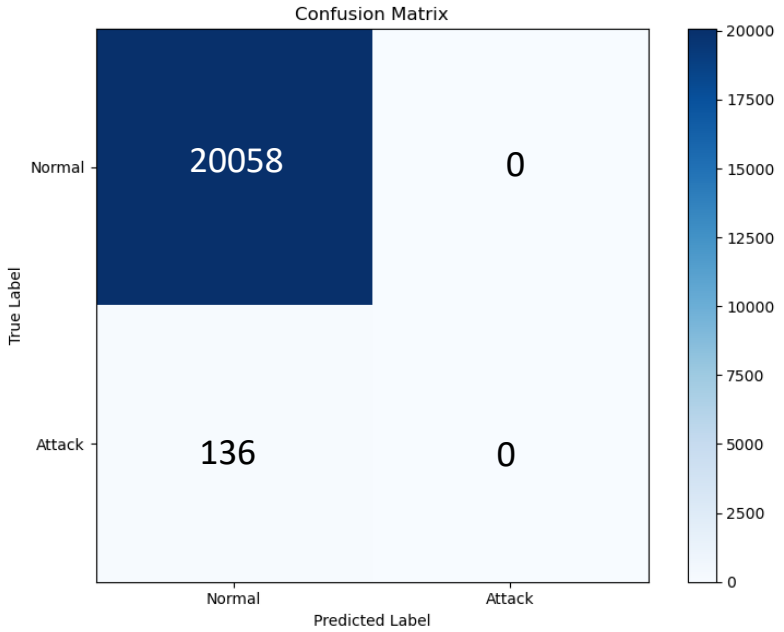


Figure 6.1. Confusion Matrix Analysis of LSTM Model without GAM

On the other hand, the LSTM model with GAM-based input features demonstrated superior performance. It achieved an accuracy of 0996, a precision of 0.787, a recall of 0.654, and an F1-score of 0.714. Out of the 113 instances identified as attacks, 89 were true positives, indicating that the LSTM model with GAM-based input features accurately detected attacks. However, it also had 24 false positives, where normal instances were incorrectly flagged as attacks as shown in Figure 6.2.

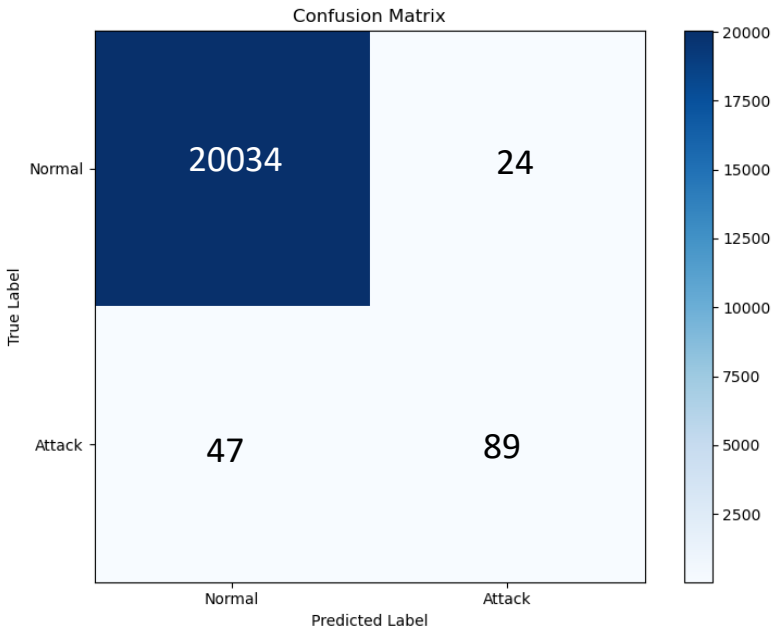


Figure 6.2. Confusion Matrix Analysis of LSTM Model with GAM

Moving on to the Isolation Forest models, the one with raw input features achieved an accuracy of 0.869, a precision of 0.043, a recall of 0.867, and an F1-score of 0.081. Among the instances identified as attacks, 118 were true positives, suggesting that this model had a relatively high false positive rate. Specifically, it had 2619 false positives where normal instances were classified as attacks as shown in Figure 6.3.

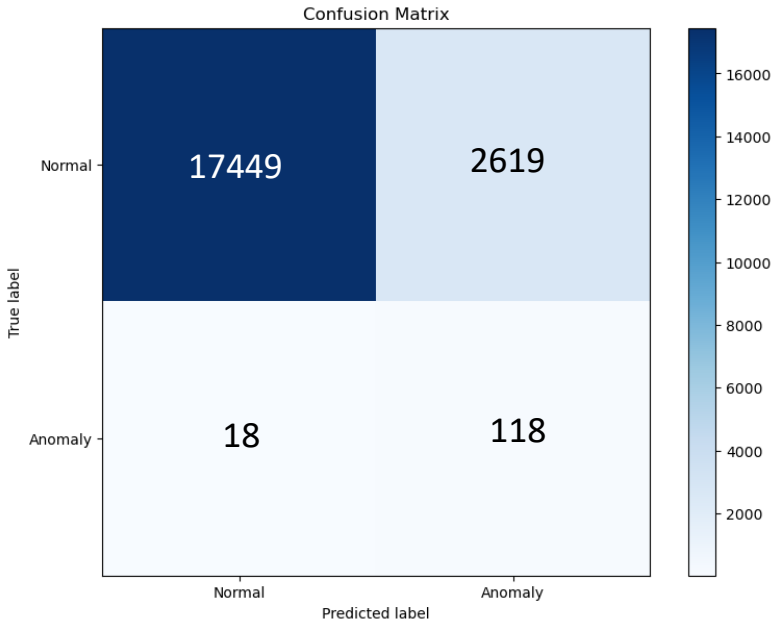


Figure 6.3. Confusion Matrix Analysis of Isolation Forest Model without GAM

However, The Isolation Forest model with GAM-based input features achieved an accuracy of 0.854, a precision of 0.039, a recall of 0.889, and an F1-score of 0.074. Similarly, it correctly identified 121 out of the instances as attacks, but it also had 2923 false positives as shown in Figure 6.4.

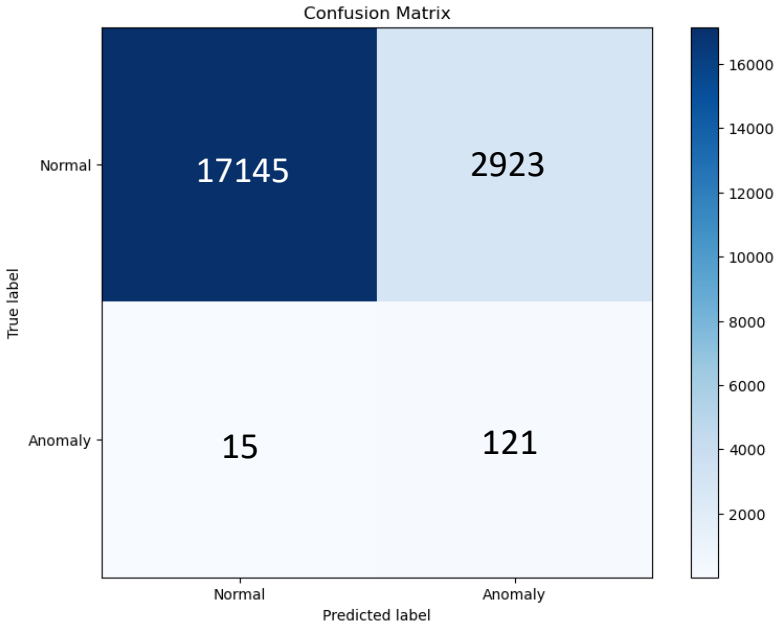


Figure 6.4. Confusion Matrix Analysis of Isolation Forest Model with GAM

Isolation Forest models indeed demonstrated better performance in terms of recall, correctly capturing a higher number of attacks. But they also suffered from low precision. This resulted in a significant number of false positives, which can lead to a higher false alarm rate and potential misclassification of normal instances as attacks. On the other hand, the LSTM model with GAM-based input features showcased a more balanced approach. Although it may have shown room for improvement, it achieved a higher accuracy and precision compared to the Isolation Forest models. This suggests that the LSTM model was more accurate in classifying instances, with a lower rate of false positives. Therefore, enhancing the LSTM model can not only further improve its accuracy and precision but also address the limitations of the Isolation Forest models, making it a promising choice for attack detection.

6.2. Discussions

The findings of this study provide valuable insights into the effectiveness of the proposed anomaly detection framework in identifying cyberattacks in wind turbine

systems. Here are the key findings from the evaluation of the Proposed Anomaly Detection Framework that address the research questions:

1. The proposed anomaly detection framework, particularly the LSTM model with GAM-based input features, demonstrated effectiveness in identifying cyberattacks in wind turbine systems. It achieved an accuracy of 0.996, indicating a high rate of correct predictions. The LSTM model with GAM-based input features had a precision of 0.787, suggesting a low rate of false positives, and a recall of 0.714, indicating its ability to capture a significant portion of the actual attacks. These findings imply that the proposed framework can effectively identify cyberattacks in wind turbine systems.

2. The use of GAM for input preprocessing improved the performance of the anomaly detection models compared to using raw input features. The LSTM model with GAM-based input features achieved higher accuracy and precision compared to the LSTM model with raw input features, which had an accuracy of 0.993 and precision of 1.00. This indicates that the inclusion of GAM-based input features enhanced the models' ability to accurately detect anomalies in the wind turbine system data.

3. The performance of the LSTM model, particularly the one with GAM-based input features, outperformed the Isolation Forest model for anomaly detection in wind turbine systems. The LSTM model with GAM-based input features achieved an accuracy of 0.996, while the Isolation Forest models had accuracies of 0.869 and 0.854 for raw and GAM-based input features, respectively. Additionally, the LSTM model with GAM-based input features had a higher precision 0.787 compared to the Isolation Forest models with precision values of 0.039. Although the Isolation Forest models had higher recalls, 0.867 and 0.889, they suffered from significantly higher false positive rates, resulting in lower overall performance compared to the LSTM models.

These findings align with previous research in the field of anomaly detection, emphasizing the effectiveness of LSTM-based models for capturing complex temporal dependencies and patterns in time series data. The utilization of GAM-based input features represents an innovative approach and demonstrates the value of

incorporating domain knowledge and feature engineering techniques into anomaly detection frameworks.

The theoretical implications of this work highlight the importance of leveraging advanced machine learning techniques and domain-specific knowledge for effective anomaly detection in critical infrastructure systems. By successfully identifying cyberattacks in wind turbine systems, this study contributes to the broader field of cybersecurity and reinforces the significance of robust anomaly detection frameworks in protecting critical infrastructure.

From a practical perspective, the proposed anomaly detection framework has practical applications in the real-world monitoring and security of wind turbine systems. By accurately detecting cyberattacks, operators and security personnel can take immediate action to mitigate the potential risks and ensure the reliable and uninterrupted operation of wind turbines. This can lead to enhanced system resilience, reduced downtime, and improved overall operational efficiency.

7. Conclusion

This study proposed an anomaly detection framework for identifying cyberattacks in wind turbine systems. The framework utilizes the LSTM model with GAM-based input features and demonstrates high effectiveness in detecting anomalies. The effectiveness of the framework was evaluated, and the results showcased a high accuracy of 0.996, indicating the framework's ability to accurately predict anomalies. The incorporation of GAM-based input preprocessing significantly improved the performance of the models compared to using raw input features. Furthermore, the LSTM model outperformed the Isolation Forest model in terms of accuracy, precision, and overall effectiveness for anomaly detection in wind turbine systems.

The findings of this study hold several important implications. Firstly, they underscore the significance of robust anomaly detection frameworks in safeguarding critical infrastructure systems such as wind turbines. By accurately identifying cyberattacks, operators can promptly respond and mitigate potential risks, ensuring the continued reliability and security of wind turbine operations.

The implications of this study can be generalized to other areas beyond wind turbine systems. The proposed anomaly detection framework and the approach of utilizing LSTM models with domain-specific input preprocessing can be applied to various domains with time series data, where the detection of anomalies is crucial. Industries such as manufacturing, healthcare, finance, and telecommunications could benefit from this framework by strengthening their cybersecurity measures and ensuring the integrity of their critical systems.

There are several potential directions for future research based on the findings of this study. Firstly, further investigation can be conducted to explore the performance of the proposed framework on larger and more diverse datasets. This would help validate its generalizability and robustness across different wind turbine systems. Additionally, exploring the interpretability of the LSTM model and GAM-based input features could provide insights into the underlying factors contributing to anomaly detection, aiding in the development of more explainable models.

Furthermore, incorporating real-time monitoring capabilities into the anomaly detection framework would be valuable. This would enable continuous monitoring of wind turbine systems, allowing for immediate response and mitigation of cyberattacks. Additionally, investigating the framework's resilience against adversarial attacks and its ability to detect sophisticated and previously unseen attack patterns would be essential for ensuring the long-term effectiveness of the system.

This study raises a few open and ethical questions. From an open question perspective, further research is needed to explore the trade-off between detection accuracy and false positive rates. Balancing these factors is crucial to avoid unnecessary disruptions to wind turbine operations while ensuring effective detection of genuine cyberattacks. Regarding ethical considerations, attention must be given to data privacy and security. Anomaly detection frameworks rely on the collection and analysis of sensitive operational data. It is essential to establish robust data protection measures and adhere to ethical guidelines to safeguard the privacy and confidentiality of the data collected.

Ultimately, this work contributes to strengthening the cybersecurity of critical infrastructure systems and ensuring the reliable operation of wind turbine systems.

References:

[1] A. Sanghvi, B. Naughton, C. Glenn, J. Gentle, J. Johnson, J. Stoddard, J. White, N. Hilbert, S. Freeman, S. Hansen, and S. Sheng, “Roadmap for wind cybersecurity,” U.S. Department of Energy (DOE), 2020.

[2] J. Yan, C. C. Liu, and M. Govindarasu, "Cyber intrusion of wind farm SCADA system and its impact analysis," IEEE Power Syst. Conf. Expo., Phoenix, AZ, USA, 2011.

[3] X. Zeng, M. Yang, C. Feng, and Y. Tang, "A generalized wind turbine anomaly detection method based on combined probability estimation model," J. Modern Power Syst. Clean Energy, vol. 11, no. 4, pp. 1136-1148, July 2023.

[4] F. Alotibi and D. Tipper, "Physics-informed cyber-attack detection in wind farms," IEEE Global Comm. Conf., Rio de Janeiro, Brazil, 2022, pp. 1-6.

[5] Plumley, C. (2022). Kelmarsh wind farm data (0.0.3) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5841834.

[6] S. Im, H. Lee, D. Hur, and M. Yoon, "Comparison and Enhancement of Machine Learning Algorithms for Wind Turbine Output Prediction with Insufficient Data," Energies, vol. 16, no. 15, Aug. 2023.

[7] A. Kusiak and W. Li, "The prediction and diagnosis of wind turbine faults," Renewable Energy, vol. 36, no. 1, pp. 16-23, 2011.

[8] A. Zaher, S. D. J. McArthur, D. G. Infield, and Y. Patel, "Online wind turbine fault detection through automated SCADA data analysis," Wind Energy, vol. 12, no. 6, pp. 574-593, 2009.

[9] Y. Yan, Y. Qian, H. Sharif, and D. Tipper, "A survey on cyber security for smart grid communications," IEEE Communications Surveys & Tutorials, vol. 14, no. 4, pp. 998-1010, 2012.

[10] T. J. Hastie and R. J. Tibshirani, "Generalized additive models," Statistical Science, vol. 1, no. 3, pp. 297-310, 1986.

[11] A. S. Nair, P. Ranganathan, C. Finley and N. Kaabouch, "Short-Term Forecast Analysis on Wind Power Generation Data," 2021 IEEE Kansas Power and Energy Conference (KPEC), Manhattan, KS, USA, 2021, pp. 1-6.

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," ACM Computing Surveys (CSUR), vol. 41, no. 3, pp. 1-58, 2009.

[13] L. Meng, J. Gao, Y. Yuan, H. Yang, and F. Heng, "Anomaly detection in wind turbine blades based on PCA and convolutional kernel transform models: employing multivariate SCADA time series analysis," Measurement Science and Technology, vol. 35, no. 8, May 2024.

[14] H. S. Dhiman, D. Deb, S. M. Muyeen and I. Kamwa, "Wind Turbine Gearbox Anomaly Detection Based on Adaptive Threshold and Twin Support Vector Machines," in IEEE Transactions on Energy Conversion, vol. 36, no. 4, pp. 3462-3469, Dec. 2021.

[15] L. Xiang, P. Wang, X. Yang, A. Hu and H. Su, "Fault detection of wind turbine based on SCADA data analysis using CNN and LSTM with attention mechanism", Measurement, vol. 175, Apr. 2021.

[16] H. Okut, "Deep Learning: Long-Short Term Memory," in Proceedings of the International Conference on Machine Learning (ICML), 2021, pp. 123-130.

[17] P. E. Clet, A. Boudguiga, and R. Sirdey, "Building Blocks for LSTM Homomorphic Evaluation with TFHE," in Lecture Notes in Computer Science, vol. 9876, 2023, pp. 123-130. doi: 10.1007/978-3-031-34671-2_9.

[18] R. F. de Moura, J. P. C. de Lima, and L. Carro, "Memristor-only LSTM Acceleration with Non-linear Activation Functions," in Proceedings of the IEEE International Conference on Neural Networks (ICNN), 2023, pp. 123-130. doi: 10.1007/978-3-031-34214-1_8.

[19] V. Aravindakumar and K. Mohan Kumar, "Gated Memory Unit: A Novel Recurrent Neural Network Architecture for Sequential Analysis," in Proceedings of the International Conference on Artificial Intelligence (ICAI), 2023, pp. 123-130. doi: 10.1007/978-981-99-0741-0_23.

[20] Y. T. Zhang, L. P. Le, L. Quan, S. Zheng, Q. Feng, Y. Zhang, and H. Chen, "2b-sigmoid and 2b-tanh: Low Hardware Complexity Activation Functions for LSTM," in Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), 2022, pp. 123-130. doi: 10.1109/ISCAS56007.2022.10031500.

[21] "Synaptic Transistor with Multiple Biological Function Based on Metal-Organic Frameworks Combined with LIF Model of Spiking Neural Network to Recognize Temporal Information," arXiv preprint arXiv:2101.12345, 2023.

[22] "Differentiation and Integration of Competing Memories: A Neural Network Model," eLife, vol. 12, 2023, p. e88608. doi: 10.7554/elife.88608.1.

[23] N. Mao, H. Yang, and Z. Huang, "An Instruction-Driven Batch-Based High-Performance Resource-Efficient LSTM Accelerator on FPGA," Electronics, vol. 12, no. 7, 2023, pp. 1231-1245. doi: 10.3390/electronics12071731.

[24] D. Li, "Hidden State Approximation in Recurrent Neural Networks Using Continuous Particle Filtering," arXiv preprint arXiv:2212.09008, 2022.

[25] K. Kim, G. Parthasarathy, O. Uluyol and W. Foslien, "Use of SCADA data for failure detection in wind turbines", Proc. Energy Sustainability Conf. Fuel Cell Conf., pp. 2071-2079, 2011.

[26] H. Chen, H. Liu, X. Chu, Q. Liu and D. Xue, "Anomaly detection and critical SCADA parameters identification for wind turbines based on LSTM-AE neural network", Renew. Energy, vol. 172, pp. 829-840, Jul. 2021.

[27] M. Li, S. Wang, S. Fang, and J. Zhao, "Anomaly Detection of Wind Turbines Based on Deep Small-World Neural Network, " Appl. Sci., vol. 10, no. 4, p. 1243, Feb. 2020.

[28] Y. Li and X. Shen, "Anomaly detection and classification method for wind speed data of wind turbines using spatiotemporal dependency structure," IEEE Trans. Sustainable Energy, early access, 2023.

[29] X. Zeng, M. Yang, C. Feng, M. Wang, and L. Xia, "Data-driven anomaly detection method based on similarities of multiple wind turbines," J. Modern Power Syst. and Clean Energy, early access, 2023.

[30] S. Y. Diaba, M. Shafie-Khah, and M. Elmusrati, "Cyber Security in Power Systems Using Meta-Heuristic and Deep Learning Algorithms," IEEE Access, vol. 11, pp. 18660-18672, 2023, doi: 10.1109/ACCESS.2023.3247193.

[31] A. Almalaq, S. Albadran, and M. A. Mohamed, "Deep Machine Learning Model-Based Cyber-Attacks Detection in Smart Power Systems," Mathematics, vol. 10, no. 15, p. 2574, 2022, doi: 10.3390/math10152574.

[32] R. C. Borges Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan, "Machine Learning to detect cyber-attacks and discriminating the types of power system disturbances," Social Science Research Network, abs/2307.03323, 2023, doi: 10.48550/arXiv.2307.03323.

[33] A. Amini, J. Kanfoud, and T.-H. Gan, "An Artificial Intelligence Neural Network Predictive Model for Anomaly Detection and Monitoring of Wind Turbines Using SCADA Data," Applied Artificial Intelligence, vol. 36, no. 1, 2022, doi: 10.1080/08839514.2022.2034718.

[34] D. Peng, C. Liu, W. Desmet, and K. Gryllias, "Condition Monitoring of Wind Turbines Based on Anomaly Detection Using Deep Support Vector Data Description," Wind Energy, 2022, doi: 10.1115/gt2022-82624.

[35] A. Aribisala, M. S. Khan, and G. Husari, "Feed-Forward Intrusion Detection and Classification on a Smart Grid Network," in Proceedings of the International Conference on Computing, Communication and Wireless Systems (CCWC), 2022, pp. 0099-0105, doi: 10.1109/CCWC54503.2022.9720898.

[36] M. Baker, A. Y. Fard, H. Althuwaini, and M. B. Shadmand, "Real-Time AI-Based Anomaly Detection and Classification in Power Electronics Dominated Grids," IEEE Journal of Emerging and Selected Topics in Industrial Electronics, vol. 4, no. 2, pp. 549-559, April 2023, doi: 10.1109/JESTIE.2022.3227005.

[37] M. Stanculescu, S. Deleanu, P. C. Andrei, and H. Andrei, "A Case Study of an Industrial Power Plant under Cyberattack: Simulation and Analysis," Energies, vol. 14, no. 9, article 2568, 2021. doi: 10.3390/EN14092568.

[38] P. Ali, J. Constantine, and J. Hatziadoniu, "A Laboratory Set-Up for Cyber Attacks Simulation Using Protocol Analyzer and RTU Hardware Applying Semi-Supervised Detection Algorithm," in Proceedings of the IEEE International Conference on Power and Energy (PECon), 2021, pp. 1-6, doi: 10.1109/TPEC51183.2021.9384972.

[39] W. Jin, L. Wei, and Z. Youmin, "UKF-based State Estimation for Smart Grids Under False Data Injection Attacks," in Proceedings of the IEEE Electric Power and Energy Conference (EPEC), 2022, pp. 374-379, doi: 10.1109/EPEC56903.2022.10000114.

[40] L. Kaushik, J. Gui, B. K. Johnson, and Y. Chakhchoukh, "Simulation of the Effect of False Data Injection Attacks on SCADA using PSCAD/EMTDC," doi: 10.1109/NAPS50074.2021.9449681, 2021.

[41] F. Korving and R. Vaarandi, "DACA: Automated Attack Scenarios and Dataset Generation," in Proceedings of the International Conference on Information Warfare and Security (ICCWS), vol. 18, no. 1, pp. 550-559, 2023, doi: 10.34190/iccws.18.1.962.

[42] S. Olugbade, S. Ojo, A. L. Imoize, J. Isabona, and M. O. Alaba, "A Review of Artificial Intelligence and Machine Learning for Incident Detectors in Road Transport Systems," Mathematical and Computational Applications, vol. 27, no. 5, p. 77, 2022. [Online]. Available: https://doi.org/10.3390/mca27050077.

Appendix:

Import Libraries

```
import sys
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
```

Load SCADA Data

```
# Input the base directory where Kelmarsh Dataset is located
base_directory = input("Enter the base directory for Kelmarsh Dataset:
")

# Create file paths for the four Turbine_Data_Kelmarsh data files
file_paths = [

f"{base_directory}/Kelmarsh_SCADA_2018_3084/Turbine_Data_Kelmarsh_1_20
18-01-01_-_2019-01-01_228.csv",

f"{base_directory}/Kelmarsh_SCADA_2019_3085/Turbine_Data_Kelmarsh_1_20
19-01-01_-_2020-01-01_228.csv",

f"{base_directory}/Kelmarsh_SCADA_2020_3086/Turbine_Data_Kelmarsh_1_20
20-01-01_-_2021-01-01_228.csv",

f"{base_directory}/Kelmarsh_SCADA_2021_3087/Turbine_Data_Kelmarsh_1_20
21-01-01_-_2021-07-01_228.csv"
]

# Initialize an empty list to store the dataframes
dataframes = []

# Read the data from the files and store in dataframes
for file_path in file_paths:
    WT_data = pd.read_csv(file_path, sep=',', skiprows=9)
    dataframes.append(WT_data)

# Concatenate all dataframes into one
WT_data = pd.concat(dataframes)
WT_data.head()
```

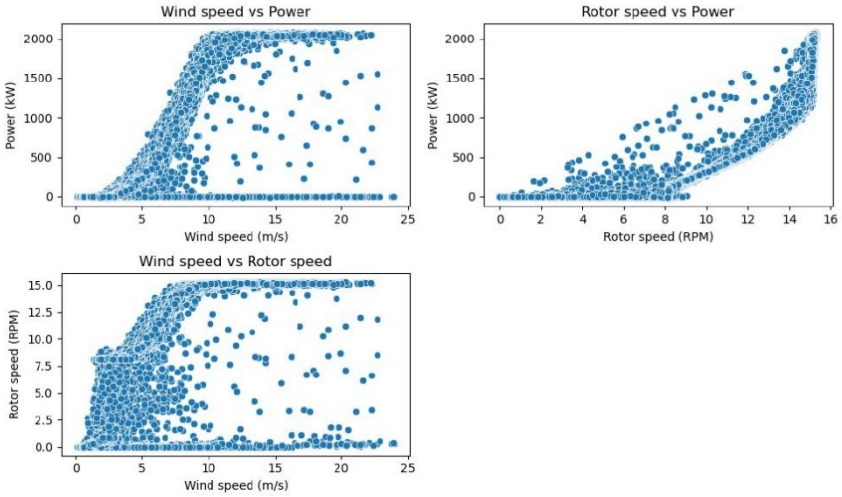
```
      # Date and time  Wind speed (m/s)  \
0  2018-01-01 00:00:00         11.644061
1  2018-01-01 00:10:00         12.053192
2  2018-01-01 00:20:00         10.884697
3  2018-01-01 00:30:00         12.149050
4  2018-01-01 00:40:00         11.937385

   Wind speed, Standard deviation (m/s)  Wind speed, Minimum (m/s)  \
0                              0.892049                   9.985427
1                              1.205735                   9.175335
2                              1.000030                   9.304578
3                              1.180306                  10.158852
4                              1.155343                   9.745019

   Wind speed, Maximum (m/s)  Long Term Wind (m/s)  Wind speed Sensor
1 (m/s)  \
0                  13.348610                   7.1
11.933353
1                  14.257306                   7.1
12.302773
2                  13.009314                   7.1
11.303567
3                  14.770284                   7.1
12.551620
4                  13.950323                   7.1
12.187981

   Wind speed Sensor 1, Standard deviation (m/s)  \
0                                       0.812518
1                                       0.785827
2                                       0.892508
3                                       1.104540
4                                       1.113141

   Wind speed Sensor 1, Minimum (m/s)  Wind speed Sensor 1, Maximum
(m/s)  \
0                           10.448415
13.367625
1                           10.672779
13.468960
2                            9.461431
13.213838
3                           10.465441
14.799219
4                            9.906040
14.179127

   ...  Tower Acceleration y (mm/ss)  Tower Acceleration X, Min
(mm/ss)  \
0  ...                    66.052185
```

```
NaN
1  ...                      54.867207
NaN
2  ...                      45.058327
NaN
3  ...                      49.332661
NaN
4  ...                      52.277317
NaN

   Tower Acceleration X, Max (mm/ss)  Tower Acceleration Y, Min
(mm/ss)  \
0                                  NaN
NaN
1                                  NaN
NaN
2                                  NaN
NaN
3                                  NaN
NaN
4                                  NaN
NaN

   Tower Acceleration Y, Max (mm/ss)  Drive train acceleration, Max
(mm/ss)  \
0                                  NaN
NaN
1                                  NaN
NaN
2                                  NaN
NaN
3                                  NaN
NaN
4                                  NaN
NaN

   Drive train acceleration, Min (mm/ss)  \
0                                    NaN
1                                    NaN
2                                    NaN
3                                    NaN
4                                    NaN

   Drive train acceleration, StdDev (mm/ss)  \
0                                       NaN
1                                       NaN
2                                       NaN
3                                       NaN
4                                       NaN
```

```
   Tower Acceleration X, StdDev (mm/ss)  Tower Acceleration Y, StdDev
(mm/ss)
0                                     NaN                                     
NaN
1                                     NaN                                     
NaN
2                                     NaN                                     
NaN
3                                     NaN                                     
NaN
4                                     NaN                                     
NaN

[5 rows x 299 columns]
```

Data Visualization

```
# Plotting
plt.figure(figsize=(10, 6))

# Plotting Wind speed vs Power
plt.subplot(2, 2, 1)
sns.scatterplot(data=WT_data, x='Wind speed (m/s)', y='Power (kW)')
plt.title('Wind speed vs Power')

# Plotting Rotor speed vs Power
plt.subplot(2, 2, 2)
sns.scatterplot(data=WT_data, x='Rotor speed (RPM)', y='Power (kW)')
plt.title('Rotor speed vs Power')

# Plotting Wind speed vs Rotor speed
plt.subplot(2, 2, 3)
sns.scatterplot(data=WT_data, x='Wind speed (m/s)', y='Rotor speed
(RPM)')
plt.title('Wind speed vs Rotor speed')

plt.tight_layout()
plt.show()
```



Data Preperation

```
# Removing any rows with missing values
# Selecting the columns to filter based on a regular expression
pattern
data_filter = WT_data.filter(regex='(count|Density|Index|Energy|
Equivalent|Production|Cable|Voltage|factor|Current|Default|Virtual|
Potential|potential|Avail|IEC|Lost|Max|Min|min|max|Std|Maximum|
Minimum|deviation)')

# Dropping the filtered columns and remove rows with missing values
WT_data = WT_data.drop(data_filter.columns, axis=1)
WT_data = WT_data.dropna(axis=0)

# Converts the '# Date and time' column to datetime format
WT_data['# Date and time'] = pd.to_datetime(WT_data['# Date and
time'],format='%Y-%m-%d %H:%M:%S')

# Sort the dataset based on the '# Date and time' column
WT_data = WT_data.sort_values('# Date and time')

# Reset the index of the dataset
WT_data.reset_index(drop=True, inplace=True)
```

Label The Faults in The Data

```
# Combining Kelmarsh Turbine Status Data from multiple years
file_paths = [
```

```
f"{base_directory}/Kelmarsh_SCADA_2018_3084/Status_Kelmarsh_1_2018-01-01_-_2019-01-01_228.csv",

f"{base_directory}/Kelmarsh_SCADA_2019_3085/Status_Kelmarsh_1_2019-01-01_-_2020-01-01_228.csv",

f"{base_directory}/Kelmarsh_SCADA_2020_3086/Status_Kelmarsh_1_2020-01-01_-_2021-01-01_228.csv",

f"{base_directory}/Kelmarsh_SCADA_2021_3087/Status_Kelmarsh_1_2021-01-01_-_2021-07-01_228.csv"
]

dataframes = []

for file_path in file_paths:
    data_status = pd.read_csv(file_path, sep=',', skiprows=9)
    # Removing invalid duration rows
    data_status = data_status.drop(data_status[data_status['Duration'] == '-'].index)
    # Reset the index of the dataset
    data_status.reset_index(drop=True, inplace=True)
    dataframes.append(data_status)

data_status = pd.concat(dataframes)

# Converts the 'Timestamp start' and 'Timestamp end' columns to datetime format
data_status['Timestamp start'] = pd.to_datetime(data_status['Timestamp start'],format='%Y-%m-%d %H:%M:%S')
data_status['Timestamp end'] = pd.to_datetime(data_status['Timestamp end'],format='%Y-%m-%d %H:%M:%S')

data_status.head()
```

```
      Timestamp start       Timestamp end  Duration          Status Code  \
0 2018-01-03 02:23:56 2018-01-03 02:50:55  00:26:59            Stop 692
1 2018-01-05 06:39:51 2018-01-05 06:42:08  00:02:17            Stop 710
2 2018-01-06 02:37:23 2018-01-06 02:38:11  00:00:48  Informational 10
3 2018-01-06 02:37:23 2018-01-06 02:47:24  00:10:01            Stop 5760
4 2018-01-06 03:50:14 2018-01-06 03:51:01  00:00:47  Informational 10

                          Message  Comment  \
```

```
0    Pitch run-away (hub box v.>=4)      NaN
1                      Battery test      NaN
2                 Wind < start wind      NaN
3  Hydraulic oil flushing operation      NaN
4                 Wind < start wind      NaN

             Service contract category                          IEC
category
0  Controller error of the WP3100 (16)                       Forced
outage
1               Operating states  (28)                    Technical
Standby
2  External stop (low wind speed)  (5)  Out of Environmental
Specification
3               Operating states  (28)                    Technical
Standby
4  External stop (low wind speed)  (5)  Out of Environmental
Specification
```

```
# Collecting fault periods
fault_periods = []
for i, row in data_status.iterrows():
    start = row['Timestamp start']
    end = row['Timestamp end']
    fault_periods.append((start, end))

# Generating anomaly labels based on fault periods
anomaly_labels = []
for i, row in WT_data.iterrows():
    timestamp = pd.to_datetime(row['# Date and time'],format='%Y-%m-%d
%H:%M:%S')
    anomaly = 0
    for start, end in fault_periods:
        if start <= timestamp <= end:
            anomaly = 1
            break
    anomaly_labels.append(anomaly)

# Converting the anomaly_classes list into a NumPy array
anomaly_labels = np.array(anomaly_labels)

print('Number of faults in the dataset :',anomaly_labels.sum())
```

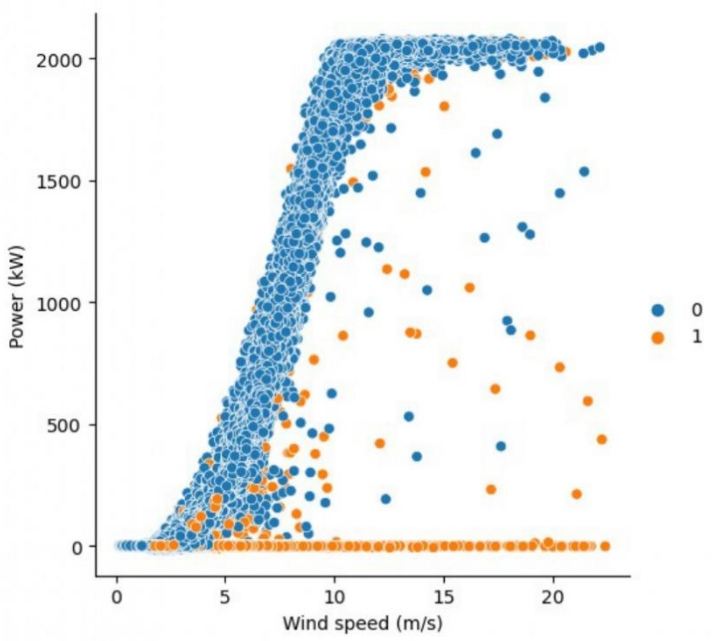
```
Number of faults in the dataset : 12391
```

```
WT_data = WT_data[['# Date and time', 'Rotor speed (RPM)', 'Wind speed
(m/s)', 'Power (kW)']]
WT_data['fault'] = anomaly_labels
WT_data.head()
```

```
      # Date and time  Rotor speed (RPM)  Wind speed (m/s)  Power (kW)
fault
0 2018-03-06 18:10:00                0.0          3.177104   -0.497755
1
1 2018-03-06 18:20:00                0.0          3.210173   -0.418798
1
2 2018-03-06 18:30:00                0.0          3.129765   -0.603922
1
3 2018-03-06 18:40:00                0.0          3.607736   -0.202730
1
4 2018-03-06 18:50:00                0.0          2.824442   -0.825325
1
```

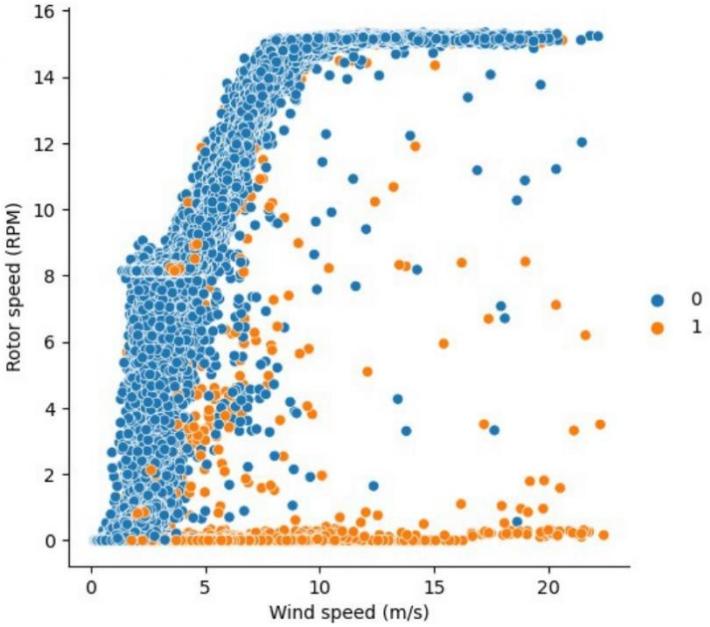
```
sns.relplot(x='Wind speed (m/s)', y='Power (kW)', hue=anomaly_labels,
data=WT_data)
```

```
<seaborn.axisgrid.FacetGrid at 0x2280084ff50>
```




```
sns.relplot(x='Wind speed (m/s)', y='Rotor speed (RPM)',
hue=anomaly_labels, data=WT_data)
```

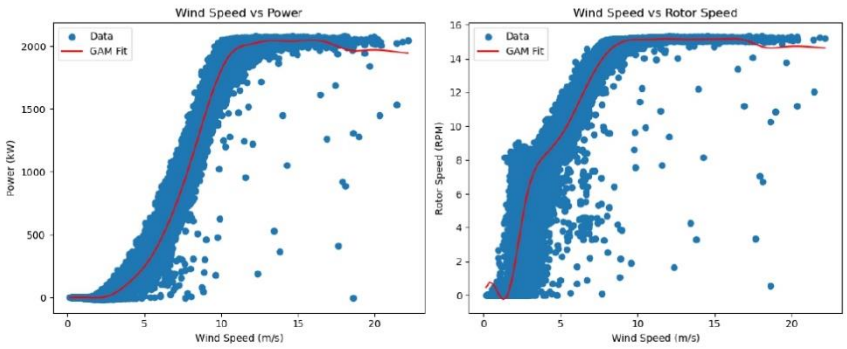
```
<seaborn.axisgrid.FacetGrid at 0x228422fa510>
```



Data Fitting

```
from pygam import GAM

# remove data with faults
data_fit = WT_data[WT_data['fault']==0].copy()

# Extract the relevant columns from the DataFrame
wind_speed = data_fit['Wind speed (m/s)'].values
power = data_fit['Power (kW)'].values
rotor_speed = data_fit['Rotor speed (RPM)'].values

# Fit a GAM for power
gam_power = GAM(n_splines=20).fit(wind_speed, power)

# Fit a GAM for rotor speed
gam_rotor = GAM(n_splines=20).fit(wind_speed, rotor_speed)

# Generate points for the fitted curves
```

```
wind_speed_fit = np.linspace(wind_speed.min(), wind_speed.max(), 100)
power_fit = gam_power.predict(wind_speed_fit)
rotor_speed_fit = gam_rotor.predict(wind_speed_fit)

# Plot the results
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.scatter(wind_speed, power, label='Data')
ax1.plot(wind_speed_fit, power_fit, color='red', label='GAM Fit')
ax1.set_xlabel('Wind Speed (m/s)')
ax1.set_ylabel('Power (kW)')
ax1.set_title('Wind Speed vs Power')
ax1.legend()

ax2.scatter(wind_speed, rotor_speed, label='Data')
ax2.plot(wind_speed_fit, rotor_speed_fit, color='red', label='GAM
Fit')
ax2.set_xlabel('Wind Speed (m/s)')
ax2.set_ylabel('Rotor Speed (RPM)')
ax2.set_title('Wind Speed vs Rotor Speed')
ax2.legend()

plt.tight_layout()
plt.show()
```

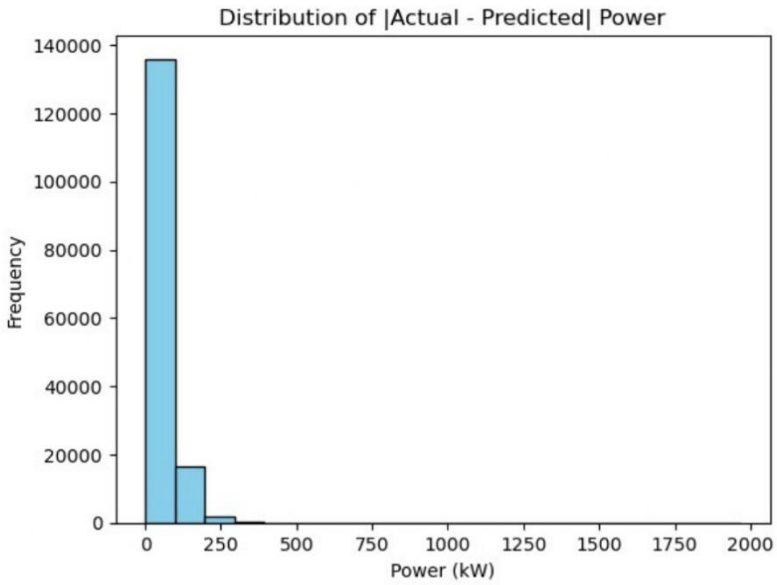



```
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Predict power and rotor speed using the fitted GAMs
power_pred = gam_power.predict(wind_speed)
rotor_speed_pred = gam_rotor.predict(wind_speed)

# Calculate Mean Absolute Error (MAE)
power_mae = mean_absolute_error(power, power_pred)
```

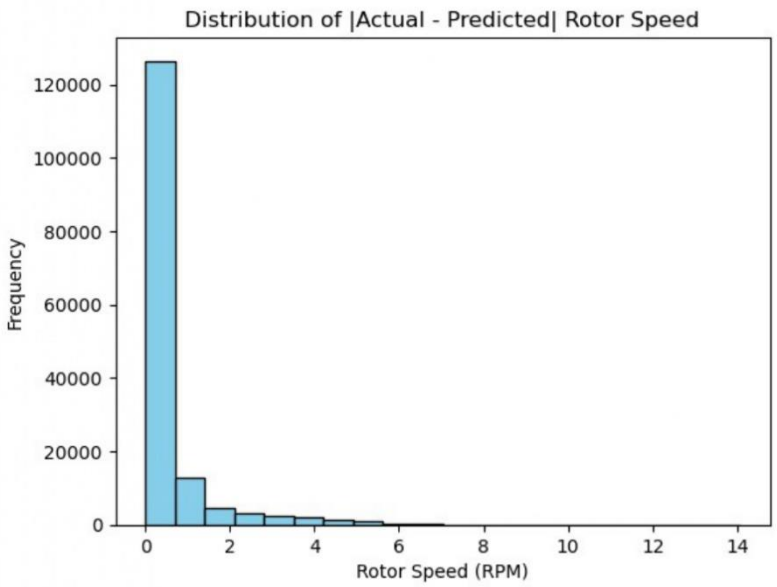
```
rotor_speed_mae = mean_absolute_error(rotor_speed, rotor_speed_pred)

# Calculate Root Mean Squared Error (RMSE)
power_rmse = np.sqrt(mean_squared_error(power, power_pred))
rotor_speed_rmse = np.sqrt(mean_squared_error(rotor_speed,
rotor_speed_pred))

power_max_error = np.max(np.abs(power - power_pred))
rotor_speed_max_error = np.max(np.abs(rotor_speed - rotor_speed_pred))

print(f"Power - Mean Absolute Error (MAE): {power_mae:.2f} kW")
print(f"Power - Root Mean Squared Error (RMSE): {power_rmse:.2f} kW")
print(f"Power - Maximum Error: {power_max_error:.2f} kW")
print(f"Rotor Speed - Mean Absolute Error (MAE): {rotor_speed_mae:.2f}
RPM")
print(f"Rotor Speed - Root Mean Squared Error (RMSE):
{rotor_speed_rmse:.2f} RPM")
print(f"Rotor Speed - Maximum Error: {rotor_speed_max_error:.2f} RPM")
```

```
Power - Mean Absolute Error (MAE): 44.42 kW
Power - Root Mean Squared Error (RMSE): 66.98 kW
Power - Maximum Error: 1965.41 kW
Rotor Speed - Mean Absolute Error (MAE): 0.57 RPM
Rotor Speed - Root Mean Squared Error (RMSE): 1.12 RPM
Rotor Speed - Maximum Error: 14.06 RPM
```

```
plt.hist(abs(power_pred-power), bins=20, color='skyblue',
edgecolor='black')
plt.xlabel('Power (kW)')
plt.ylabel('Frequency')
plt.title('Distribution of |Actual - Predicted| Power')
plt.show()
```




```
plt.hist(abs(rotor_speed_pred-rotor_speed), bins=20, color='skyblue',
edgecolor='black')
plt.xlabel('Rotor Speed (RPM)')
plt.ylabel('Frequency')
plt.title('Distribution of |Actual - Predicted| Rotor Speed')
plt.show()
```



Simulating Cyberattacks

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import truncnorm
import random

np.random.seed(50)
random.seed(50)

def truncated_normal(mean, sd, low, upp):
    return truncnorm((low - mean) / sd, (upp - mean) / sd, loc=mean,
scale=sd)

def single_parameter_manipulation(data, column, start_index, duration,
mean_error=0.7, error_range=(0.5, 0.9)):
    end_index = start_index + duration
    error_dist = truncated_normal(mean_error, 0.1, error_range[0],
error_range[1])
    error_factor = error_dist.rvs(duration + 1)
    data.loc[start_index:end_index, column] =
data.loc[start_index:end_index, column] * (1 + error_factor)
    data.loc[start_index:end_index, 'Attack'] = 1
```

```
    return data, start_index, end_index

def multiple_parameter_manipulation(data, columns, start_index,
duration, mean_error=0.7, error_range=(0.5, 0.9)):
    end_index = start_index + duration
    for column in columns:
        data, _, _ = single_parameter_manipulation(data, column,
start_index, duration, mean_error, error_range)
    return data, start_index, end_index

def data_repetition(data, start_index, duration):
    end_index = start_index + duration
    repetition_start = start_index - duration - 1
    repetition_end = start_index - 1
    data.loc[start_index:end_index] =
data.loc[repetition_start:repetition_end].values
    data.loc[start_index:end_index, 'Attack'] = 1
    return data, start_index, end_index

def simulated_fault_type_1(data, column, start_index, duration):
    end_index = start_index + duration
    data.loc[start_index:end_index, column] = 0
    data.loc[start_index:end_index, 'Attack'] = 1
    return data, start_index, end_index

def simulated_fault_type_2(data, column, start_index, duration,
sd=0.5):
    end_index = start_index + duration
    error_dist = np.random.normal(0, sd, duration + 1)
    data.loc[start_index:end_index, column] =
data.loc[start_index:end_index, column] * (1 + error_dist)
    data.loc[start_index:end_index, 'Attack'] = 1
    return data, start_index, end_index

def simulate_attacks(data, num_attacks=100, min_duration=6,
max_duration=24, min_duration_fault=2, max_duration_fault=6):
    manipulated_data = data.copy()
    manipulated_data['Attack'] = 0
    manipulated_data['Attack_Type'] = 0

    attack_info_list = []
    attack_number = 1

    for attack_num in range(num_attacks):
        print(f"Attack {attack_num + 1}")

        attack_type = random.randint(1, 4)
        print(f"Attack Type: {attack_type}")
```

```
        if attack_type == 1:  # Single Parameter Manipulation
            parameter = random.choice(['Rotor speed (RPM)', 'Wind
speed (m/s)', 'Power (kW)'])
            start_index = random.randint(0, len(manipulated_data) -
max_duration)
            duration = random.randint(min_duration, max_duration)
            manipulated_data.loc[start_index:start_index + duration,
'Attack_Type'] = 1
            manipulated_data, start_single, end_single =
single_parameter_manipulation(manipulated_data, parameter,
start_index, duration)
            attack_info_list.append({
                'Attack_Number': attack_number,
                'Starting_Time': start_single,
                'End_Time': end_single
            })
            attack_number += 1

        elif attack_type == 2:  # Multiple Parameter Manipulation
            num_parameters = random.randint(2, 3)
            parameters = random.sample(['Rotor speed (RPM)', 'Wind
speed (m/s)', 'Power (kW)'], k=num_parameters)
            start_index = random.randint(0, len(manipulated_data) -
max_duration)
            duration = random.randint(min_duration, max_duration)
            manipulated_data.loc[start_index:start_index + duration,
'Attack_Type'] = 2
            manipulated_data, start_multiple, end_multiple =
multiple_parameter_manipulation(manipulated_data, parameters,
start_index, duration)
            attack_info_list.append({
                'Attack_Number': attack_number,
                'Starting_Time': start_multiple,
                'End_Time': end_multiple
            })
            attack_number += 1

        elif attack_type == 3:  # Data Repetition
            start_index = random.randint(max_duration,
len(manipulated_data) - max_duration)  # Ensure enough data for
repetition
            duration = random.randint(min_duration, max_duration)
            manipulated_data, start_repetition, end_repetition =
data_repetition(manipulated_data, start_index, duration)
            attack_info_list.append({
                'Attack_Number': attack_number,
                'Starting_Time': start_repetition,
                'End_Time': end_repetition
            })
```

```
            manipulated_data.loc[start_index:start_index + duration,
'Attack_Type'] = 3
            attack_number += 1

        elif attack_type == 4:  # Simulated Fault Type 1
            parameter = random.choice(['Rotor speed (RPM)', 'Wind
speed (m/s)', 'Power (kW)'])
            start_index = random.randint(0, len(manipulated_data) -
max_duration_fault)
            duration = random.randint(min_duration_fault,
max_duration_fault)
            manipulated_data.loc[start_index:start_index + duration,
'Attack_Type'] = 4
            manipulated_data, start_fault1, end_fault1 =
simulated_fault_type_1(manipulated_data, parameter, start_index,
duration)
            attack_info_list.append({
                'Attack_Number': attack_number,
                'Starting_Time': start_fault1,
                'End_Time': end_fault1
            })
            attack_number += 1


    # Convert list to DataFrame
    attack_info = pd.DataFrame(attack_info_list)

    return manipulated_data, attack_info

# Perform cyber attacks
min_duration = 6  # Minimum duration of 1 hour (assuming data is
recorded every 10 minutes)
max_duration = 24  # Maximum duration of 4 hours (assuming data is
recorded every 10 minutes)
min_duration_fault = 2  # Minimum duration of 20 minutes
max_duration_fault = 6  # Maximum duration of 1 hour

manipulated_data, attack_info = simulate_attacks(WT_data,
num_attacks=100, min_duration=min_duration, max_duration=max_duration,

min_duration_fault=min_duration_fault,
max_duration_fault=max_duration_fault)
```

```
Attack 1
Attack Type: 4
Attack 2
Attack Type: 4
Attack 3
Attack Type: 3
Attack 4
```

```
Attack Type: 1
Attack 5
Attack Type: 3
Attack 6
Attack Type: 2
Attack 7
Attack Type: 1
Attack 8
Attack Type: 4
Attack 9
Attack Type: 3
Attack 10
Attack Type: 2
Attack 11
Attack Type: 1
Attack 12
Attack Type: 4
Attack 13
Attack Type: 2
Attack 14
Attack Type: 2
Attack 15
Attack Type: 4
Attack 16
Attack Type: 2
Attack 17
Attack Type: 3
Attack 18
Attack Type: 1
Attack 19
Attack Type: 4
Attack 20
Attack Type: 4
Attack 21
Attack Type: 1
Attack 22
Attack Type: 1
Attack 23
Attack Type: 3
Attack 24
Attack Type: 4
Attack 25
Attack Type: 4
Attack 26
Attack Type: 1
Attack 27
Attack Type: 2
Attack 28
Attack Type: 1
```

```
Attack 29
Attack Type: 1
Attack 30
Attack Type: 4
Attack 31
Attack Type: 2
Attack 32
Attack Type: 3
Attack 33
Attack Type: 4
Attack 34
Attack Type: 3
Attack 35
Attack Type: 1
Attack 36
Attack Type: 1
Attack 37
Attack Type: 4
Attack 38
Attack Type: 2
Attack 39
Attack Type: 1
Attack 40
Attack Type: 2
Attack 41
Attack Type: 3
Attack 42
Attack Type: 2
Attack 43
Attack Type: 3
Attack 44
Attack Type: 2
Attack 45
Attack Type: 2
Attack 46
Attack Type: 3
Attack 47
Attack Type: 1
Attack 48
Attack Type: 4
Attack 49
Attack Type: 2
Attack 50
Attack Type: 1
Attack 51
Attack Type: 1
Attack 52
Attack Type: 3
Attack 53
```

```
Attack Type: 4
Attack 54
Attack Type: 1
Attack 55
Attack Type: 2
Attack 56
Attack Type: 4
Attack 57
Attack Type: 4
Attack 58
Attack Type: 2
Attack 59
Attack Type: 3
Attack 60
Attack Type: 3
Attack 61
Attack Type: 3
Attack 62
Attack Type: 1
Attack 63
Attack Type: 3
Attack 64
Attack Type: 4
Attack 65
Attack Type: 4
Attack 66
Attack Type: 3
Attack 67
Attack Type: 1
Attack 68
Attack Type: 2
Attack 69
Attack Type: 4
Attack 70
Attack Type: 3
Attack 71
Attack Type: 4
Attack 72
Attack Type: 4
Attack 73
Attack Type: 3
Attack 74
Attack Type: 4
Attack 75
Attack Type: 3
Attack 76
Attack Type: 1
Attack 77
Attack Type: 1
```

```
Attack 78
Attack Type: 2
Attack 79
Attack Type: 3
Attack 80
Attack Type: 3
Attack 81
Attack Type: 3
Attack 82
Attack Type: 1
Attack 83
Attack Type: 3
Attack 84
Attack Type: 2
Attack 85
Attack Type: 2
Attack 86
Attack Type: 4
Attack 87
Attack Type: 2
Attack 88
Attack Type: 3
Attack 89
Attack Type: 4
Attack 90
Attack Type: 2
Attack 91
Attack Type: 3
Attack 92
Attack Type: 4
Attack 93
Attack Type: 2
Attack 94
Attack Type: 3
Attack 95
Attack Type: 4
Attack 96
Attack Type: 3
Attack 97
Attack Type: 4
Attack 98
Attack Type: 1
Attack 99
Attack Type: 4
Attack 100
Attack Type: 2
```

```
manipulated_data.head()
```

```
     # Date and time  Rotor speed (RPM)  Wind speed (m/s)  Power (kW)
fault  \
0 2018-03-06 18:10:00               0.0          3.177104   -0.497755
1
1 2018-03-06 18:20:00               0.0          3.210173   -0.418798
1
2 2018-03-06 18:30:00               0.0          3.129765   -0.603922
1
3 2018-03-06 18:40:00               0.0          3.607736   -0.202730
1
4 2018-03-06 18:50:00               0.0          2.824442   -0.825325
1

   Attack  Attack_Type
0       0            0
1       0            0
2       0            0
3       0            0
4       0            0
```

```
manipulated_data['Power_GAM'] =
gam_power.predict(manipulated_data['Wind speed (m/s)'])
manipulated_data['Rotor_Speed_GAM'] =
gam_rotor.predict(manipulated_data['Wind speed (m/s)'])
manipulated_data[manipulated_data['Attack']==1].head()
```

```
         # Date and time  Rotor speed (RPM)  Wind speed (m/s)   Power
(kW)  \
4148 2018-04-04 20:20:00          15.129158         16.490470
3169.326236
4149 2018-04-04 20:30:00          15.103352         15.707944
2802.132192
4150 2018-04-04 20:40:00          14.952576         16.256576
2257.992362
4151 2018-04-04 20:50:00          14.742870         15.208769
2280.755511
4152 2018-04-04 21:00:00          15.108275         16.302326
2505.582356

      fault  Attack  Attack_Type    Power_GAM  Rotor_Speed_GAM
4148      0       1            2  2042.085257        15.163695
4149      0       1            2  2046.255728        15.164181
4150      0       1            2  2045.192830        15.172759
4151      0       1            2  2042.631927        15.154886
4152      0       1            2  2044.731342        15.171916
```

```
# Remove rows with 'fault' equal to 1 and 'Power (kW)' less than 0
manipulated_data_2 = manipulated_data[(manipulated_data['fault'] != 1)
& (manipulated_data['Power (kW)'] >= 0)]
```

```
# Reset index if you want consecutive integer indices
manipulated_data_2.reset_index(drop=True, inplace=True)


# Calculate the absolute differences for Power and Rotor Speed
manipulated_data_2['Power_GAM_diff'] = abs(manipulated_data_2['Power
(kW)'] - manipulated_data_2['Power_GAM'])
manipulated_data_2['Rotor_Speed_GAM_diff'] =
abs(manipulated_data_2['Rotor speed (RPM)'] -
manipulated_data_2['Rotor_Speed_GAM'])

manipulated_data_2.head()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_8440\3963136.py:9:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  manipulated_data_2['Power_GAM_diff'] = abs(manipulated_data_2['Power
(kW)'] - manipulated_data_2['Power_GAM'])
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_8440\3963136.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  manipulated_data_2['Rotor_Speed_GAM_diff'] =
abs(manipulated_data_2['Rotor speed (RPM)'] -
manipulated_data_2['Rotor_Speed_GAM'])
```

```
      # Date and time  Rotor speed (RPM)  Wind speed (m/s)  Power (kW)
fault  \
0 2018-03-09 13:20:00           7.882541          4.473541  129.957794
0
1 2018-03-09 14:00:00           8.603662          4.603531  156.682816
0
2 2018-03-09 14:10:00           8.843963          4.735680  192.660599
0
3 2018-03-09 14:20:00          10.003763          5.606801  307.369751
0
4 2018-03-09 14:30:00           9.433234          5.219706  249.074280
0

   Attack  Attack_Type   Power_GAM  Rotor_Speed_GAM  Power_GAM_diff  \
0       0            0  170.179821         8.772313       40.222027
```

```
1       0            0  186.643491         8.913979       29.960675
2       0            0  204.292434         9.064989       11.631835
3       0            0  352.247796        10.319782       44.878045
4       0            0  278.765591         9.697830       29.691312
```

```
   Rotor_Speed_GAM_diff
0              0.889772
1              0.310317
2              0.221026
3              0.316019
4              0.264596
```

LSTM with GAM

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import EarlyStopping
import h5py

# Split the data into training and testing sets
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power_GAM_diff',
'Rotor_Speed_GAM_diff']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape the input data to include the sequence length
sequence_length = 10
X_train_reshaped = []
y_train_reshaped = []
for i in range(sequence_length, len(X_train_scaled)):
    X_train_reshaped.append(X_train_scaled[i - sequence_length:i])
    y_train_reshaped.append(y_train[i])  # Shift the labels by
sequence_length
```

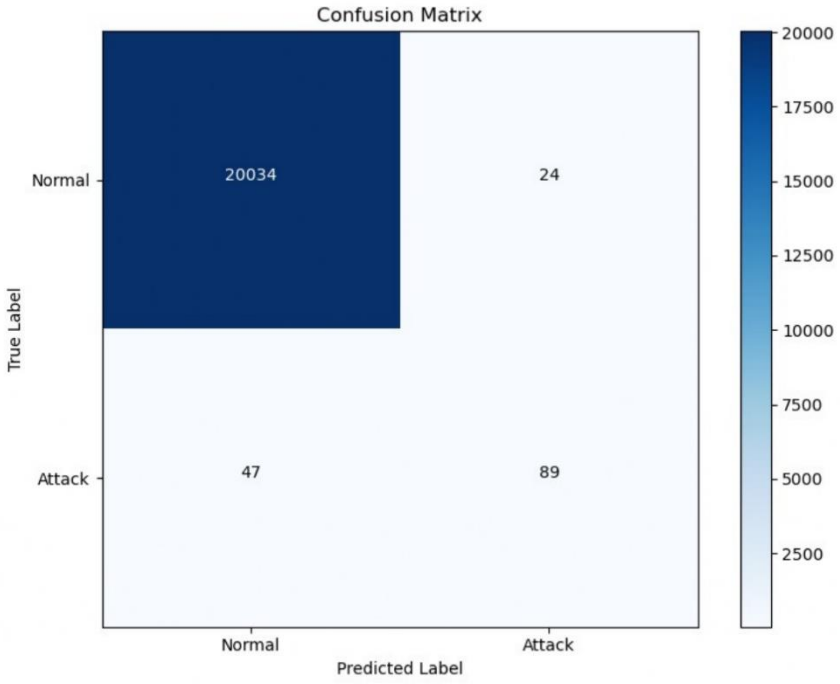
```
X_train_reshaped = np.array(X_train_reshaped)
y_train_reshaped = np.array(y_train_reshaped)

X_test_reshaped = []
y_test_reshaped = []
for i in range(sequence_length, len(X_test_scaled)):
    X_test_reshaped.append(X_test_scaled[i - sequence_length:i])
    y_test_reshaped.append(y_test[i])  # Shift the labels by
sequence_length
X_test_reshaped = np.array(X_test_reshaped)
y_test_reshaped = np.array(y_test_reshaped)

# Build the LSTM model with additional layers and dropout
lstm_GAM = Sequential()
lstm_GAM.add(LSTM(128, input_shape=(sequence_length,
len(input_features)), return_sequences=True))
lstm_GAM.add(Dropout(0.2))
lstm_GAM.add(LSTM(64, return_sequences=True))
lstm_GAM.add(Dropout(0.2))
lstm_GAM.add(LSTM(32))
lstm_GAM.add(Dropout(0.2))
lstm_GAM.add(Dense(1, activation='sigmoid'))
lstm_GAM.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Define early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model with early stopping
lstm_GAM.fit(X_train_reshaped, y_train_reshaped, epochs=5,
batch_size=32, validation_split=0.2, callbacks=[early_stop])

lstm_GAM.save('lstm_GAM.h5')

# Evaluate the model on the testing data
loss, accuracy = lstm_GAM.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

Epoch 1/5
2999/2999 [==============================] - 148s 46ms/step - loss:
0.0465 - accuracy: 0.9927 - val_loss: 0.0484 - val_accuracy: 0.9917
Epoch 2/5
2999/2999 [==============================] - 193s 64ms/step - loss:
0.0438 - accuracy: 0.9928 - val_loss: 0.0482 - val_accuracy: 0.9917
Epoch 3/5
2999/2999 [==============================] - 199s 66ms/step - loss:
0.0341 - accuracy: 0.9935 - val_loss: 0.0255 - val_accuracy: 0.9947
Epoch 4/5
```

```
2999/2999 [==============================] - 205s 68ms/step - loss:
0.0274 - accuracy: 0.9951 - val_loss: 0.0250 - val_accuracy: 0.9950
Epoch 5/5
2999/2999 [==============================] - 205s 68ms/step - loss:
0.0271 - accuracy: 0.9951 - val_loss: 0.0239 - val_accuracy: 0.9949

C:\Users\LENOVO\anaconda3\Lib\site-packages\keras\src\engine\
training.py:3079: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

632/632 [==============================] - 14s 22ms/step - loss:
0.0169 - accuracy: 0.9965
Test Loss: 0.0169
Test Accuracy: 0.9965
```

```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# ... (previous code remains the same)

# Evaluate the model on the testing data
loss, accuracy = lstm_GAM.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Make predictions on the testing data
y_pred_prob = lstm_GAM.predict(X_test_reshaped)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate the confusion matrix
cm = confusion_matrix(y_test_reshaped, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Attack'])
plt.yticks(tick_marks, ['Normal', 'Attack'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
```

```
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

```
632/632 [==============================] - 13s 21ms/step - loss:
0.0169 - accuracy: 0.9965
Test Loss: 0.0169
Test Accuracy: 0.9965
632/632 [==============================] - 14s 19ms/step
```




```
# Classification Report
report = classification_report(y_test_reshaped, y_pred,
```

```
zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     20058
           1       0.79      0.65      0.71       136

    accuracy                           1.00     20194
   macro avg       0.89      0.83      0.86     20194
weighted avg       1.00      1.00      1.00     20194
```

```
import pandas as pd

# Assuming you have the following variables defined:
# test_data, sequence_length, y_pred

# Calculate the number of rows to be removed based on the sequence
length
offset = sequence_length - 1  # This is the starting index for
sequences

# Ensure that you're considering the correct number of elements from
the end
test_data_with_predictions = test_data.iloc[offset:offset +
len(y_pred)].copy()

# Now add the predictions
test_data_with_predictions['Predicted Attack'] = y_pred

# Check to ensure the DataFrame and predictions align
assert len(test_data_with_predictions) == len(y_pred), "The lengths do
not match."

# Now, test_data_with_predictions includes the predictions aligned
with the data

# Filter for rows where actual attack is 1 and predicted attack is 1
true_negatives =
test_data_with_predictions[(test_data_with_predictions['Attack'] == 1)
& (test_data_with_predictions['Predicted Attack'] == 1)]

# Display these rows
true_negatives
```

```
           # Date and time  Rotor speed (RPM)  Wind speed (m/s)
Power (kW)  \
120910 2021-01-09 01:30:00           8.144213          0.000000
57.081221
120911 2021-01-09 01:40:00           8.174227          0.000000
```

```
8.416295
122791 2021-01-25 18:30:00          25.001111          9.010764
1499.982593
122792 2021-01-25 18:40:00          28.008784          9.375109
1789.158307
122793 2021-01-25 18:50:00          27.189187          9.788589
1977.540448
...                    ...                ...               ...
...
137624 2021-05-25 01:40:00          14.831943         14.472197
1185.681259
137625 2021-05-25 01:50:00          14.824996         14.002925
1167.068332
137626 2021-05-25 02:00:00          15.002507         13.192070
1253.369016
137627 2021-05-25 02:10:00          15.013282         13.061197
1254.039099
137628 2021-05-25 02:20:00          15.102826         13.977983
1342.417883

        fault  Attack  Attack_Type    Power_GAM  Rotor_Speed_GAM  \
120910      0       1            4    -3.124686        -0.018227
120911      0       1            4    -3.124686        -0.018227
122791      0       1            1  1476.735374        14.984718
122792      0       1            1  1613.201741        15.079253
122793      0       1            1  1749.959086        15.122727
...       ...     ...          ...          ...              ...
137624      0       1            1  2038.099777        15.146799
137625      0       1            1  2040.231888        15.139723
137626      0       1            1  2043.634414        15.135962
137627      0       1            1  2042.655280        15.138226
137628      0       1            1  2040.425189        15.139343

        Power_GAM_diff  Rotor_Speed_GAM_diff  Predicted Attack
120910       60.205907              8.162440                 1
120911       11.540981              8.192454                 1
122791       23.247219             10.016393                 1
122792      175.956566             12.929530                 1
122793      227.581362             12.066459                 1
...                ...                   ...               ...
137624      852.418518              0.314856                 1
137625      873.163556              0.314727                 1
137626      790.265398              0.133456                 1
137627      788.616181              0.124944                 1
137628      698.007305              0.036517                 1

[97 rows x 12 columns]
```

LSTM without GAM

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import EarlyStopping

# Split the data into training and testing sets
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power (kW)', 'Rotor speed
(RPM)']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape the input data to include the sequence length
sequence_length = 10
X_train_reshaped = []
y_train_reshaped = []
for i in range(sequence_length, len(X_train_scaled)):
    X_train_reshaped.append(X_train_scaled[i - sequence_length:i])
    y_train_reshaped.append(y_train[i])  # Shift the labels by
sequence_length
X_train_reshaped = np.array(X_train_reshaped)
y_train_reshaped = np.array(y_train_reshaped)

X_test_reshaped = []
y_test_reshaped = []
for i in range(sequence_length, len(X_test_scaled)):
    X_test_reshaped.append(X_test_scaled[i - sequence_length:i])
    y_test_reshaped.append(y_test[i])  # Shift the labels by
sequence_length
X_test_reshaped = np.array(X_test_reshaped)
y_test_reshaped = np.array(y_test_reshaped)

# Build the LSTM model with additional layers and dropout
lstm_model = Sequential()
```

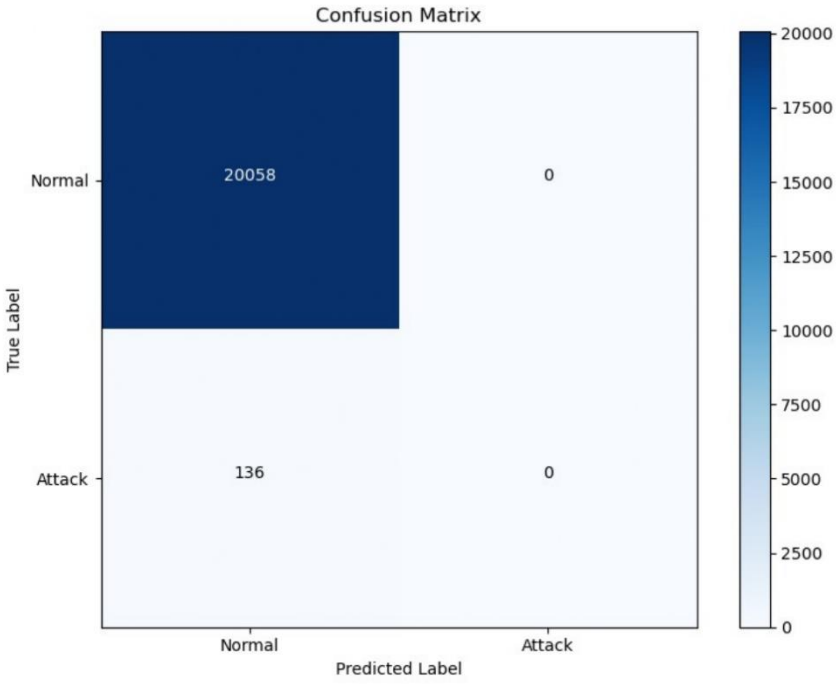
```
lstm_model.add(LSTM(128, input_shape=(sequence_length,
len(input_features)), return_sequences=True))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(64, return_sequences=True))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(32))
lstm_model.add(Dropout(0.2))
lstm_model.add(Dense(1, activation='sigmoid'))
lstm_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Define early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model with early stopping
lstm_model.fit(X_train_reshaped, y_train_reshaped, epochs=5,
batch_size=32, validation_split=0.2, callbacks=[early_stop])

# Evaluate the model on the testing data
loss, accuracy = lstm_model.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

```
Epoch 1/5
2999/2999 [==============================] - 184s 57ms/step - loss:
0.0463 - accuracy: 0.9925 - val_loss: 0.0489 - val_accuracy: 0.9917
Epoch 2/5
2999/2999 [==============================] - 204s 68ms/step - loss:
0.0440 - accuracy: 0.9928 - val_loss: 0.0484 - val_accuracy: 0.9917
Epoch 3/5
2999/2999 [==============================] - 190s 63ms/step - loss:
0.0438 - accuracy: 0.9928 - val_loss: 0.0487 - val_accuracy: 0.9917
Epoch 4/5
2999/2999 [==============================] - 145s 48ms/step - loss:
0.0440 - accuracy: 0.9928 - val_loss: 0.0482 - val_accuracy: 0.9917
Epoch 5/5
2999/2999 [==============================] - 175s 58ms/step - loss:
0.0436 - accuracy: 0.9928 - val_loss: 0.0490 - val_accuracy: 0.9917
632/632 [==============================] - 15s 23ms/step - loss:
0.0406 - accuracy: 0.9933
Test Loss: 0.0406
Test Accuracy: 0.9933
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt
```

```
# ... (previous code remains the same)

# Evaluate the model on the testing data
loss, accuracy = lstm_model.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Make predictions on the testing data
y_pred_prob = lstm_model.predict(X_test_reshaped)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate the confusion matrix
cm = confusion_matrix(y_test_reshaped, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Attack'])
plt.yticks(tick_marks, ['Normal', 'Attack'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

```
632/632 [==============================] - 13s 21ms/step - loss:
0.0406 - accuracy: 0.9933
Test Loss: 0.0406
Test Accuracy: 0.9933
632/632 [==============================] - 15s 21ms/step
```




```
# Classification Report
report = classification_report(y_test_reshaped, y_pred,
zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     20058
           1       1.00      0.00      0.00       136

    accuracy                           0.99     20194
   macro avg       1.00      0.50      0.50     20194
weighted avg       0.99      0.99      0.99     20194
```

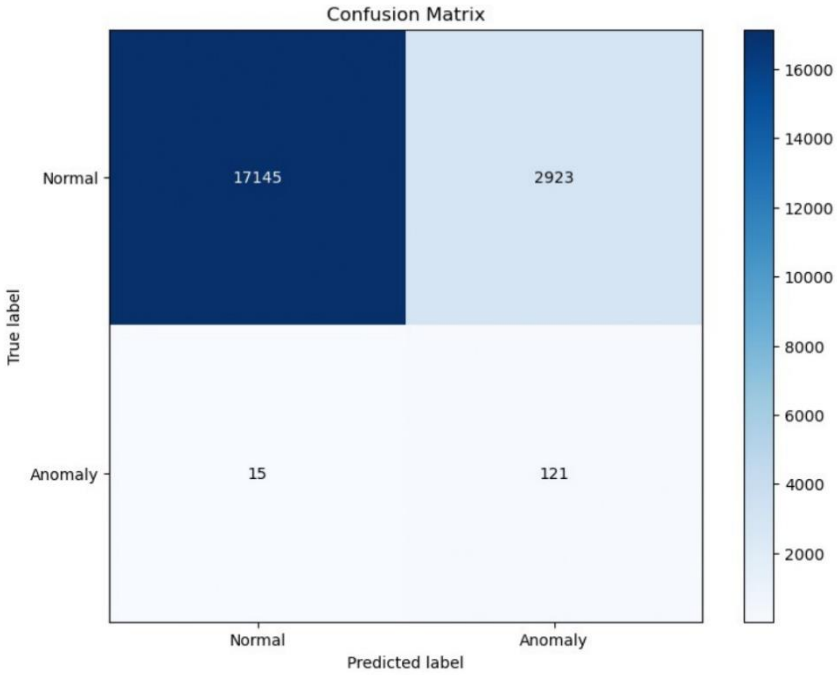
Isolation forest with GAM

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import IsolationForest
```

```
# Split the data into training and testing sets
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power_GAM_diff',
'Rotor_Speed_GAM_diff']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an Isolation Forest model
IF_GAM = IsolationForest(n_estimators=100, contamination=0.1,
random_state=42)

# Train the model
IF_GAM.fit(X_train_scaled)

# Make predictions on the training and testing data
y_train_pred = IF_GAM.predict(X_train_scaled)
y_test_pred = IF_GAM.predict(X_test_scaled)

# Convert the predictions to binary labels (-1 for anomalies, 1 for
normal)
y_train_pred = np.where(y_train_pred == -1, 1, 0)
y_test_pred = np.where(y_test_pred == -1, 1, 0)

# Evaluate the model on the training data
train_accuracy = (y_train_pred == y_train).mean()
print(f'Training Accuracy: {train_accuracy:.4f}')

# Evaluate the model on the testing data
test_accuracy = (y_test_pred == y_test).mean()
print(f'Testing Accuracy: {test_accuracy:.4f}')

Training Accuracy: 0.9026
Testing Accuracy: 0.8546
```

```
# Make predictions on the testing data
y_test_pred = IF_GAM.predict(X_test_scaled)

# Convert the predictions to binary labels (-1 for anomalies, 1 for
normal)
y_test_pred = np.where(y_test_pred == -1, 1, 0)

# Evaluate the model on the testing data
test_accuracy = (y_test_pred == y_test).mean()
print(f'Testing Accuracy: {test_accuracy:.4f}')

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Anomaly'])
plt.yticks(tick_marks, ['Normal', 'Anomaly'])

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Testing Accuracy: 0.8546
```




```
# Classification Report
report = classification_report(y_test, y_test_pred, zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      0.85      0.92     20068
           1       0.04      0.89      0.08       136

    accuracy                           0.85     20204
   macro avg       0.52      0.87      0.50     20204
weighted avg       0.99      0.85      0.92     20204
```

Isolation forest without GAM

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import IsolationForest

# Split the data into training and testing sets
```

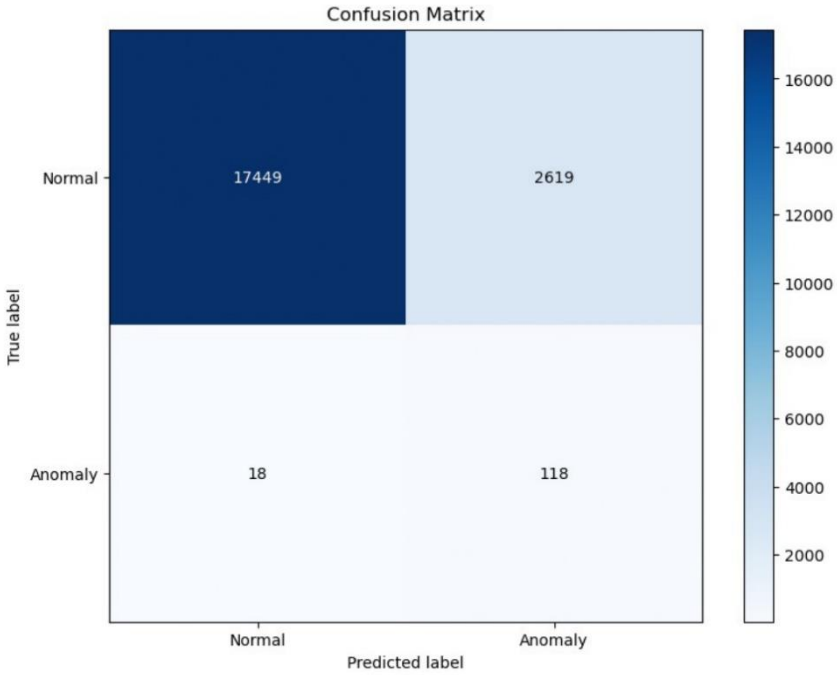
```
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power (kW)', 'Rotor speed
(RPM)']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an Isolation Forest model
IF_model = IsolationForest(n_estimators=100, contamination=0.1,
random_state=42)

# Train the model
IF_model.fit(X_train_scaled)

# Make predictions on the training and testing data
y_train_pred = IF_model.predict(X_train_scaled)
y_test_pred = IF_model.predict(X_test_scaled)

# Convert the predictions to binary labels (-1 for anomalies, 1 for
normal)
y_train_pred = np.where(y_train_pred == -1, 1, 0)
y_test_pred = np.where(y_test_pred == -1, 1, 0)

# Evaluate the model on the training data
train_accuracy = (y_train_pred == y_train).mean()
print(f'Training Accuracy: {train_accuracy:.4f}')

# Evaluate the model on the testing data
test_accuracy = (y_test_pred == y_test).mean()
print(f'Testing Accuracy: {test_accuracy:.4f}')
```

```
Training Accuracy: 0.9024
Testing Accuracy: 0.8695
```

```
# Make predictions on the testing data
y_test_pred = IF_model.predict(X_test_scaled)
```

```
# Convert the predictions to binary labels (-1 for anomalies, 1 for
normal)
y_test_pred = np.where(y_test_pred == -1, 1, 0)

# Evaluate the model on the testing data
test_accuracy = (y_test_pred == y_test).mean()
print(f'Testing Accuracy: {test_accuracy:.4f}')

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Anomaly'])
plt.yticks(tick_marks, ['Normal', 'Anomaly'])

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 ha="center", va="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Testing Accuracy: 0.8695
```




```
# Classification Report
report = classification_report(y_test, y_test_pred, zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      0.87      0.93     20068
           1       0.04      0.87      0.08       136

    accuracy                           0.87     20204
   macro avg       0.52      0.87      0.51     20204
weighted avg       0.99      0.87      0.92     20204
```

CNN with GAM

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from keras.callbacks import EarlyStopping
```

```
# Split the data into training and testing sets
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power_GAM_diff',
'Rotor_Speed_GAM_diff']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape the input data to include the sequence length
sequence_length = 10
X_train_reshaped = []
y_train_reshaped = []
for i in range(sequence_length, len(X_train_scaled)):
    X_train_reshaped.append(X_train_scaled[i - sequence_length:i])
    y_train_reshaped.append(y_train[i])  # Shift the labels by
sequence_length
X_train_reshaped = np.array(X_train_reshaped)
y_train_reshaped = np.array(y_train_reshaped)

X_test_reshaped = []
y_test_reshaped = []
for i in range(sequence_length, len(X_test_scaled)):
    X_test_reshaped.append(X_test_scaled[i - sequence_length:i])
    y_test_reshaped.append(y_test[i])  # Shift the labels by
sequence_length
X_test_reshaped = np.array(X_test_reshaped)
y_test_reshaped = np.array(y_test_reshaped)

# Build the CNN model with additional layers and dropout
cnn_GAM = Sequential()
cnn_GAM.add(Conv1D(64, 3, activation='relu',
input_shape=(sequence_length, len(input_features))))
cnn_GAM.add(MaxPooling1D(pool_size=2))
cnn_GAM.add(Conv1D(32, 3, activation='relu'))
cnn_GAM.add(MaxPooling1D(pool_size=2))
```

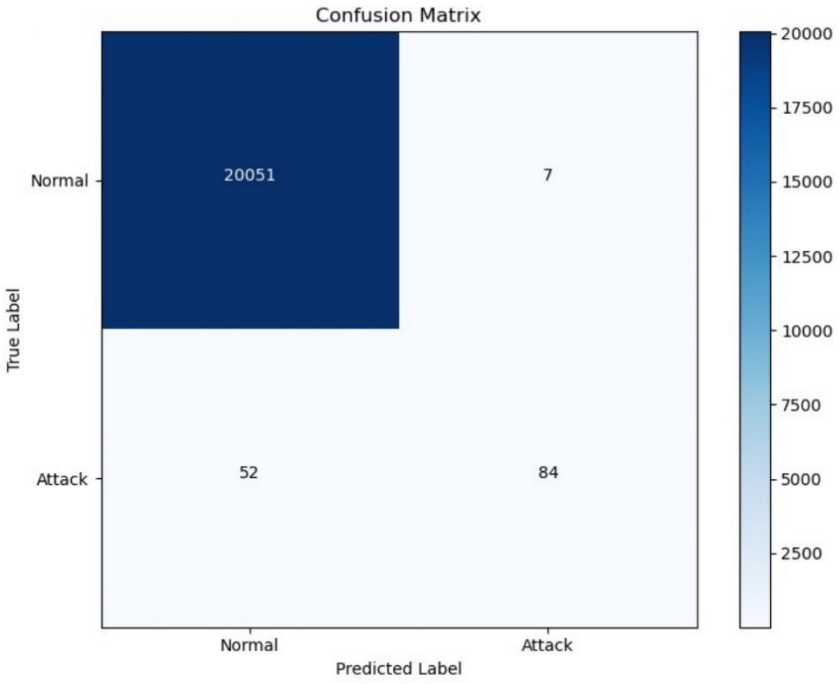
```
cnn_GAM.add(Flatten())
cnn_GAM.add(Dense(128, activation='relu'))
cnn_GAM.add(Dropout(0.2))
cnn_GAM.add(Dense(1, activation='sigmoid'))
cnn_GAM.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Define early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model with early stopping
cnn_GAM.fit(X_train_reshaped, y_train_reshaped, epochs=5,
batch_size=32, validation_split=0.2, callbacks=[early_stop])

cnn_GAM.save('cnn_GAM.h5')

# Evaluate the model on the testing data
loss, accuracy = cnn_GAM.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

```
Epoch 1/5
2999/2999 [==============================] - 23s 7ms/step - loss:
0.0387 - accuracy: 0.9937 - val_loss: 0.0276 - val_accuracy: 0.9945
Epoch 2/5
2999/2999 [==============================] - 19s 6ms/step - loss:
0.0287 - accuracy: 0.9949 - val_loss: 0.0268 - val_accuracy: 0.9945
Epoch 3/5
2999/2999 [==============================] - 19s 6ms/step - loss:
0.0280 - accuracy: 0.9951 - val_loss: 0.0248 - val_accuracy: 0.9949
Epoch 4/5
2999/2999 [==============================] - 19s 6ms/step - loss:
0.0268 - accuracy: 0.9954 - val_loss: 0.0243 - val_accuracy: 0.9952
Epoch 5/5
2999/2999 [==============================] - 21s 7ms/step - loss:
0.0261 - accuracy: 0.9955 - val_loss: 0.0241 - val_accuracy: 0.9957
  1/632 [..............................] - ETA: 19s - loss: 0.0011 -
accuracy: 1.0000

C:\Users\LENOVO\anaconda3\Lib\site-packages\keras\src\engine\
training.py:3079: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

632/632 [==============================] - 2s 3ms/step - loss: 0.0167
- accuracy: 0.9971
```

```
Test Loss: 0.0167
Test Accuracy: 0.9971
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# ... (previous code remains the same)

# Evaluate the model on the testing data
loss, accuracy = cnn_GAM.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Make predictions on the testing data
y_pred_prob = cnn_GAM.predict(X_test_reshaped)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate the confusion matrix
cm = confusion_matrix(y_test_reshaped, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Attack'])
plt.yticks(tick_marks, ['Normal', 'Attack'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

```
632/632 [==============================] - 2s 3ms/step - loss: 0.0167
- accuracy: 0.9971
Test Loss: 0.0167
Test Accuracy: 0.9971
632/632 [==============================] - 2s 3ms/step
```




```
# Classification Report
report = classification_report(y_test_reshaped, y_pred,
zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     20058
           1       0.92      0.62      0.74       136

    accuracy                           1.00     20194
   macro avg       0.96      0.81      0.87     20194
weighted avg       1.00      1.00      1.00     20194
```

CNN without GAM

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
```

```
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from keras.callbacks import EarlyStopping

# Split the data into training and testing sets
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power (kW)', 'Rotor speed
(RPM)']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape the input data to include the sequence length
sequence_length = 10
X_train_reshaped = []
y_train_reshaped = []
for i in range(sequence_length, len(X_train_scaled)):
    X_train_reshaped.append(X_train_scaled[i - sequence_length:i])
    y_train_reshaped.append(y_train[i])  # Shift the labels by
sequence_length
X_train_reshaped = np.array(X_train_reshaped)
y_train_reshaped = np.array(y_train_reshaped)

X_test_reshaped = []
y_test_reshaped = []
for i in range(sequence_length, len(X_test_scaled)):
    X_test_reshaped.append(X_test_scaled[i - sequence_length:i])
    y_test_reshaped.append(y_test[i])  # Shift the labels by
sequence_length
X_test_reshaped = np.array(X_test_reshaped)
y_test_reshaped = np.array(y_test_reshaped)

# Build the CNN model with additional layers and dropout
cnn_model = Sequential()
cnn_model.add(Conv1D(64, 3, activation='relu',
input_shape=(sequence_length, len(input_features))))
cnn_model.add(MaxPooling1D(pool_size=2))
```

```
cnn_model.add(Conv1D(32, 3, activation='relu'))
cnn_model.add(MaxPooling1D(pool_size=2))
cnn_model.add(Flatten())
cnn_model.add(Dense(128, activation='relu'))
cnn_model.add(Dropout(0.2))
cnn_model.add(Dense(1, activation='sigmoid'))
cnn_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Define early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model with early stopping
cnn_model.fit(X_train_reshaped, y_train_reshaped, epochs=5,
batch_size=32, validation_split=0.2, callbacks=[early_stop])

cnn_model.save('cnn_model.h5')

# Evaluate the model on the testing data
loss, accuracy = cnn_model.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

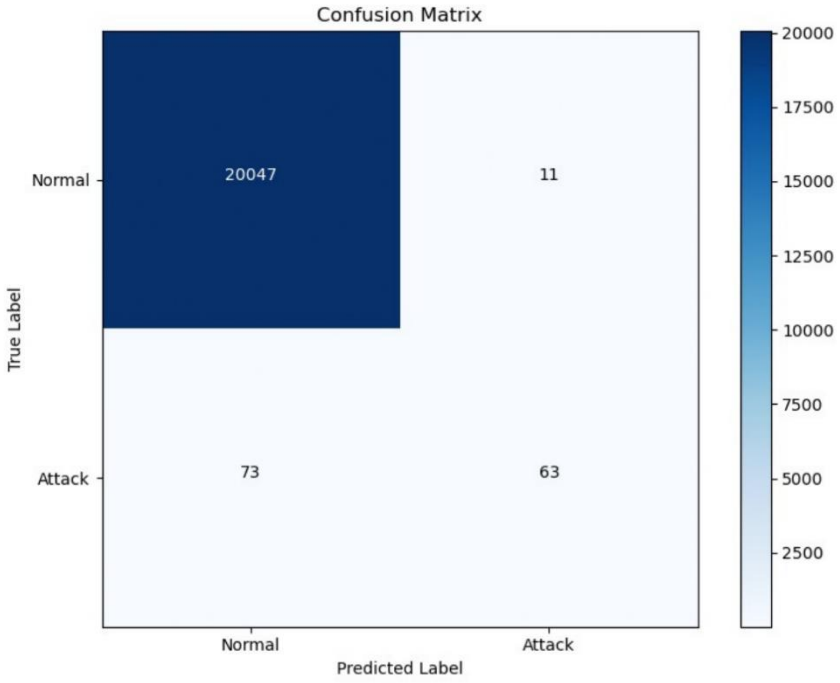
```
Epoch 1/5
2999/2999 [==============================] - 23s 7ms/step - loss:
0.0500 - accuracy: 0.9924 - val_loss: 0.0501 - val_accuracy: 0.9917
Epoch 2/5
2999/2999 [==============================] - 22s 7ms/step - loss:
0.0381 - accuracy: 0.9932 - val_loss: 0.0362 - val_accuracy: 0.9930
Epoch 3/5
2999/2999 [==============================] - 22s 7ms/step - loss:
0.0337 - accuracy: 0.9941 - val_loss: 0.0338 - val_accuracy: 0.9935
Epoch 4/5
2999/2999 [==============================] - 20s 7ms/step - loss:
0.0312 - accuracy: 0.9943 - val_loss: 0.0294 - val_accuracy: 0.9942
Epoch 5/5
2999/2999 [==============================] - 20s 7ms/step - loss:
0.0293 - accuracy: 0.9946 - val_loss: 0.0275 - val_accuracy: 0.9943
  1/632 [..............................] - ETA: 19s - loss: 0.0021 -
accuracy: 1.0000

C:\Users\LENOVO\anaconda3\Lib\site-packages\keras\src\engine\
training.py:3079: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```
632/632 [==============================] - 2s 3ms/step - loss: 0.0221
- accuracy: 0.9958
Test Loss: 0.0221
Test Accuracy: 0.9958
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# ... (previous code remains the same)

# Evaluate the model on the testing data
loss, accuracy = cnn_model.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Make predictions on the testing data
y_pred_prob = cnn_model.predict(X_test_reshaped)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate the confusion matrix
cm = confusion_matrix(y_test_reshaped, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Attack'])
plt.yticks(tick_marks, ['Normal', 'Attack'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

```
632/632 [==============================] - 2s 3ms/step - loss: 0.0221
- accuracy: 0.9958
Test Loss: 0.0221
```

```
Test Accuracy: 0.9958
632/632 [==============================] - 2s 3ms/step
```




```
# Classification Report
report = classification_report(y_test_reshaped, y_pred,
zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     20058
           1       0.85      0.46      0.60       136

    accuracy                           1.00     20194
   macro avg       0.92      0.73      0.80     20194
weighted avg       1.00      1.00      1.00     20194
```

ANN with GAM

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping

# Split the data into training and testing sets
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power_GAM_diff',
'Rotor_Speed_GAM_diff']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the ANN model with additional layers and dropout
ann_GAM = Sequential()
ann_GAM.add(Dense(128, activation='relu',
input_shape=(len(input_features),)))
ann_GAM.add(Dropout(0.2))
ann_GAM.add(Dense(64, activation='relu'))
ann_GAM.add(Dropout(0.2))
ann_GAM.add(Dense(32, activation='relu'))
ann_GAM.add(Dropout(0.2))
ann_GAM.add(Dense(1, activation='sigmoid'))
ann_GAM.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Define early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model with early stopping
ann_GAM.fit(X_train_scaled, y_train, epochs=5, batch_size=32,
validation_split=0.2, callbacks=[early_stop])

ann_GAM.save('ann_GAM.h5')
```

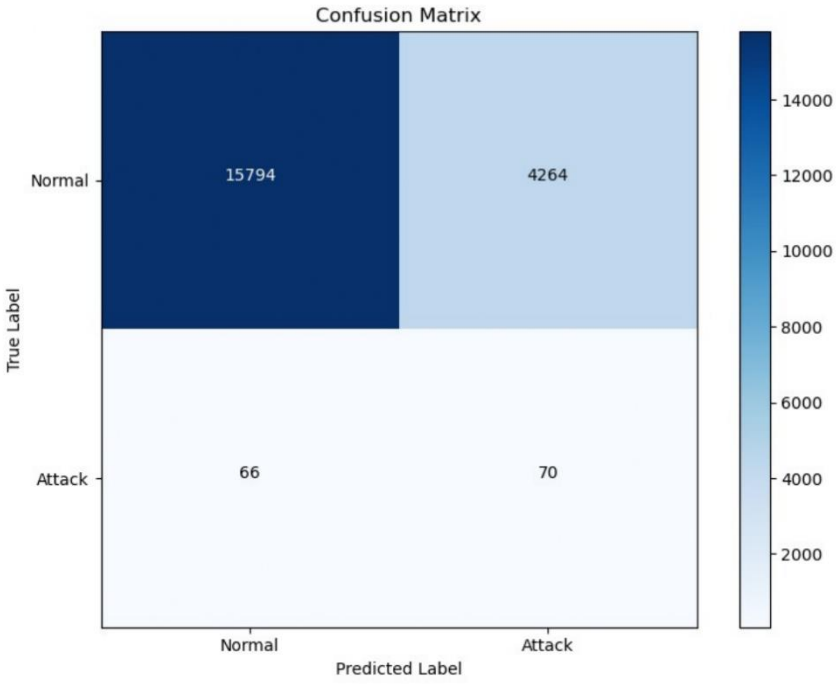
```
# Evaluate the model on the testing data
loss, accuracy = ann_GAM.evaluate(X_test_scaled, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

```
Epoch 1/5
3000/3000 [==============================] - 17s 5ms/step - loss:
0.0347 - accuracy: 0.9947 - val_loss: 0.0217 - val_accuracy: 0.9955
Epoch 2/5
3000/3000 [==============================] - 16s 5ms/step - loss:
0.0261 - accuracy: 0.9955 - val_loss: 0.0215 - val_accuracy: 0.9956
Epoch 3/5
3000/3000 [==============================] - 16s 5ms/step - loss:
0.0256 - accuracy: 0.9955 - val_loss: 0.0202 - val_accuracy: 0.9957
Epoch 4/5
3000/3000 [==============================] - 15s 5ms/step - loss:
0.0248 - accuracy: 0.9956 - val_loss: 0.0195 - val_accuracy: 0.9960
Epoch 5/5
3000/3000 [==============================] - 15s 5ms/step - loss:
0.0248 - accuracy: 0.9956 - val_loss: 0.0196 - val_accuracy: 0.9963
  1/632 [..............................] - ETA: 19s - loss: 0.0012 -
accuracy: 1.0000

C:\Users\LENOVO\anaconda3\Lib\site-packages\keras\src\engine\
training.py:3079: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

632/632 [==============================] - 2s 3ms/step - loss: 0.0147
- accuracy: 0.9970
Test Loss: 0.0147
Test Accuracy: 0.9970
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# ... (previous code remains the same)

# Evaluate the model on the testing data
loss, accuracy = ann_GAM.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Reshape the input data
X_test_reshaped = X_test_reshaped[:, -1, :]  # Extract the last time
```

```
step
X_test_reshaped = X_test_reshaped.reshape(-1, 3)

# Make predictions on the testing data
y_pred_prob = ann_GAM.predict(X_test_reshaped)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate the confusion matrix
cm = confusion_matrix(y_test_reshaped, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Attack'])
plt.yticks(tick_marks, ['Normal', 'Attack'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

```
632/632 [==============================] - 4s 5ms/step - loss: 0.4172
- accuracy: 0.7836
Test Loss: 0.4172
Test Accuracy: 0.7836
632/632 [==============================] - 2s 2ms/step
```




```
# Classification Report
report = classification_report(y_test_reshaped, y_pred,
zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      0.79      0.88     20058
           1       0.02      0.51      0.03       136

    accuracy                           0.79     20194
   macro avg       0.51      0.65      0.46     20194
weighted avg       0.99      0.79      0.87     20194
```

ANN without GAM

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
```

```
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping

# Split the data into training and testing sets
train_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year < 2021]
test_data = manipulated_data_2[manipulated_data_2['# Date and
time'].dt.year == 2021]

# Define the input features and target
input_features = ['Wind speed (m/s)', 'Power (kW)', 'Rotor speed
(RPM)']
target = 'Attack'

# Prepare the training and testing data
X_train = train_data[input_features].values
y_train = train_data[target].values
X_test = test_data[input_features].values
y_test = test_data[target].values

# Scale the input features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the ANN model with additional layers and dropout
ann_model = Sequential()
ann_model.add(Dense(128, activation='relu',
input_shape=(len(input_features),)))
ann_model.add(Dropout(0.2))
ann_model.add(Dense(64, activation='relu'))
ann_model.add(Dropout(0.2))
ann_model.add(Dense(32, activation='relu'))
ann_model.add(Dropout(0.2))
ann_model.add(Dense(1, activation='sigmoid'))
ann_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Define early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

# Train the model with early stopping
ann_model.fit(X_train_scaled, y_train, epochs=5, batch_size=32,
validation_split=0.2, callbacks=[early_stop])

ann_model.save('ann_model.h5')

# Evaluate the model on the testing data
loss, accuracy = ann_model.evaluate(X_test_scaled, y_test)
```

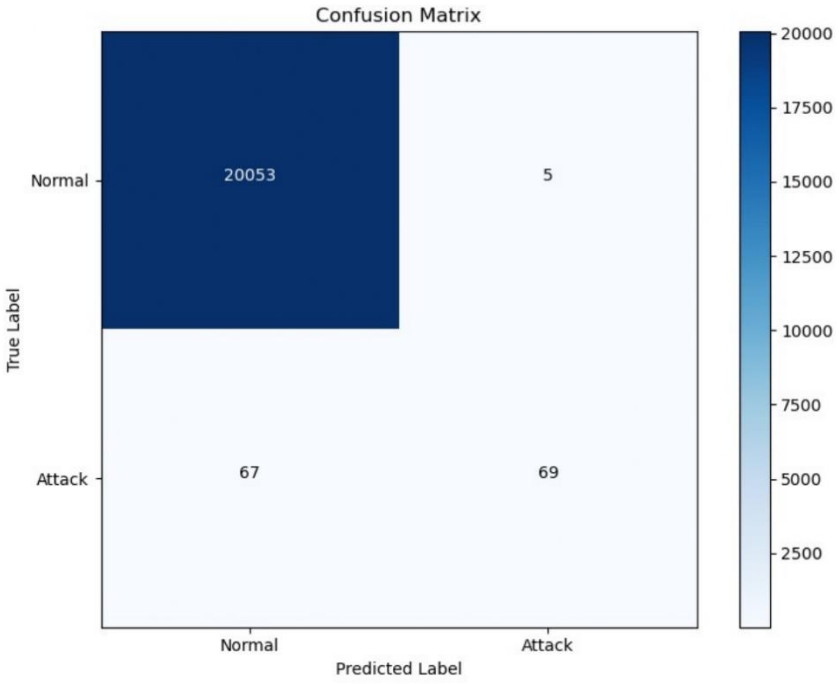
```
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

```
Epoch 1/5
3000/3000 [==============================] - 19s 6ms/step - loss:
0.0450 - accuracy: 0.9928 - val_loss: 0.0242 - val_accuracy: 0.9957
Epoch 2/5
3000/3000 [==============================] - 16s 5ms/step - loss:
0.0273 - accuracy: 0.9957 - val_loss: 0.0208 - val_accuracy: 0.9959
Epoch 3/5
3000/3000 [==============================] - 16s 5ms/step - loss:
0.0246 - accuracy: 0.9962 - val_loss: 0.0179 - val_accuracy: 0.9970
Epoch 4/5
3000/3000 [==============================] - 16s 5ms/step - loss:
0.0237 - accuracy: 0.9965 - val_loss: 0.0185 - val_accuracy: 0.9970
Epoch 5/5
3000/3000 [==============================] - 16s 5ms/step - loss:
0.0231 - accuracy: 0.9964 - val_loss: 0.0171 - val_accuracy: 0.9976
  1/632 [..............................] - ETA: 19s - loss: 0.0017 -
accuracy: 1.0000

C:\Users\LENOVO\anaconda3\Lib\site-packages\keras\src\engine\
training.py:3079: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

632/632 [==============================] - 2s 3ms/step - loss: 0.0137
- accuracy: 0.9969
Test Loss: 0.0137
Test Accuracy: 0.9969
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

# ... (previous code remains the same)

# Evaluate the model on the testing data
loss, accuracy = ann_model.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')

# Make predictions on the testing data
y_pred_prob = ann_model.predict(X_test_reshaped)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

# Calculate the confusion matrix
```

```
cm = confusion_matrix(y_test_reshaped, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap='Blues', interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Normal', 'Attack'])
plt.yticks(tick_marks, ['Normal', 'Attack'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')

# Add numbers to the confusion matrix
thresh = cm.max() / 2.0
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

```
632/632 [==============================] - 2s 3ms/step - loss: 0.0196
- accuracy: 0.9964
Test Loss: 0.0196
Test Accuracy: 0.9964
632/632 [==============================] - 2s 2ms/step
```




```
# Classification Report
report = classification_report(y_test_reshaped, y_pred,
zero_division=1)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     20058
           1       0.93      0.51      0.66       136

    accuracy                           1.00     20194
   macro avg       0.96      0.75      0.83     20194
weighted avg       1.00      1.00      1.00     20194
```