VULNERABILITY ANALYSIS OF IOT SENSORS USING DEEP LEARNING TECHNOLOGY

ST. TERESA'S COLLEGE (AUTONOMOUS)
AFFILIATED TO MAHATMA GANDHI UNIVERSITY

PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree of

BCA
(CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

By

ASHWITHA SAJEEV – SB21BCA004
&
SHINU MARY JOHN – SB21BCA033

III DC BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

Under the guidance of
MS. VEENA ANTONY

DEPARTMENT OF BCA (CLOUD TECHNOLOGY & INFORMATION SYSTEM MANAGEMENT)
MARCH 2024

VULNERABILITY ANALYSIS OF IOT SENSORS USING DEEP LEARNING TECHNOLOGY

ST. TERESA'S COLLEGE (AUTONOMOUS)
AFFILIATED TO MAHATMA GANDHI UNIVERSITY

PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree of

BCA
(CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

By

ASHWITHA SAJEEV – SB21BCA004
&
SHINU MARY JOHN – SB21BCA033

III DC BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

Under the guidance of
MS. VEENA ANTONY

DEPARTMENT OF BCA (CLOUD TECHNOLOGY & INFORMATION SYSTEM MANAGEMENT)
MARCH 2024

DECLARATION

We, undersigned, hereby declare that the project report, **‘VULNERABILITY ANALYSIS OF IOT SENSORS USING DEEP LEARNING TECHNOLOGY**”, submitted for partial fulfilment of the requirements for the award of degree of BCA (Cloud Technology and Information Security Management) at St. Teresa’s College (Autonomous), Ernakulam (Affiliated to Mahatma Gandhi University), Kerala, is a Bonafede work done by us under the supervision of Ms. Veena Antony. This submission represents our ideas in our own words where ideas or words of others have not been included. We have adequately and accurately cited and referenced the sources. We also declare that we have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data idea fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Ernakulam
March 2024

Ashwitha Sajeev-SB21BCA004
Shinu Mary John-SB21BCA033

ST. TERESA'S COLLEGE (AUTONOMOUS), ERNAKULAM

BCA (CLOUD TECHNOLOGY AND INFORMATION SECURITY MANAGEMENT)

DEPARTMENT OF BCA (CLOUD TECHNOLOGY & INFORMATION SYSTEM MANAGEMENT)

THE FEAR OF THE LORD IS THE BEGINNING OF WISDOM

ST. TERESA'S COLLEGE
(AUTONOMOUS)
ESTD 1925

CERTIFICATE

This is to certify that the report entitled "**VULNERABILITY ANALYSIS OF IOT SENSORS USING DEEP LEARNING TECHNOLOGY**", submitted by Ashwitha Sajeev and Shinu Mary John to the Mahatma Gandhi University in partial fulfilment of the requirements for the award of the Degree of BCA (Cloud Technology and Information Security Management) is a Bonafede record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

[illegible]

Ms. ARCHANA MENON P
Head of the department

[illegible]

Ms. VEENA ANTONY
Internal Supervisor

ERNAKULAM
DEPT. OF COMPUTERS
ST. TERESA'S COLLEGE

[illegible] 25/[illegible]/24

External Supervisor

ACKNOWLEDGEMENT

First and foremost, we thank God Almighty for his blessings. We take this opportunity to express our gratitude to all those who helped us in completing this project successfully. I wish to express our sincere gratitude to the **Manager Rev. Dr. Sr. Vinitha CSST** and the Principal **Dr. Alphonsa Vijaya Joseph** for providing all the facilities.

We express our sincere gratitude towards the Head of the department **Ms. Archana Menon P** for her support. We deeply express sincere thanks to our project guide **Ms. Veena Antony** for her proper guidance and support throughout the project work.

We are indebted to our beloved teachers whose cooperation and suggestions throughout the project helped immensely. We thank all our friends and classmates for their support.

We convey our hearty thanks to our parents for their moral support, suggestions and encouragement.

ABSTRACT

The proliferation of Internet of Things (IoT) sensors has opened doors to novel applications but also introduced new security challenges. These sensors are susceptible to vulnerabilities that could be exploited for malicious purposes. This study delves into the effectiveness of Deep Learning (DL) algorithms in analyzing vulnerabilities within IoT sensors. We propose a framework centred on Convolutional Neural Networks (CNNs) for intrusion detection within IoT networks. The framework investigates three distinct CNN architectures to determine the most accurate approach for vulnerability analysis:

1. **Baseline CNN:** This model employs a standard CNN architecture for network traffic classification.
2. **CNN-BiLSTM:** This variation incorporates Bidirectional Long Short-Term Memory (BiLSTM) layers after the convolutional layers. BiLSTM layers excel at capturing sequential dependencies within data, potentially leading to higher accuracy by considering the temporal nature of network traffic.
3. **CNN-GRU:** This model utilizes Gated Recurrent Unit (GRU) layers instead of BiLSTMs. GRUs offer similar functionality but with a simpler architecture.

The KDD Cup 99 dataset, a well-established benchmark for intrusion detection, serves as the foundation for training and evaluating these models. The code implements essential data preprocessing steps:

- **Label Encoding:** Categorical features are converted into numerical representations for compatibility with the CNN architecture.
- **Feature Standardization:** Numerical features are normalized using standardization techniques to ensure all features contribute equally during training.
- **Reshaping for CNN Layer:** The data is reshaped into a format suitable for the CNN's convolutional operations.

Following preprocessing, each model undergoes training and evaluation. Their performance is primarily compared based on their accuracy in classifying diverse network traffic patterns into distinct attack categories.

By meticulously comparing the accuracy of these three CNN architectures, this study aims to identify the most accurate approach for vulnerability analysis in IoT sensor networks. Additionally, the code demonstrates how to load a pre-trained model and make predictions on new, unseen sensor data. This practical example showcases the potential application of the most accurate DL model in real-world IoT security scenarios.

This research contributes to the ongoing exploration of DL for enhancing IoT security. By comparing the accuracy of CNN, CNN-BiLSTM, and CNN-GRU architectures in intrusion detection, we offer valuable insights for securing sensor-based networks and mitigating potential vulnerabilities. Ultimately, this work aims to identify the most accurate DL architecture for intrusion detection, leading to more robust security solutions for IoT sensor networks.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1:
INTRODUCTION

1.1 Background

The objective of the project is to develop and evaluate machine learning models for detecting and analysing vulnerabilities in IoT sensor networks.

The code implements three different deep learning architectures: CNN-BiLSTM, CNN-GRU, and CNN. These architectures are applied to a dataset containing network traffic data, specifically the KDD Cup 10 percent dataset, which is a part of the commonly used dataset for network intrusion detection tasks. The KDDCup99 is the original IoT net-work intrusion dataset that was created in 1999.

1.2 Internet of Things

The Internet of Things (IoT) has revolutionized various aspects of our lives, connecting everyday objects to the internet and enabling them to collect and transmit data. However, the growing prevalence of IoT sensors introduces significant security challenges. These sensors are often resource-constrained and may have inherent vulnerabilities that malicious actors can exploit to gain unauthorized access, disrupt operations, or steal sensitive data.

The Internet of Things (IoT) refers to everyday objects that are equipped with sensors and software, allowing them to collect and exchange data over the internet. This creates a network of connected devices that can automate tasks and improve efficiency. However, IoT faces some challenges such as **security, standardization** and **privacy**. We can address these challenges by implementing stronger encryption, regular software updates, and secure authentication methods that can improve IoT security, establishing industry-wide standards for communication protocols and data formats can ensure better compatibility between devices and clear user consent and data anonymization practices can help protect user privacy in the IoT world.

1.3 Deep Learning

Deep learning is a powerful subfield of machine learning inspired by the structure and function of the human brain. It uses artificial neural networks with multiple layers to learn complex patterns from data. Unlike traditional machine learning algorithms that require manual feature extraction, deep learning excels at automatically extracting features from raw data like images, text, or sound.

In technical terms, deep learning uses something called "neural networks," which are inspired by the human brain. These networks consist of layers of interconnected nodes that process information. The more layers, the "deeper" the network, allowing it to learn more complex features and perform more sophisticated tasks.

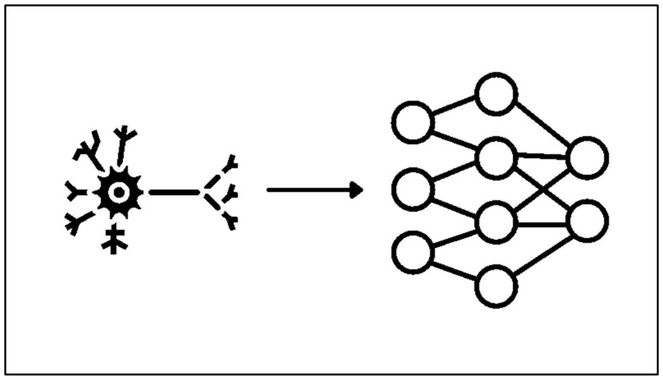


Fig 1.1 The Similarity Between Neurons and Neural Networks

1.3.1 Parts of a Neural Network

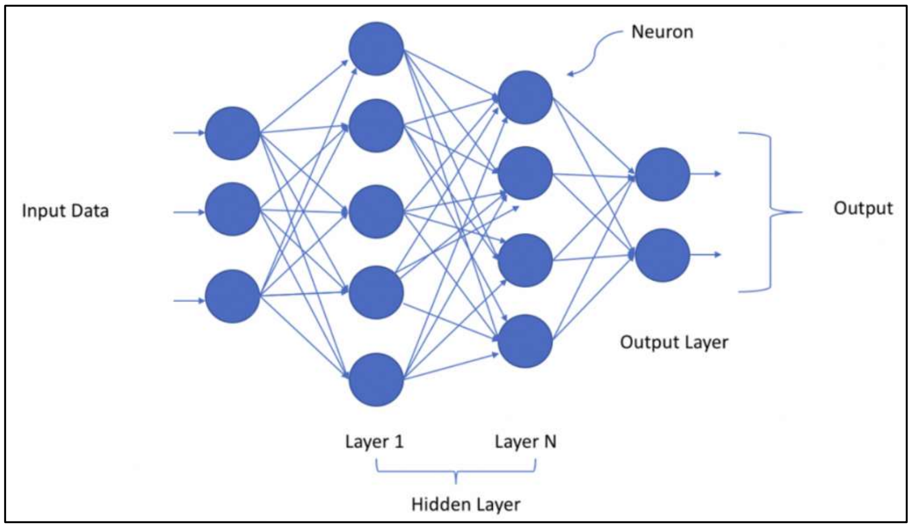


Fig 1.2 Parts of a neural network

A neural network typically consists of several interconnected layers, each performing specific operations on the input data. Here are the main parts of a neural network:

1. **Input Layer:**
    - The input layer is where the data is fed into the network.
    - Each neuron in the input layer represents a feature or attribute of the input data.
    - The number of neurons in the input layer corresponds to the dimensionality of the input data.
2. **Hidden Layers:**
    - Hidden layers are the intermediate layers between the input and output layers.
    - Each hidden layer contains a set of neurons, and the number of hidden layers and neurons per layer can vary based on the complexity of the problem.

- Hidden layers are responsible for learning and extracting features from the input data through nonlinear transformations.

3. **Weights and Biases:**
   - Each connection between neurons in adjacent layers is associated with a weight.
   - Weights determine the strength of the connection between neurons and are adjusted during the training process to minimize the error.
   - Biases are additional parameters added to each neuron that allow the network to learn more complex functions.
4. **Activation Function:**
   - The activation function introduces nonlinearity into the network, enabling it to learn complex patterns and relationships in the data.
   - Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax (for classification tasks).
5. **Output Layer:**
   - The output layer produces the final predictions or outputs of the network.
   - The number of neurons in the output layer depends on the nature of the task (e.g., regression, binary classification, or multi-class classification).
   - The activation function used in the output layer depends on the type of problem being solved (e.g., sigmoid for binary classification, softmax for multi-class classification).
6. **Loss Function:**
   - The loss function measures the difference between the predicted output and the actual target output.
   - It serves as a measure of how well the model is performing during training.
   - The goal of training is to minimize the loss function by adjusting the network's weights and biases.
7. **Optimizer:**
   - The optimizer is responsible for updating the weights and biases of the network during the training process.
   - It uses the gradients of the loss function with respect to the network parameters to determine how to adjust them in order to minimize the loss.

These are the key components of a neural network architecture.

1.3.2 Deep Learning Algorithms

Here's a brief description of some commonly used deep learning algorithms:

1. **Convolutional Neural Networks (CNNs):**
    - CNNs are primarily used for tasks involving image recognition and processing.
    - They consist of convolutional layers that apply filters to input data, followed by pooling layers to reduce dimensionality.
    - CNNs are highly effective in capturing spatial hierarchies of features in images.
2. **Recurrent Neural Networks (RNNs):**
    - RNNs are designed to handle sequential data where the order of elements matters, such as time series data or natural language.
    - They have connections that form directed cycles, allowing them to retain information over time.
    - However, traditional RNNs suffer from the vanishing gradient problem, limiting their ability to capture long-term dependencies.
3. **Long Short-Term Memory (LSTM):**
    - LSTMs are a type of RNN architecture designed to address the vanishing gradient problem.
    - They incorporate specialized memory cells and gating mechanisms to selectively remember or forget information over long sequences.
    - LSTMs have been widely used in tasks involving sequential data processing, such as language modelling, machine translation, and speech recognition.
4. **Gated Recurrent Units (GRUs):**
    - GRUs are another variant of the RNN architecture, similar to LSTMs but with a simplified gating mechanism.
    - They have fewer parameters compared to LSTMs, making them computationally less expensive and easier to train.
    - GRUs are suitable for tasks requiring memory over long sequences, such as language modelling and sentiment analysis.

These algorithms, among others, form the backbone of deep learning and have been instrumental in driving advancements in artificial intelligence across various industries and applications.

1.3.3 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a powerful type of deep learning algorithm specifically designed for image recognition and analysis. They excel at finding patterns in grid-like data, making them the go-to choice for various computer vision tasks.

Structure of a CNN:

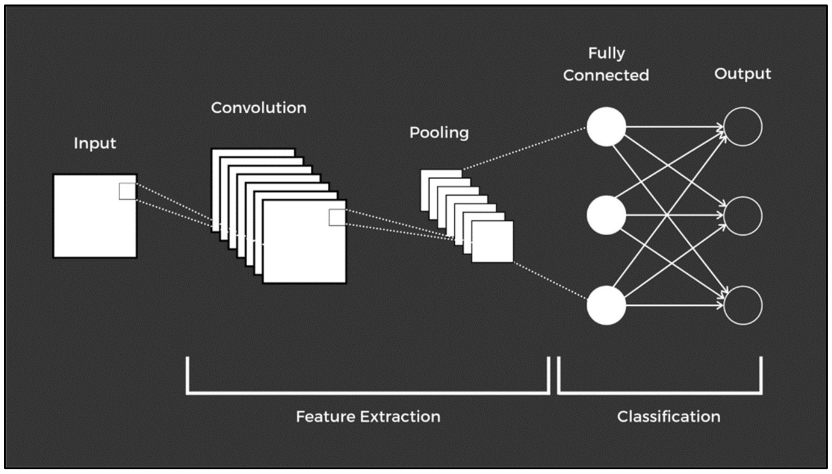


Fig 1.3 Structure of a CNN

A CNN architecture typically consists of three main types of layers:

1. **Convolutional Layers:** These layers are the heart of a CNN. They apply filters (also called kernels) to the input image, extracting features like edges, shapes, and textures. The filter slides across the image, performing element-wise multiplication with the underlying data and generating a feature map.
2. **Pooling Layers:** These layers down sample the feature maps produced by the convolutional layers. This reduces the dimensionality of the data, making it computationally efficient and helping to control overfitting. Common pooling techniques include max pooling, which selects the maximum value from a specific region of the feature map.
3. **Fully-Connected Layers:** These layers function similarly to traditional neural networks, taking the outputs from the pooling layers and performing classifications. They use activation functions to introduce non-linearity and help the network learn complex relationships between the features.

How CNNs Learn:

- **Training:** CNNs are trained on large datasets of labelled images. During training, the network adjusts the weights associated with its filters and neurons based on the difference between the predicted output and the actual label of the image.
- **Backpropagation:** This is a critical training technique used to adjust the weights in the network. Errors are propagated backward through the layers, allowing the network to learn from its mistakes and improve its feature extraction and classification capabilities.

Applications of CNNs:

- **Image Classification:** Recognizing objects, animals, or scenes within images.
- **Object Detection:** Locating and identifying specific objects within an image.

- **Image Segmentation:** Dividing an image into different regions corresponding to specific objects or features.
- **Facial Recognition:** Identifying individuals based on their facial features.

1.3.4 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a powerful type of deep learning architecture designed to handle sequential data. Unlike traditional neural networks that process individual data points, RNNs excel at analysing sequences where the order of elements matters. This makes them good for tasks like language translation, speech recognition, and time series forecasting.

RNN Architecture:

An RNN is built on repeating modules called cells. Each cell processes a single element from the sequence and updates its hidden state based on two factors: the current input (the element itself) and the previous hidden state (the network's memory of the sequence so far). This updated hidden state is then passed on to the next cell in the sequence, allowing information to flow and accumulate context throughout the network.

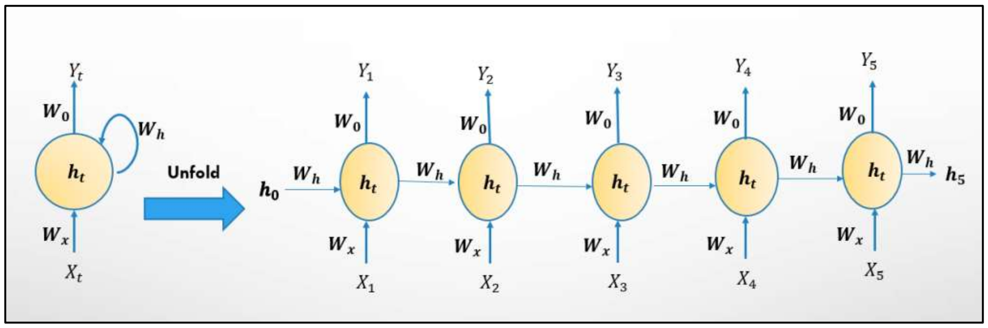


Fig 1.4 Structure of RNN

Imagine an RNN processing a sentence word by word. The first word becomes the current input, and the hidden state is initialized. As the network processes each subsequent word, the current input and the previous hidden state (containing information from prior words) are used to update the hidden state. This allows the network to consider the context of previous words when interpreting the current word.

Applications of RNNs:

- **Machine Translation:** Translating text from one language to another, considering the context of the entire sentence.
- **Speech Recognition:** Converting spoken language into text, accounting for the order of words within a speech pattern.
- **Text Generation:** Generating text that follows a specific style or language pattern.
- **Time Series Forecasting:** Predicting future values in a time series based on historical data.

1.3.5 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a specialized type of Recurrent Neural Network (RNN) designed to address a significant limitation of traditional RNNs: the vanishing gradient problem. This problem hinders RNNs' ability to learn long-term dependencies within sequential data. LSTMs overcome this limitation, making them a powerful tool for tasks involving sequences like text, speech, and time series data.

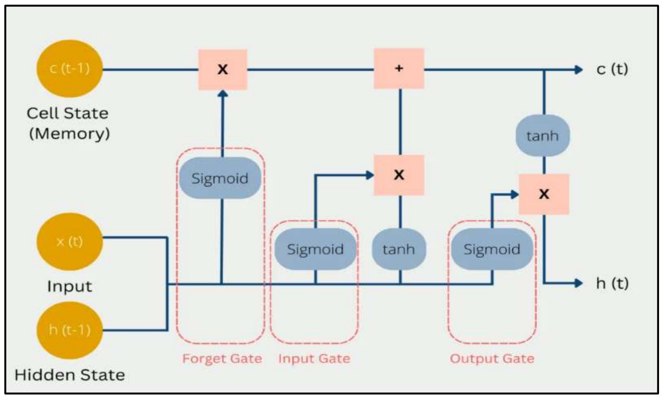


Fig 1.5 Long Short-Term Memory

The LSTM Architecture:

LSTMs address this challenge by introducing a gating mechanism that controls the flow of information within the network. This mechanism consists of several important components:

- **Cell State:** This acts as the network's memory, carrying information across different time steps.
- **Forget Gate:** This gate decides what information to forget from the cell state of the previous time step. It considers the new input and the previous cell state, ultimately outputting values between 0 and 1. A value closer to 1 signifies retaining more information, while 0 indicates forgetting.
- **Input Gate:** This gate determines what new information to store in the cell state. It analyses the current input and the previous cell state, outputting values between 0 and 1.
- **Output Gate:** This gate controls what information from the cell state to output as part of the hidden state (the network's output at a specific time step).

How LSTMs Learn:

- **Information Flow:** During training, the LSTM processes the sequence one element at a time. At each step, the forget gate, input gate, and output gate determine how information flows through the cell state and hidden state.
- **Backpropagation:** Similar to other neural networks, LSTMs utilize backpropagation to adjust their internal weights and learn from errors.

Applications of LSTMs:

- **Machine Translation:** Translating text from one language to another while considering the context of the entire sentence.
- **Speech Recognition:** Converting spoken language into text, accounting for long-term dependencies within speech patterns.
- **Time Series Forecasting:** Predicting future values in a time series based on historical data, such as stock prices or weather patterns.
- **Anomaly Detection:** Identifying unusual patterns within sequences, useful for fraud detection or system monitoring.

1.3.6 Gated Recurrent Units (GRU)

Gated Recurrent Units (GRUs) are a type of RNN that addresses the vanishing gradient problem by introducing a gating mechanism similar to LSTMs. This mechanism allows GRUs to control the flow of information within the network and focus on the most relevant parts of the sequence.

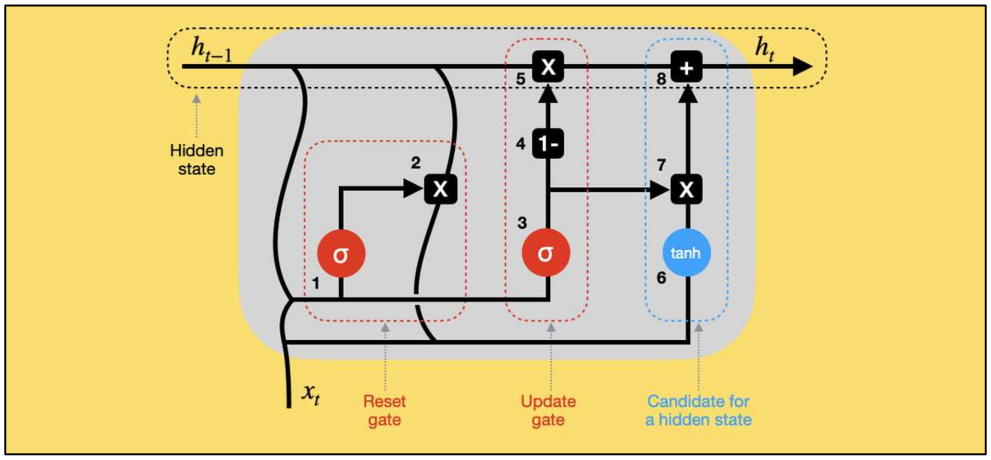


Fig 1.6 Gated Recurrent Unit Architecture

GRU Architecture:

A GRU cell resembles an RNN cell with the addition of these gating mechanisms. The update gate and reset gate control the flow of information through the hidden state, mitigating the vanishing gradient problem and allowing GRUs to learn long-term dependencies more effectively than traditional RNNs.

Similar to LSTMs, GRUs utilize gates to regulate information flow. However, GRUs have a simpler architecture compared to LSTMs, using a single update gate and a reset gate:

- **Update Gate:** This gate decides how much of the previous hidden state information to keep and how much new information from the current input to incorporate.
- **Reset Gate:** This gate determines which parts of the previous hidden state are still relevant and should be carried forward.

Applications of GRUs:

- **Similar to RNNs:** GRUs can be applied to various tasks involving sequential data, such as machine translation, speech recognition, and text generation.
- **Potentially Faster Training:** Due to their simpler architecture, GRUs can sometimes train faster than LSTMs on specific tasks.

1.4 Deep Learning Algorithms Used in Project

1.4.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a class of deep neural networks particularly well-suited for tasks involving image recognition and processing. They have played a pivotal role in advancing computer vision applications, including object detection, image classification, and semantic segmentation.

Key Components:

- **Convolutional Layers:** CNNs utilize convolutional layers to extract features from input images. These layers apply learnable filters to small regions of the input image, capturing patterns such as edges, textures, and shapes. Through successive convolutions, deeper layers can learn more abstract and complex features.

- **Pooling Layers:** Pooling layers are used to down sample feature maps produced by convolutional layers, reducing spatial dimensions and computational complexity while preserving important features. Common pooling operations include max pooling and average pooling.

- **Activation Functions:** Non-linear activation functions, such as ReLU (Rectified Linear Unit), are applied to the output of convolutional and pooling layers. These functions introduce non-linearity into the network, enabling it to learn complex mappings between input and output.

- **Fully Connected Layers:** Following the convolutional and pooling layers, fully connected layers perform high-level reasoning and decision-making. These layers integrate features learned from previous layers and map them to the output classes or labels.

Applications:

CNNs have demonstrated remarkable performance in various computer vision tasks, including:

- Image Classification: Assigning labels or categories to input images.
- Object Detection: Identifying and localizing objects within images or videos.
- Semantic Segmentation: Pixel-wise classification of objects and regions in images.
- Facial Recognition: Recognizing and verifying faces in images or videos.

- Medical Image Analysis: Analysing medical images for diagnosis and treatment planning.

1.4.2 CNN-BiLSTM (Convolutional Neural Network - Bidirectional Long Short-Term Memory)

CNN-BiLSTM is a hybrid deep learning architecture that combines the strengths of Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (BiLSTM) networks. This architecture is specifically designed for processing sequential data with spatial and temporal dependencies, such as time series data and sequences of features extracted from images.

Key Components:

1. **Convolutional Layers:** The CNN component of the architecture extracts spatial features from input data. Convolutional layers apply a series of learnable filters to input sequences, capturing local patterns and spatial hierarchies. These layers are effective at feature extraction and dimensionality reduction.
2. **Bidirectional LSTM Layers:** The BiLSTM component processes the output of the convolutional layers in both forward and backward directions. Bidirectional LSTMs consist of two LSTM networks: one processes the input sequence from the beginning to the end, while the other processes it in reverse. This enables the model to capture both past and future context for each time step, allowing for a better understanding of temporal dependencies.
3. **Pooling Layers (Optional):** Pooling layers may be incorporated after the convolutional layers to down sample feature maps and reduce computational complexity while preserving important features.
4. **Fully Connected Layers:** Following the CNN-BiLSTM layers, fully connected layers perform high-level reasoning and decision-making. These layers integrate features learned from the preceding layers and map them to the output classes or labels.

Training and Optimization:

Training CNN-BiLSTM involves feeding the network with labelled sequential data, computing the loss between predicted and true labels, and optimizing network parameters (weights and biases) using backpropagation and optimization algorithms such as stochastic gradient descent (SGD) or Adam.

Applications:

CNN-BiLSTM architectures have been successfully applied to various sequential data analysis tasks, including:

- Time Series Forecasting: Predicting future values in sequential data, such as financial time series or sensor readings.
- Video Analysis: Analysing sequences of frames in videos for action recognition, scene understanding, and video captioning.

- Natural Language Processing: Processing sequences of words or characters for tasks such as sentiment analysis, machine translation, and named entity recognition.

1.4.3 GRU (Gated Recurrent Unit)

Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture designed to address the limitations of traditional RNNs, such as vanishing gradients and difficulty in capturing long-term dependencies. GRUs are particularly effective for sequential data processing tasks where information needs to be retained over multiple time steps.

Key Components:

1. **Update Gate:** The update gate in GRU determines how much of the previous memory should be retained and how much of the new information should be incorporated. It computes the relevance of the previous memory and the new input at each time step.
2. **Reset Gate:** The reset gate controls the degree to which the previous memory should be forgotten or reset. It decides which parts of the previous memory are outdated and should be ignored.
3. **Candidate Activation:** The candidate activation computes the new candidate memory based on the current input and the reset gate. It combines the new input with the relevant parts of the previous memory.
4. **Hidden State:** The hidden state of the GRU represents the current memory state or context. It is updated at each time step based on the update gate and the candidate activation.

Training and Optimization:

During training, GRUs learn to capture temporal dependencies and patterns in sequential data. The training process involves feeding the network with labeled sequential data, computing the loss between predicted and true labels, and optimizing network parameters (weights and biases) using backpropagation and optimization algorithms such as stochastic gradient descent (SGD) or Adam.

Applications:

GRUs have been successfully applied to various sequential data processing tasks, including:

- Natural Language Processing: Processing sequences of words or characters for tasks such as language modelling, machine translation, and sentiment analysis.
- Time Series Forecasting: Predicting future values in sequential data, such as stock prices, weather data, or sensor readings.
- Speech Recognition: Converting speech signals into text for tasks such as voice transcription and virtual assistants.
- Video Analysis: Analysing sequences of frames in videos for action recognition, scene understanding, and video captioning.

1.5 Project Objective

The Internet of Things (IoT) has revolutionized various aspects of our lives, connecting everyday objects to the Internet and enabling them to collect and transmit data. However, the growing prevalence of IoT sensors introduces significant security challenges. These sensors are often resource-constrained and may have inherent vulnerabilities that malicious actors can exploit to gain unauthorized access, disrupt operations, or steal sensitive data.

This project investigates the effectiveness of Deep Learning (DL) algorithms for vulnerability analysis in IoT sensors. DL has emerged as a powerful tool for network intrusion detection, and this project explores its potential for securing sensor-based networks. We propose a framework centred on Convolutional Neural Networks (CNNs) for intrusion detection within IoT networks.

The project compares the performance of three distinct CNN architectures:

Baseline CNN: This model employs a standard CNN architecture for network traffic classification.

CNN-BiLSTM: This variation incorporates Bidirectional Long Short-Term Memory (BiLSTM) layers after the convolutional layers. BiLSTM layers excel at capturing sequential dependencies within data, potentially improving intrusion detection by considering the temporal nature of network traffic.

CNN-GRU: This model utilizes Gated Recurrent Unit (GRU) layers instead of BiLSTMs. GRUs offer similar functionality but with a simpler architecture.

The project utilizes the **KDD Cup 10 percent dataset**, a benchmark dataset for intrusion detection, to train and evaluate these models. We perform essential data preprocessing steps like label encoding, feature standardization, and reshaping for the CNN layer. Following preprocessing, each model undergoes training and evaluation.

Our primary objective is to identify the most accurate model for vulnerability analysis in IoT sensor networks. We compare the models' performance based on their ability to classify various network traffic patterns into distinct attack categories. Classification reports and visualizations of training and validation loss/accuracy curves will be employed to assess model convergence, and generalizability, and ultimately, pinpoint the most accurate model for this task.

This project contributes valuable insights into leveraging DL for enhanced IoT security. By comparing the accuracy of CNN, CNN-BiLSTM, and CNN-GRU architectures in intrusion detection, we aim to identify the most effective approach for vulnerability analysis and intrusion detection in sensor-based networks. Additionally, the project demonstrates the process of loading a trained model and making predictions on new sensor data, showcasing the practical application of the most accurate DL model in real-world IoT security scenarios.

CHAPTER 2:

LITERATURE SURVEY

Predictive maintenance (PdM) has become a crucial aspect of modern industries, utilizing machine learning (ML) and Internet-of-Things (IoT) sensors to predict equipment failures before they occur. These PdM systems offer significant benefits such as reduced downtime, lower maintenance costs, and increased production. However, the reliance on IoT sensors and ML algorithms introduces vulnerabilities to cyberattacks that can manipulate sensor data and compromise the effectiveness of PdM .

Existing research focuses on improving the accuracy of PdM systems using deep learning (DL) techniques. However, there is a critical gap regarding the impact of cyberattacks, particularly False Data Injection Attacks (FDIA), on these systems. FDIA stealthily alters sensor measurements, bypassing basic detection mechanisms and feeding manipulated data into the ML models. This can lead to delayed maintenance or even catastrophic failures in safety-critical applications, highlighting the need for further investigation.

While extensive research explores attack detection and mitigation in cyber-physical systems (CPS), the impact of FDIA on PdM systems remains largely unexplored. This is concerning, especially for applications like aircraft engine maintenance, where delayed actions due to FDIA can cause mid-air engine failures. The widespread use of PdM systems in the aerospace industry by companies like Pratt and Whitney, Rolls-Royce, and General Electric further emphasizes the importance of addressing this issue.

Modern aircraft engines, equipped with thousands of sensors, leverage advanced DL algorithms to predict maintenance needs and optimize fuel usage. However, the vulnerability of these sensor-based systems to attacks remains a challenge. Existing sensor attack detection solutions designed for broader IoT and CPS domains might not be suitable for PdM due to scalability limitations and resource constraints on individual sensors.

This research aims to bridge this gap by investigating the impact of FDIA on PdM systems. We will model realistic scenarios with a limited number of compromised sensors and analyze the effects on different deep learning-based PdM models. This will contribute valuable insights into the vulnerabilities of PdM systems and pave the way for developing robust solutions against cyberattacks**.[1]**

The vast number of interconnected devices within the Internet of Things (IoT) landscape presents significant security challenges. The proliferation of IoT devices across smart homes, industrial systems, and personal devices introduces new attack vectors for malicious actors to exploit. These devices often collect and transmit sensitive data, making them prime targets for eavesdropping, data breaches, and denial-of-service attacks. Traditional security methods are struggling to keep pace with the evolving threats posed by the ever-growing IoT ecosystem.

Deep learning has emerged as a promising approach for intrusion detection in IoT systems. Deep learning algorithms can effectively analyze large amounts of data to identify patterns and anomalies that may indicate malicious activity. This research investigates the performance of three deep learning models for intrusion detection in IoT: convolutional neural networks

(CNNs), long short-term memory (LSTM), and gated recurrent units (GRUs). CNNs are well-suited for extracting spatial features from data, making them effective for identifying anomalies in network traffic patterns. LSTMs and GRUs are a type of recurrent neural network (RNN) that can learn long-term dependencies within data sequences. This makes them suitable for analyzing time-series data collected from IoT devices, where identifying patterns and anomalies across sequences of sensor readings can be crucial for intrusion detection.

By comparing the performance of these three deep learning models on a standard IoT intrusion detection dataset, this research aims to identify the most accurate approach for intrusion detection in IoT systems. The findings of this research will contribute to the development of more secure IoT environments by improving the accuracy of intrusion detection and mitigating the risks associated with cyberattacks. **[2]**

As cyber attacks and cybercriminals target cyber-physical systems (CPSs) with increasing frequency, the need for robust detection mechanisms becomes paramount. While traditional methods struggle to keep pace, a new era of opportunity dawns with the emergence of machine learning (ML), particularly deep learning (DL). DL's layered architecture and ability to extract valuable information from training data make it superior to traditional machine learning methods in this context. The survey analyzes recent DL solutions through a six-step methodology, highlighting the importance of understanding the CPS scenario, identifying relevant attacks, formulating the detection problem, customizing DL models, acquiring training data, and evaluating performance. Existing research shows promise for DL-based attack detection, partly due to the availability of high-quality public datasets. The survey concludes by outlining key challenges, opportunities, and future research directions in this domain. **[3]**

In the domain of predictive maintenance for rotating machines, a promising approach revolves around analyzing the shape of the rotor shaft's orbit. This survey delves into a study that proposes a novel algorithm for this purpose, leveraging Convolutional Neural Networks (CNNs). CNNs are powerful tools for image pattern recognition, and in this application, the CNN is trained on a comprehensive database of various orbit shapes. This training empowers the CNN to not only detect deviations from normal operating patterns but also to classify the specific fault type causing the anomaly. This capability offers a multitude of benefits for predictive maintenance programs. Early fault detection becomes possible, allowing for timely intervention to prevent catastrophic failures and ensure operational safety. Additionally, optimized maintenance schedules can be implemented based on the insights gleaned from orbit analysis, leading to reduced maintenance costs and improved resource allocation. **[4]**

Smart home devices, while bringing convenience to our lives, introduce new security risks due to the lack of standardized security measures. Existing vulnerability studies often focus on well-known vendors, who tend to have stronger security due to public scrutiny. This research highlights the potential vulnerability of lesser-known vendors with lax security practices. Through a review of existing research and a comparative analysis of security postures between different vendors, the study aims to confirm that lesser-known vendors are under-represented in vulnerability research and have weaker security. This focus on under-researched areas contributes to a more comprehensive understanding of security vulnerabilities in smart home IoT devices. **[5]**

CHAPTER 3:
EXISTING SYSTEM

In the era of Industry 4.0, predictive maintenance (PdM) solutions have emerged as crucial tools for fault prediction in components and systems. These solutions leverage advanced machine learning algorithms, particularly deep learning, and Internet-of-Things (IoT) sensors to enhance predictive capabilities. However, the susceptibility of IoT sensors and deep learning algorithms to cyber-attacks, specifically False Data Injection Attacks (FDIA), poses a significant threat to the reliability and effectiveness of PdM systems.

3.1 Objective

This study aims to investigate the repercussions of False Data Injection Attacks on deep learning-enabled PdM systems, focusing on the vulnerabilities introduced by compromised IoT sensor data. By analysing the impact of FDIA on predictive maintenance processes, the research seeks to underscore the importance of developing resilient algorithms and detection mechanisms to mitigate these threats effectively.

3.2 Methodology

1. **Dataset Selection:**
   - The study utilizes NASA's C-MAPSS dataset for modelling False Data Injection Attacks (FDIA) on a turbofan engine Predictive Maintenance (PdM) system.

2. **Algorithm Selection and Training:**
   - Three state-of-the-art deep learning algorithms are employed for RUL prediction: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Convolutional Neural Network (CNN).
   - The algorithms are trained on the C-MAPSS dataset to predict the Remaining Useful Life (RUL) of the turbofan engine.

3. **Performance Evaluation:**
   - The performance of LSTM, GRU, and CNN models is evaluated based on their accuracy in predicting RUL using the dataset.

4. **False Data Injection Attack Modelling:**
   - Two types of False Data Injection Attacks (FDIA) are modeled and applied to the turbofan engine sensor data: continuous and interim attacks.
   - The attacks involve compromising sensor measurements by a small margin to bypass fault detection mechanisms and impact the predictive maintenance process.

5. **Impact Assessment:**
   - The effects of continuous and interim FDIA on the predictive maintenance systems based on LSTM, GRU, and CNN models are analyzed.
   - The study assesses how the attacks influence the accuracy and resilience of the deep learning-enabled PdM systems.

6. **Result Analysis:**
    - The obtained results are analyzed to determine the extent of the impact of FDIA on RUL prediction accuracy and the performance of the deep learning algorithms.
    - Insights are drawn regarding the vulnerabilities introduced by compromised IoT sensor data and the implications for predictive maintenance processes.

7. **Comparison and Validation:**
    - A comparison is made between the performance of LSTM, GRU, and CNN models in the presence of FDIA to evaluate their robustness and effectiveness.
    - The validation of the results is conducted to ensure the reliability and reproducibility of the findings.

8. **Detection Mechanism Exploration:**
    - The study may explore potential detection mechanisms or strategies to mitigate the effects of False Data Injection Attacks on deep learning-enabled predictive maintenance systems.
    - Recommendations for enhancing the security and resilience of PdM systems against cyber-attacks may be proposed based on the findings.

3.3 Result

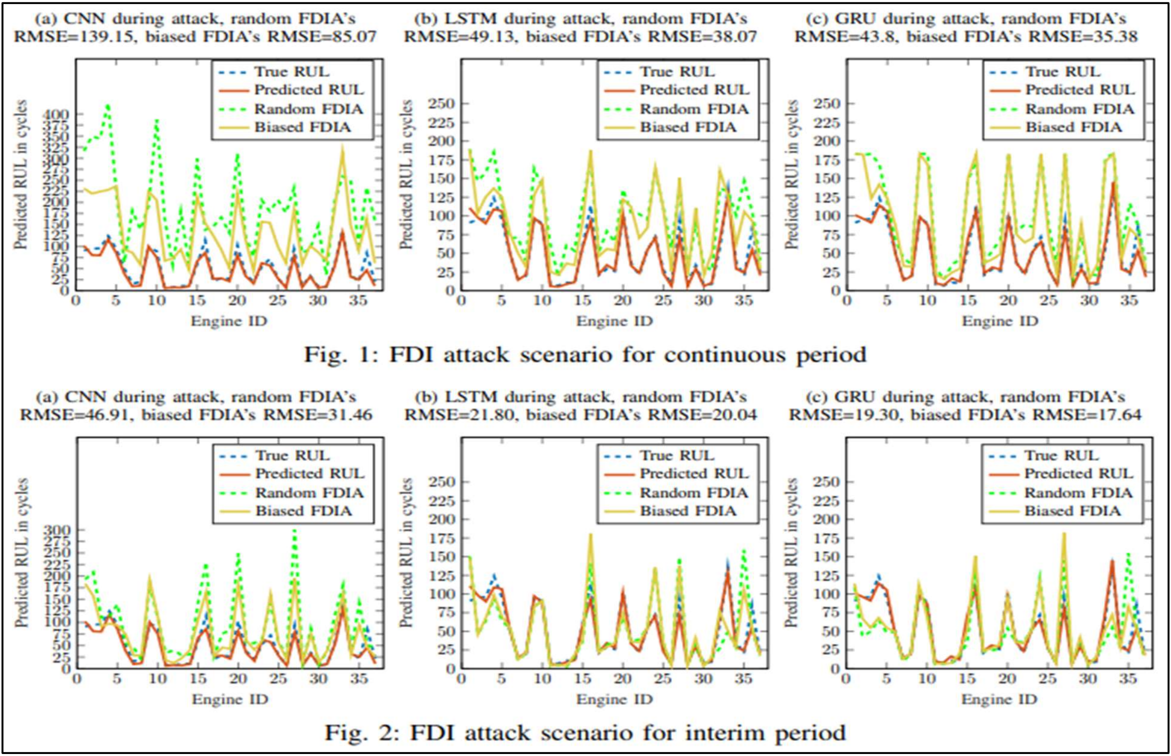


Fig 3.1 Result Analysis of PdM System

- The GRU-based PdM model surpasses existing literature in terms of RUL prediction accuracy using the C-MAPSS dataset.

- Continuous and interim FDIA significantly impact RUL prediction accuracy, highlighting the vulnerabilities of deep learning-enabled PdM systems to sensor attacks.
- The stealthy nature of FDIA complicates detection, emphasizing the necessity for robust detection techniques.

3.4 Future Scope

More datasets can be explored for intrusion detection on IoT devices in future. This work can also be extended to study the effect of other Deep Learning variants of algorithms such as genetic algorithm (GA) and bidirectional short-term memory (BiLSTM) for better performance.

CHAPTER 4:
PROPOSED SYSTEM

The goal is to detect network intrusions or attacks based on network traffic data. Each record in the dataset represents a network connection, and the task is to classify each connection as either normal or malicious. The dataset is loaded into a Pandas DataFrame and subjected to preprocessing. Categorical columns undergo encoding using LabelEncoder, while numerical columns are standardized with StandardScaler. Labels are converted to categorical format for model training.

Three different neural network architectures are implemented:

1. **CNN-BiLSTM:** A combination of Convolutional Neural Network (CNN) layers followed by Bidirectional Long Short-Term Memory (BiLSTM) layers.
2. **CNN-GRU:** Similar to CNN-BiLSTM, but with Gated Recurrent Unit (GRU) layers instead of BiLSTM.
3. **CNN:** A simple Convolutional Neural Network architecture.

The dataset is split into training and testing sets. Each model is compiled with appropriate loss function, optimizer, and evaluation metrics. Models are trained on the training data with a specified number of epochs and batch size. After training, the models are evaluated on the testing set to assess performance metrics such as accuracy, precision, recall, and F1-score. Trained models are saved to disk in HDF5 format (.h5) using model.save method for future use without retraining. Models can be integrated into a production environment for real-time or batch processing of network traffic data. Continuous monitoring of the deployed models' performance is essential to ensure accuracy over time. Periodic retraining may be necessary to adapt to evolving attack patterns and changes in network behaviour. The trained models can be integrated into a Network Intrusion Detection System (NIDS) framework for automated detection of network attacks. Real-time network traffic can be passed through the deployed models for classification, enabling swift response to potential threats. Depending on the scale of the network and available computational resources, models can be optimized for performance and scalability using techniques such as distributed training or model quantization.

4.1 Merits of Proposed System

- **Machine Learning for Evolving Threats:** The system utilizes machine learning to identify complex and evolving attack signatures that may evade traditional rule-based methods.
- **High Accuracy with CNN-BiLSTM:** By training a CNN-BiLSTM model on historical labeled data, the system achieves high accuracy in detecting various attack types.

  CNN-BiLSTM excels over a baseline CNN by:

  - Capturing long-term dependencies in network traffic data, crucial for identifying sophisticated attacks.

    - Learning relationships between network events spread across time, beneficial for detecting multi-stage attacks.
    - Creating a richer feature representation through combined CNN and BiLSTM strengths, leading to better classification of normal and attack traffic.

- **Real-time Analysis and Response:** The system performs real-time analysis, enabling prompt responses to potential attacks, minimizing the attacker's window of opportunity.
- **Flexibility and Customization:** The system allows exploration of different models (e.g., CNN-GRU) and customization of features and attack classifications based on specific network environments and security needs.
- **Automated Alerting and Actions:** The system provides automated alerting for security personnel and can initiate predefined response actions (e.g., blocking traffic, isolating systems) to mitigate attacks.

CHAPTER 5:

SYSTEM DESIGN ARCHITECTURE

5.1 Architecture Diagram

Data Analysis: This is the initial step where the raw data is examined to understand its properties and suitability for the task. This might involve tasks like identifying the data, missing entries, and analysing data distribution.

Data Preprocessing: In this stage, the raw data is transformed into a format that the machine learning model can understand and process effectively. This may involve techniques like tokenization, stemming, lemmatization, and vectorization.

Model Splitting: Here, the pre-processed data is divided into two sets: a training set and a validation set. The training set is used to train the model, while the validation set is used to evaluate the model's performance and prevent overfitting.

Model Training: This is where the machine learning model learns from the training data. The model is iteratively adjusted to improve its performance on the training set.

Model Validation: The performance of the trained model is assessed on the validation set. This helps identify if the model is overfitting on the training data and generalizes well to unseen data.

Model Saving: If the model's performance on the validation set meets the criteria, the model is then saved for future use.

Prediction: Once a model is trained and saved, it can be used to make predictions on new, unseen data.

Different model architectures, including CNN (Convolutional Neural Network), GRU (Gated Recurrent Unit), and CNN-BILSTM (Bidirectional Long Short-Term Memory) are used as the models for conducting the analysis.

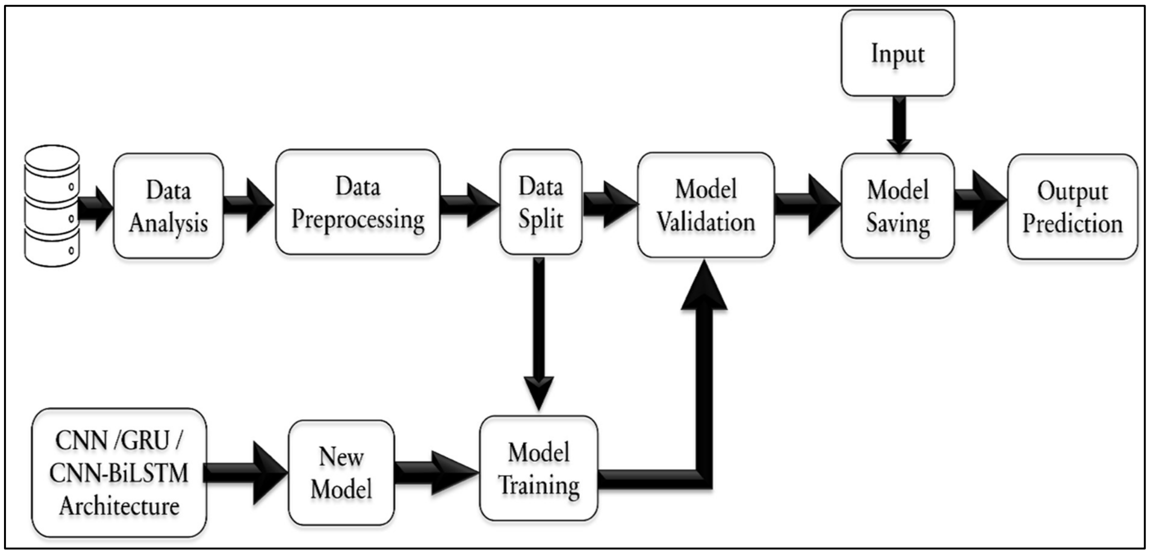


Fig 5.1 System Architecture

5.2 Use Case Diagram

A use case diagram illustrates the interaction between a user/admin and a system. This diagram shows the different functionalities available to a user of a system.

Register: This functionality allows a new user to create an account within the system.

Login: This functionality allows a registered user to gain access to the system using their credentials.

View User: This functionality allows a user to access and view their account information within the system.

Upload: This functionality allows a user to upload data or files to the system.

Log Out: This functionality allows a user to end their session and exit the system.

The user can access the functionalities: Register, login, Upload and Log Out

The admin can access the functionalities: Login, View User and Log Out

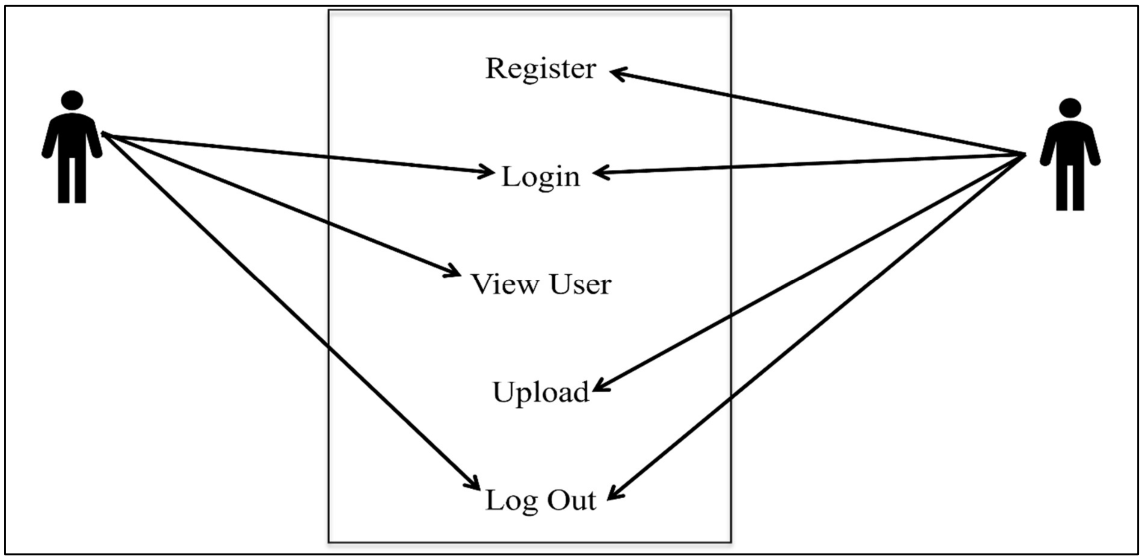


Fig 5.2 Use Case Diagram

5.3 Data Flow Diagrams

5.3.1 Level 0

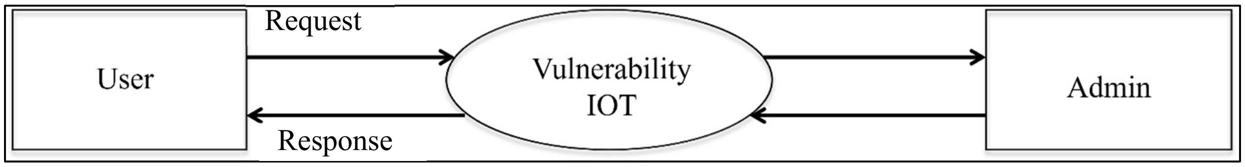


Fig 5.3 Level 0 DFD

A Level 0 DFD, which is a high-level view of a system that depicts the overall flow of data. This DFD shows a system with three main entities: a User, an Administrator, and a Vulnerability in an Internet of Things (IoT) device.

Request: The User initiates the process by requesting information from the Administrator. The nature of this request is not explicitly shown in the diagram but could be something like reporting a suspected vulnerability in an IoT device they manage.

Response: The Administrator then processes the request and sends a response back to the User. Again, the nature of this response is not explicitly shown but could be information or guidance on how to address the reported vulnerability.

5.3.2 Level 1 (Admin Side)

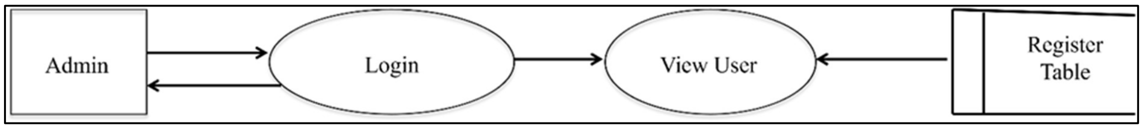


Fig 5.4 Level 1 (Admin Side) DFD

A Level 1 DFD, which is a more detailed view of a system than a Level 0 DFD.

User: The User entity is responsible for two main functions:

Report Vulnerability: The user initiates the process by reporting a suspected vulnerability in an IoT device they manage. Details about the suspected vulnerability are captured in a report.

View Response: The User can view the response sent by the Administrator regarding the reported vulnerability.

Administrator: The Administrator entity is responsible for processing the information received from the User and sending a response. It has two main functions:

Process Report: The Administrator receives and processes the report submitted by the User regarding the vulnerability in an IoT device. This likely involves tasks such as analyzing the report to assess the severity of the vulnerability.

Send Response: The Administrator sends a response back to the User regarding the reported vulnerability. The response may include information on how to address the vulnerability or next steps the user should take.

5.3.3 Level 1 (User Side)

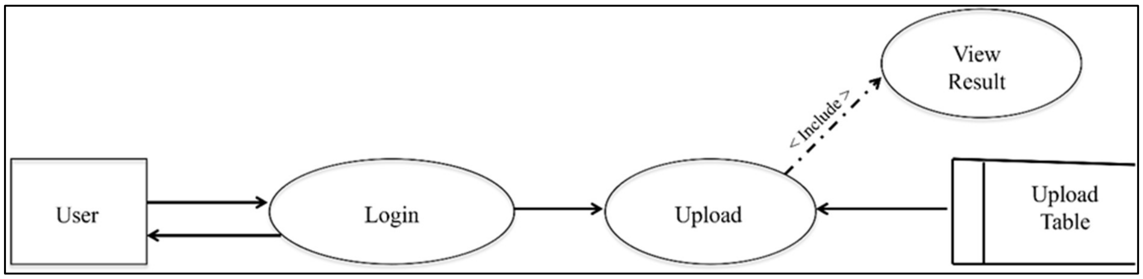


Fig 5.5 Level 1 (User Side) DFD

A Level 1 Data Flow Diagram (DFD) for a user interacting with a database system. It depicts the flow of data as a user interacts with the system to view and potentially update information.

User: The User entity is responsible for initiating interactions with the system. The user has two main functions:

Login: The user can log in to the system using their credentials.

View/Update: Once logged in, the user can view and potentially update information stored in the database.

System: The system entity is responsible for processing the user's requests and interacting with the database. It has two main functions:

Process Login: The system validates the user's credentials upon login.

Process View/Update: The system retrieves data from the database based on the user's request and presents it to the user. It also allows the user to update the data and stores the changes back into the database.

Database: The Database entity stores the system's data. It interacts with the system to provide and receive data based on user requests.

<u>Data Flows:</u>

The arrows in the DFD represent the flow of data between the user, system, and database entities. Here's a breakdown of the data flows depicted in the image:

Login Credentials: The user submits their login credentials to the system.

Login Validation: The system sends the login credentials to the database to verify the user's identity.

Login Response: The system sends a response to the user indicating whether the login was successful.

Data Request: The user can submit a request to view or update data stored in the database.

Data Retrieval: The system retrieves the requested data from the database.

Data View: The system presents the retrieved data to the user.

Data Update: The user can submit updates to the data.

Data Update Storage: The system stores the updated data back into the database.

CHAPTER 6:

SYSTEM REQUIREMENTS

6.1 Software Requirements

- Operating System: Windows 10 or above
- IDE: Notepad ++
- Front End: HTML, CSS, js,
- Back End: python, MYSQL, Django
- Tool kit: XAMPP

6.2 Hardware Requirements

PROCESSOR: Intel Core i3 – 3220 (3.3 Ghz) or above

RAM: 4 GB or above

STORAGE: 512 GB or above

OTHER: Keyboard and Mouse

CHAPTER 7:
MODULE DESCRIPTION

This analysis consists of 5 modules:

1. Data Manipulation and Analysis
2. Data Preprocessing
3. Model Building and Training
4. Model Evaluation
5. Virtualization

7.1 Module 1: Data Manipulation and Analysis:
This module imports the pandas libraries using import pandas as pd. Pandas is used for working with tabular data. It provides Data Frames, which are essentially spreadsheets in Python, for organizing and manipulating data.
pandas (pd): This module offers powerful data structures (DataFrames) for handling tabular data. It's used for:

Reading the CSV file into a DataFrame (pd.read_csv)

Displaying the DataFrame (print(df.head()))

Selecting specific columns (df.select_dtypes)

Saving a DataFrame to a CSV file (test_data.to_csv)

numpy (np): This module provides numerical computing functionalities:

Reshaping data for the Conv1D layer (X.values.reshape(...))

7.2 Module 2: Data Preprocessing:
Data is set into arrays for mathematical computation of data, after which the data is split into two parts for training and testing sets which is crucial for evaluating the model. Here the data is standardised into numerical features for better training performance, the data is transformed from text lables to numerical values which is better understood by the models.
sklearn.model_selection: This sub library from scikit-learn offers tools for splitting data into training and testing sets:

train_test_split: Splits the data for training and testing (train_test_split)

sklearn.preprocessing: This sub library from scikit-learn provides methods for data preprocessing:

StandardScaler: Standardizes numerical features for better training performance (StandardScaler)

LabelEncoder: Encodes categorical features into numerical values (LabelEncoder)

7.3 Module 3: Model Building and Training:

In this module various files and functions are included for building and training the models that are used for this analysis in the cause of this research the models included are GRU, CNN and CNN-BiLSTM.
tensorflow.keras: This module from TensorFlow offers deep learning functionalities:

Sequential: Used to build a sequential neural network model (Sequential)

Conv1D, MaxPooling1D: Used for convolutional layers and pooling in 1D data (Conv1D, MaxPooling1D)

Bidirectional, LSTM, GRU, Dense, Dropout: Used for various neural network layers, including recurrent layers (Bidirectional, LSTM, GRU), fully connected layers (Dense), and dropout regularization (Dropout)

Flatten: Flattens the data before feeding it to fully connected layers (Flatten)

to_categorical: Converts categorical labels to one-hot encoding (to_categorical)

model.compile: Compiles the model by specifying the optimizer, loss function, and metrics (model.compile)

model.fit: Trains the model on the provided data (model.fit)

model.save: Saves the trained model (model.save)

load_model: Loads a pre-trained model (load_model)

7.4 Module 4: Model Evaluation:

Here we evaluate the performance of the models after they are trained. Functions that provide tools are used for providing a detailed breakdown of the model's performance on the testing set, including metrics like precision, recall, and F1-score.
sklearn.metrics: This sublibrary from scikit-learn provides functions for evaluating machine learning models:

classification_report: Prints a classification report summarizing model performance on the testing set (classification_report)

7.5 Module 5: Visualization:

It's used here to plot the training and validation loss/accuracy curves across training epochs, which helps understand how the model learns and performs during training. The code also saves these plots as images for further analysis.
matplotlib.pyplot (plt): This module is used for creating plots and visualizations:

Plotting loss and accuracy curves (plt.plot)

Saving plots as images (plt.savefig)

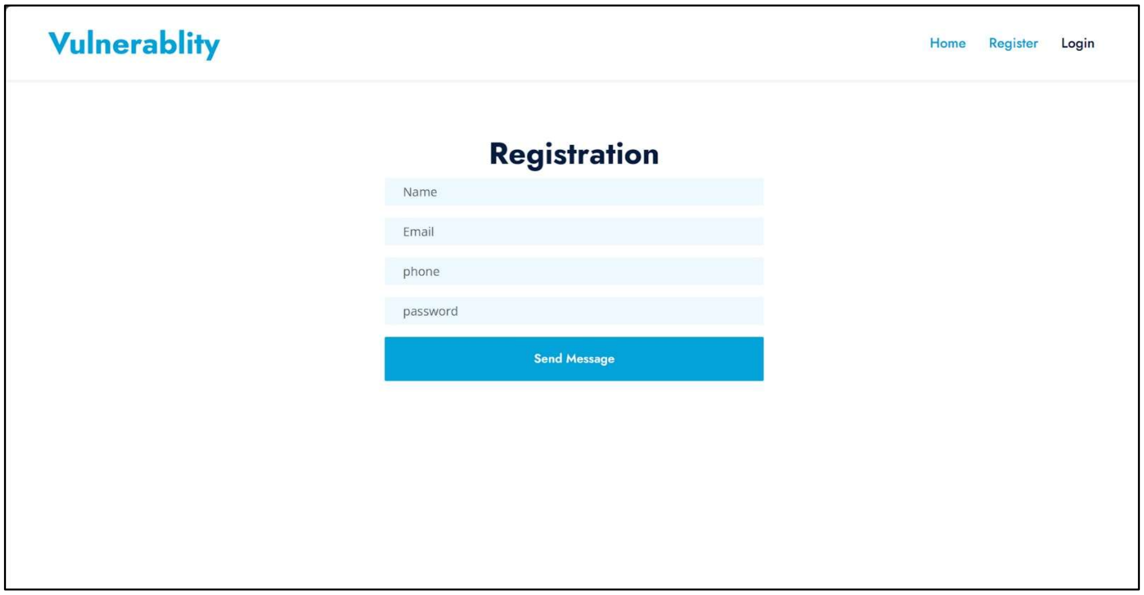
CHAPTER: 8

IMPLEMENTATION

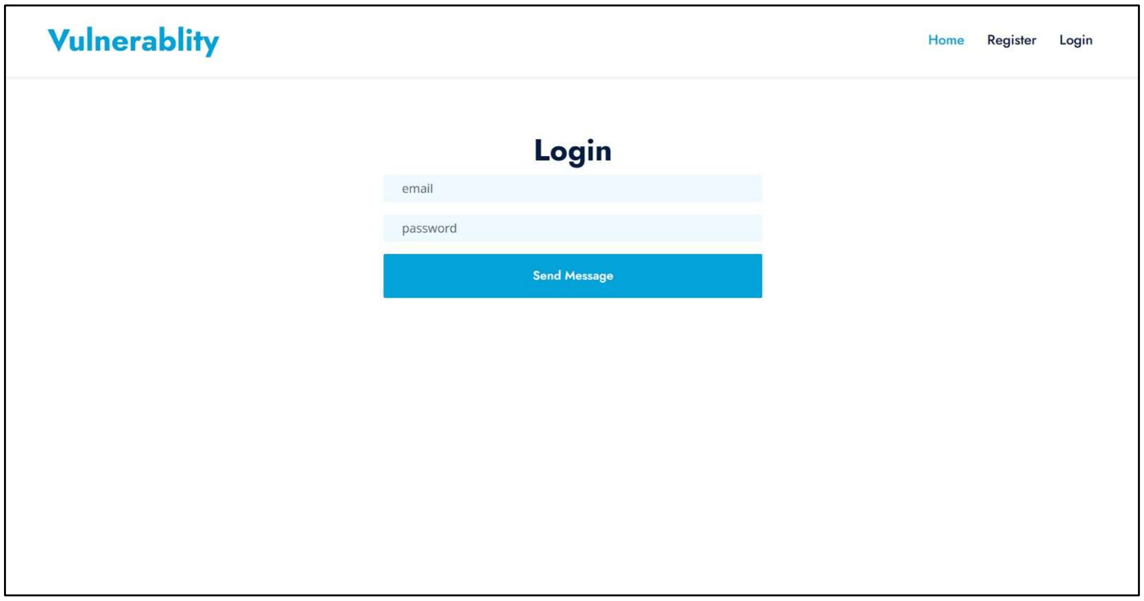
8.1 Steps to Use the Website

Step 1: The website is hosted in the url: http://127.0.0.1:8000/

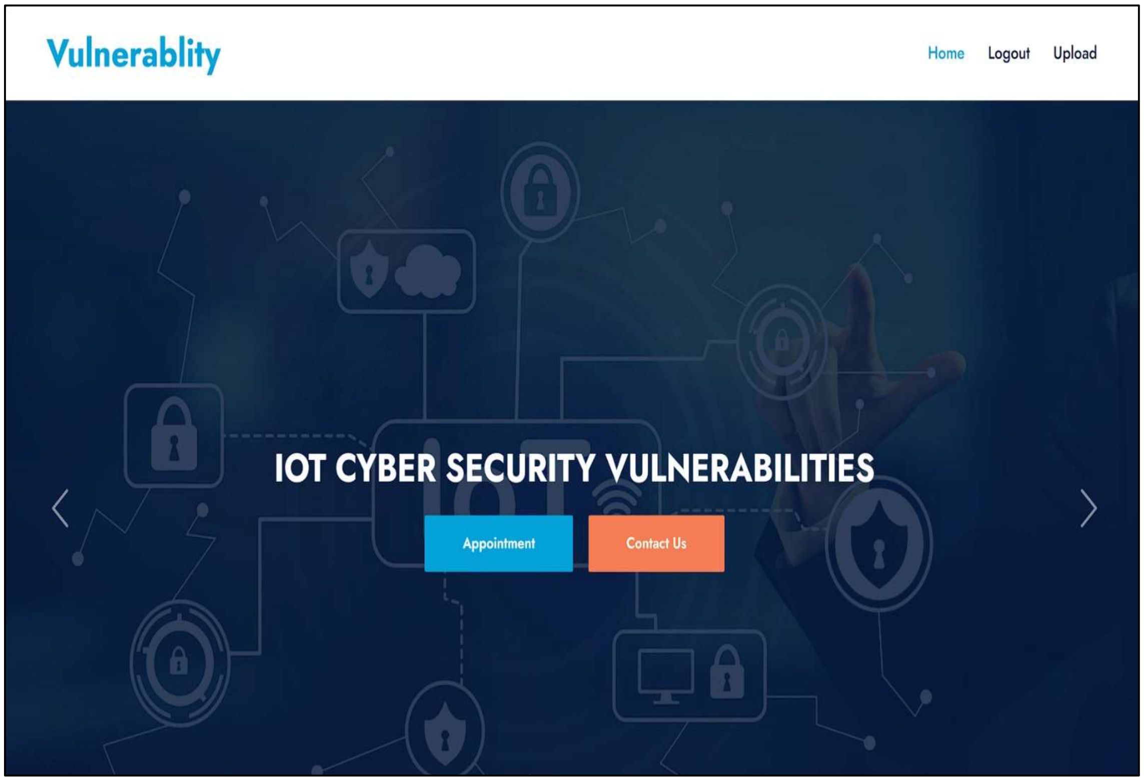
Step 2: Once the url is opened, click on the registration option and fill the necessary details to create your account.



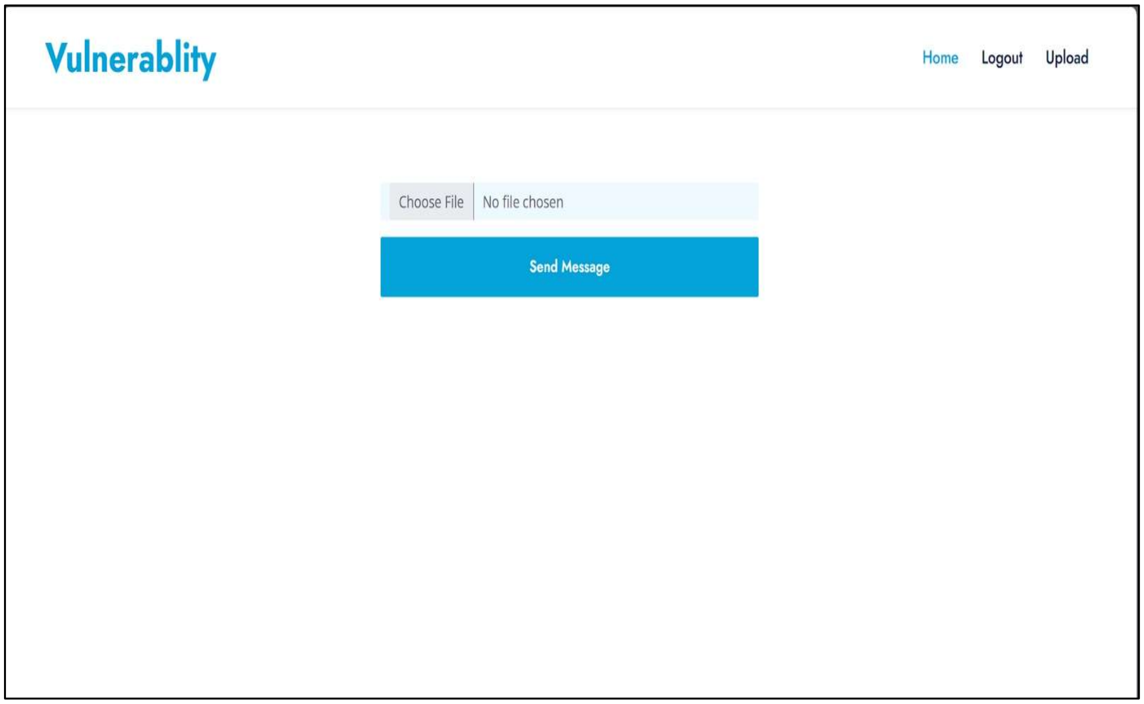
Step 3: After Registration, click Login. Fill the details of the Login page to access your account.



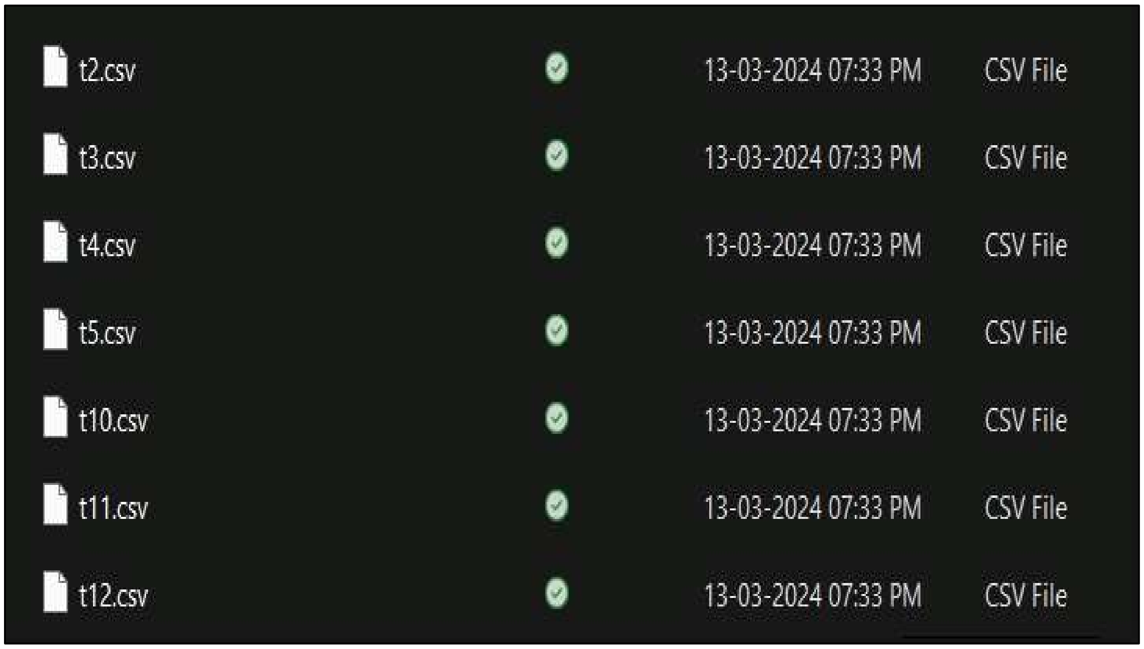
Step 4: After logging in, click on Upload found in the upper corner of the website.



Step 5: Once the Upload page opens, click on the choose file option.

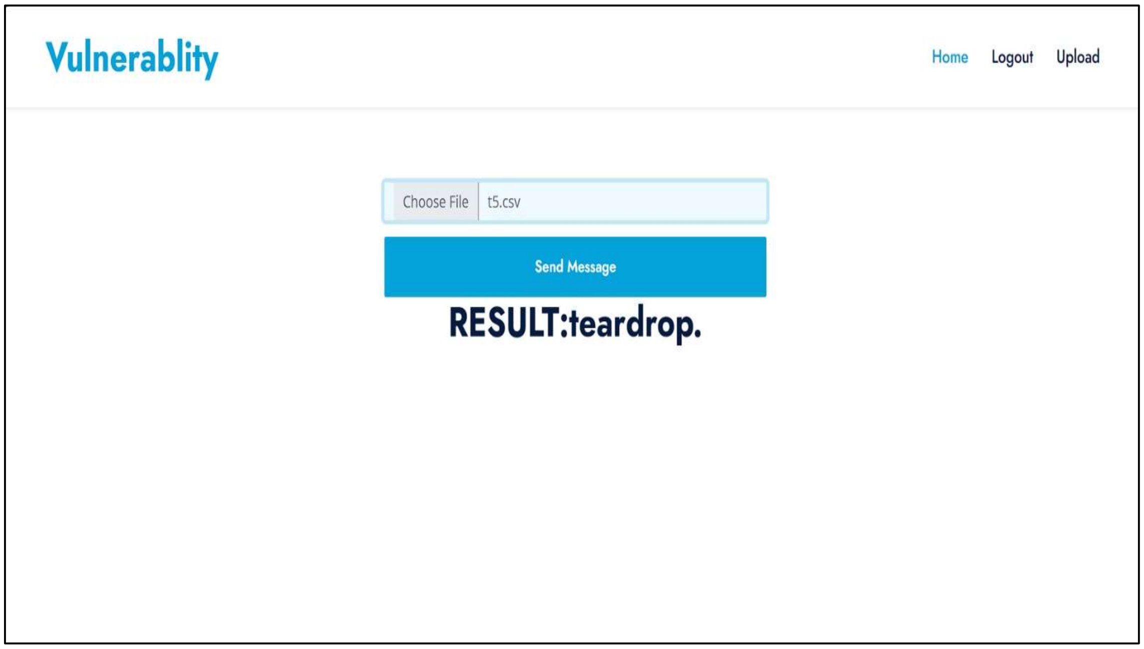


Step 6: Once the file manager is opened, select any one other test files which contains a vulnerability code.



t2.csv	✓	13-03-2024 07:33 PM	CSV File
t3.csv	✓	13-03-2024 07:33 PM	CSV File
t4.csv	✓	13-03-2024 07:33 PM	CSV File
t5.csv	✓	13-03-2024 07:33 PM	CSV File
t10.csv	✓	13-03-2024 07:33 PM	CSV File
t11.csv	✓	13-03-2024 07:33 PM	CSV File
t12.csv	✓	13-03-2024 07:33 PM	CSV File

Step7: After the test file is uploaded, the website will show the vulnerability saved in the test file.



t5.csv

RESULT:teardrop.

CHAPTER: 9

RESULT ANALYSIS

9.1 Training Accuracy

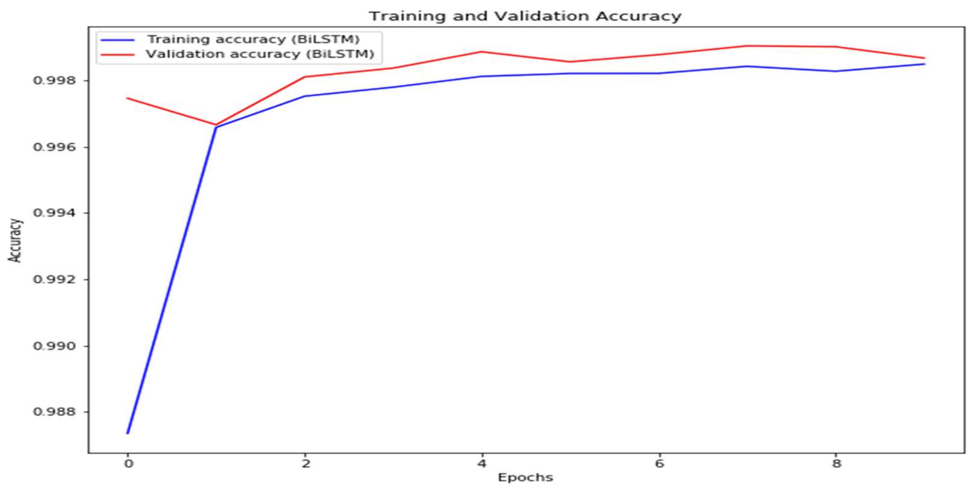


Fig 9.1 Training Validation Accuracy CNN-BiLSTM

The graph shows the relationship between training and validation accuracy of a CNN-BiLSTM model over epochs. The training accuracy (red line) increases steadily, reaching nearly 0.998. The validation accuracy (blue line) also increases, reaching a maximum of 0.992. This suggests the model is learning well and generalizing to unseen data.

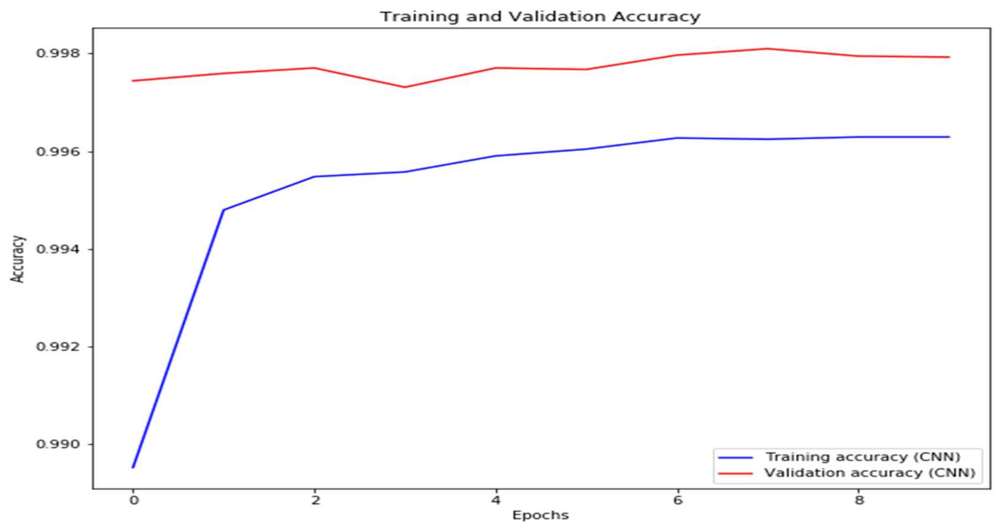


Fig 9.2 Training Validation Accuracy CNN

The graph depicts the training and validation accuracy of a base CNN model over epochs. Both accuracy values (training in red, validation in blue) increase over training epochs, indicating the model is learning effectively. The validation accuracy reaches around 0.992, suggesting good generalizability to unseen data.

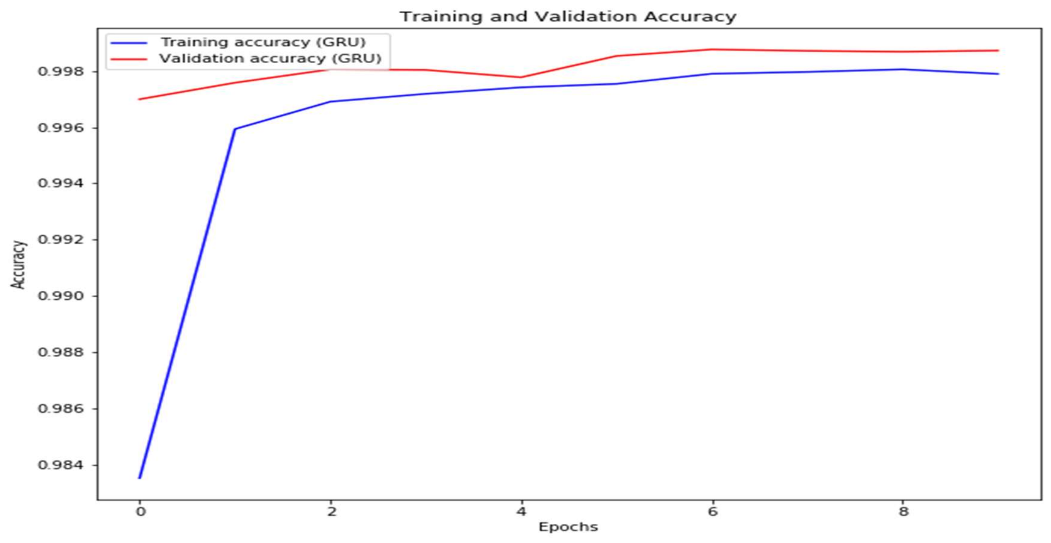
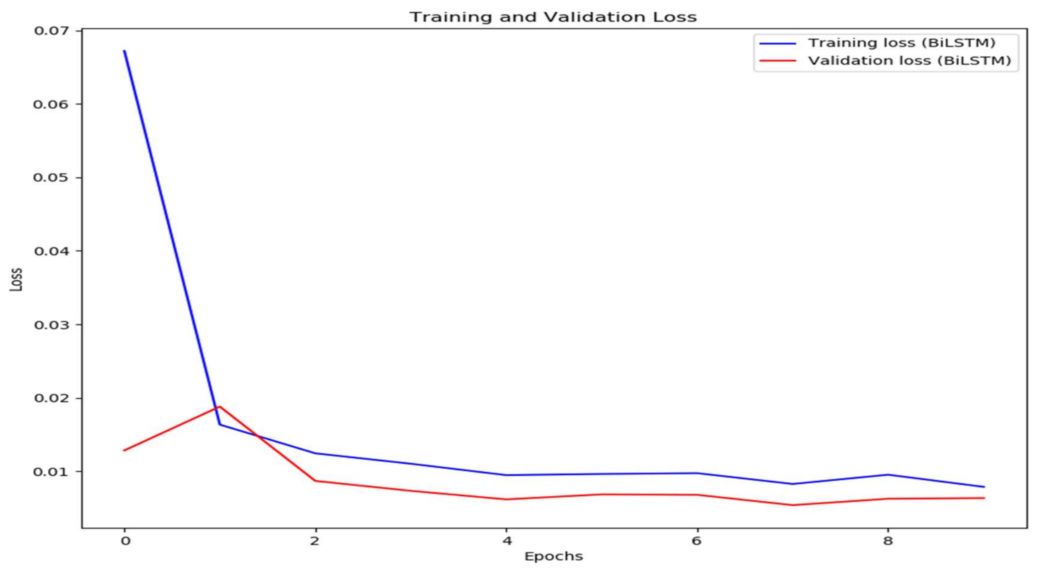


Fig 9.3 Training Validation accuracy GRU

The graph shows training (red) and validation (blue) accuracy of a GRU model over epochs. Both lines rise with training epochs, indicating successful learning. Validation accuracy reaches around 0.99, suggesting the model generalizes well to unseen data.

9.2 Training Loss



+

Fig 9.4 Training validation Loss CNN-BiLSTM

This graph depicts the training and validation loss of a Convolutional Neural Network with Bidirectional Long Short-Term Memory (CNN-BiLSTM) model. Ideally, both training (green) and validation (blue) loss decrease with epochs. A downward trend suggests the model is learning effectively. If validation loss increases while training loss keeps dropping, it might indicate overfitting.

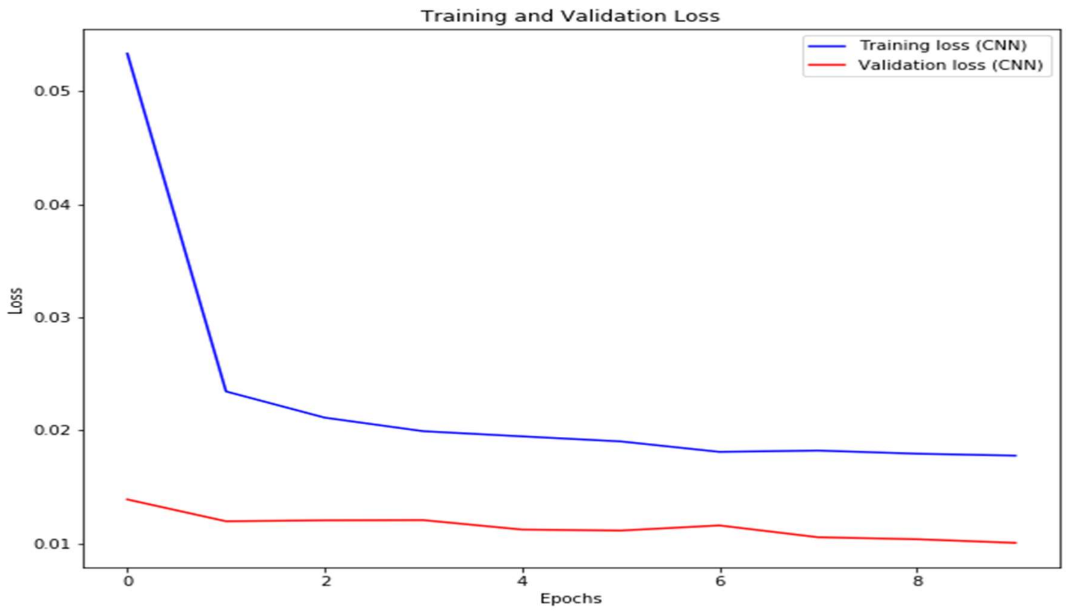


Fig 9.5 Training Validation Loss CNN

This graph depicts the training (red) and validation (blue) loss of a Base Convolutional Neural Network (CNN) model. Ideally, both loss values decrease with training epochs. The downward trend observed here suggests the model is learning effectively from the training data. The validation loss remains relatively low, indicating that the model generalizes well to unseen data.

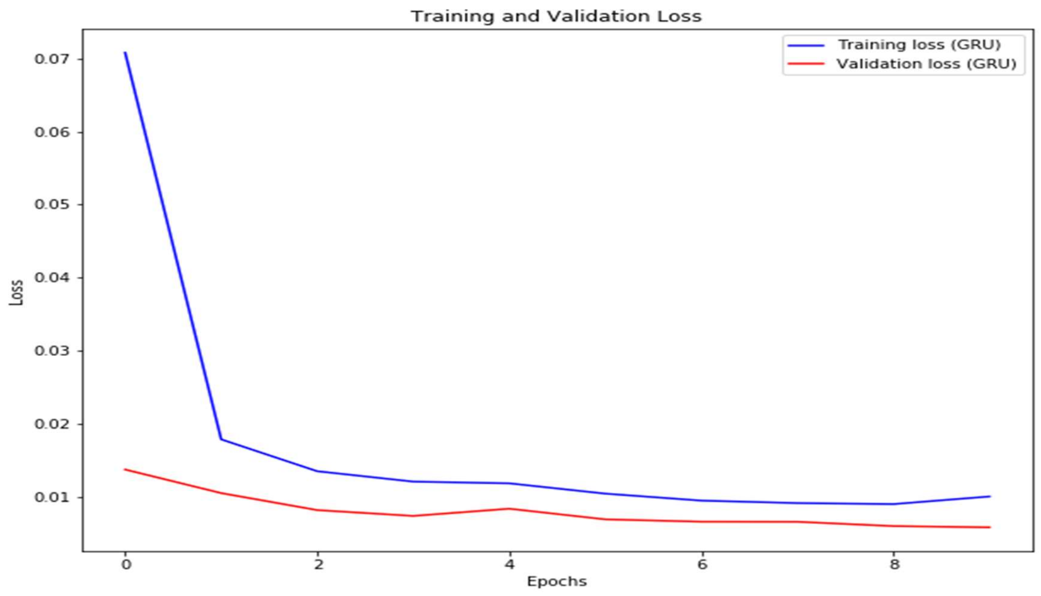
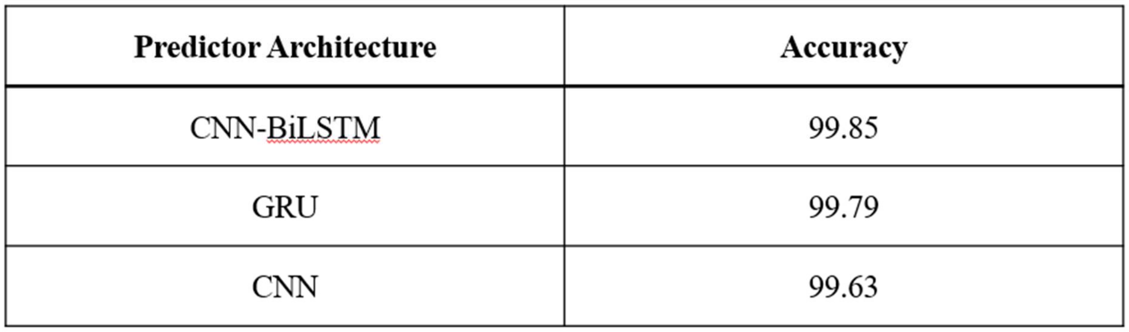


Fig 9.6 Training Validation Loss GRU

This graph shows the training and validation loss of a Gated Recurrent Unit (GRU) model. The training loss (green) decreases steadily over epochs, indicating the model is learning the patterns in the training data. The validation loss (blue) also shows a downward trend, suggesting the model generalizes well to unseen data.



Predictor Architecture	Accuracy
CNN-BiLSTM	99.85
GRU	99.79
CNN	99.63

Fig 9.7 Accuracy Prediction Table

The table shows the accuracy of three different predictor architectures for a natural language processing task. The architectures are CNN-BILSTM, GRU, and CNN. The CNN-BILSTM architecture has the highest accuracy, at 99.85%. The GRU architecture has an accuracy of 99.79%, and the CNN architecture has an accuracy of 99.63%.

In conclusion, the CNN-BILSTM architecture achieved the best performance among the three architectures tested**.**

CHAPTER: 10

CONCLUSION

In conclusion, this project endeavors to assess the efficacy of three distinct deep learning models—CNN, GRU, and CNN-BiLSTM—in the realm of network intrusion detection. By employing various neural network architectures, the objective is to discern which model yields the highest accuracy, hence serving as the optimal choice for detecting potential network intrusions.

Upon conducting extensive experimentation and evaluation, it becomes evident that the CNN-BiLSTM model emerges as the most promising candidate among the three. This conclusion is substantiated by the attained accuracy metrics, where CNN-BiLSTM outperforms both the CNN and GRU architectures.

The provided code facilitates a comprehensive assessment of these models, encompassing essential stages such as data preprocessing, model construction, training, evaluation, and persistence. Notably, each model undergoes rigorous training on the dataset, followed by evaluation on a separate testing set to gauge its performance.

CNN-BiLSTM, characterized by its integration of convolutional and bidirectional LSTM layers, achieves a remarkable accuracy rate of 99.8%. This outstanding performance underscores the effectiveness of leveraging sophisticated architectures capable of capturing both spatial and temporal dependencies within network traffic data.

In summary, this project underscores the critical role of deep learning models, particularly CNN-BiLSTM, in bolstering network security through robust intrusion detection mechanisms. By leveraging advanced neural network architectures and meticulous evaluation methodologies, it lays the foundation for developing resilient and adaptive network intrusion detection systems capable of safeguarding against a myriad of cyber threats.

APPENDICES

SOURCE CODE

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import classification_report

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense,
Dropout, Flatten

from tensorflow.keras.utils import to_categorical


# Replace 'path/to/kddcup.data_10_percent' with the actual path to your file

file_path = 'kddcup.data_10_percent'


# Define column names based on the dataset documentation

column_names = [

    'duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes',

    'land', 'wrong_fragment', 'urgent', 'hot', 'num_failed_logins',

    'logged_in', 'num_compromised', 'root_shell', 'su_attempted',

    'num_root', 'num_file_creations', 'num_shells', 'num_access_files',

    'num_outbound_cmds', 'is_host_login', 'is_guest_login', 'count',

    'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate',

    'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate',

    'dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate',

    'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',

    'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',

    'dst_host_srv_serror_rate', 'dst_host_rerror_rate',

    'dst_host_srv_rerror_rate', 'label']
```

```
# Read the dataset into a pandas DataFrame
df = pd.read_csv(file_path, header=None, names=column_names)

# Display the first few rows of the DataFrame
print(df.head())

unique_labels = df['label'].unique()
print("Unique Labels:", unique_labels)

# Assuming df is your DataFrame with non-integer valued columns
non_integer_columns = df.select_dtypes(exclude=['int64', 'float64']).columns
label_encoder = LabelEncoder()
for column in non_integer_columns:
    df[column] = label_encoder.fit_transform(df[column])

unique_labels = df['label'].unique()
print("Unique Labels:", unique_labels)

# Encode categorical labels
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['label'])

# Split the dataset into features and labels
X = df.drop('label', axis=1)
y = df['label']

# Standardize numerical features
numerical_columns = X.select_dtypes(include=np.number).columns
scaler = StandardScaler()
X[numerical_columns] = scaler.fit_transform(X[numerical_columns])
```

```
# Convert labels to categorical format
y = to_categorical(y)

# Reshape the input data for the Conv1D layer
X = X.values.reshape(X.shape[0], X.shape[1], 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the CNN-BiLSTM model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Bidirectional(LSTM(50, activation='relu')))
model.add(Dropout(0.5))
model.add(Dense(23, activation='softmax'))  # Adjust the number of units based on your problem

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history_bilstm = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model on the test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)
print(classification_report(y_test_classes, y_pred_classes))

# Assuming 'model' is your trained Keras model
```

```
model.save('model_bilstm.h5')


from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, GRU, Dropout, Dense

# Encode categorical labels
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['label'])

# Split the dataset into features and labels
X = df.drop('label', axis=1)
y = df['label']

# Standardize numerical features
numerical_columns = X.select_dtypes(include=np.number).columns
scaler = StandardScaler()
X[numerical_columns] = scaler.fit_transform(X[numerical_columns])

# Convert labels to categorical format
y = to_categorical(y)

# Reshape the input data for the Conv1D layer
X = X.values.reshape(X.shape[0], X.shape[1], 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Build the CNN-GRU model

model = Sequential()

model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X.shape[1], 1)))

model.add(MaxPooling1D(pool_size=2))

model.add(GRU(50, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(23, activation='softmax'))  # Adjust the number of units based on your problem


# Compile the model

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


# Print the model summary

model.summary()


# Train the model

history_gru = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))


# Assuming 'model' is your trained Keras model

model.save('model_gru.h5')


from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from keras.utils import to_categorical

from keras.models import Sequential

from keras.layers import Conv1D, MaxPooling1D, Dropout, Flatten, Dense


# Encode categorical labels

label_encoder = LabelEncoder()
```

```
df['label'] = label_encoder.fit_transform(df['label'])

# Split the dataset into features and labels
X = df.drop('label', axis=1)
y = df['label']

# Standardize numerical features
numerical_columns = X.select_dtypes(include=np.number).columns
scaler = StandardScaler()
X[numerical_columns] = scaler.fit_transform(X[numerical_columns])

# Convert labels to categorical format
y = to_categorical(y)

# Reshape the input data for the Conv1D layer
X = X.values.reshape(X.shape[0], X.shape[1], 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the CNN model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(23, activation='softmax'))  # Adjust the number of units based on your
problem
```

```
# Compile the model

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


# Print the model summary

model.summary()


# Train the model

history_cnn = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test,
y_test))


# Assuming 'model' is your trained Keras model

model.save('model_cnn.h5')


import matplotlib.pyplot as plt


# Plot the loss and accuracy curves for training and validation

fig, ax = plt.subplots(1, 1, figsize=(10, 8))  # Create a 3x1 grid of subplots


# First subplot: Loss

ax.plot(history_bilstm.history['loss'], color='b', label="Training loss (BiLSTM)")

ax.plot(history_bilstm.history['val_loss'], color='r', label="Validation loss (BiLSTM)")

ax.set_title("Training and Validation Loss")

ax.set_xlabel("Epochs")

ax.set_ylabel("Loss")

ax.legend(loc='best')


# Save the figure

plt.savefig("training_validation_loss_Bilstm.png")

import matplotlib.pyplot as plt

# Plot the loss and accuracy curves for training and validation

fig, ax = plt.subplots(1, 1, figsize=(10, 8))  # Create a 3x1 grid of subplots
```

```
# First subplot: Loss
ax.plot(history_gru.history['loss'], color='b', label="Training loss (GRU)")
ax.plot(history_gru.history['val_loss'], color='r', label="Validation loss (GRU)")
ax.set_title("Training and Validation Loss")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.legend(loc='best')

# Save the figure
plt.savefig("training_validation_loss_gru.png")

import matplotlib.pyplot as plt

# Plot the loss and accuracy curves for training and validation
fig, ax = plt.subplots(1, 1, figsize=(10, 8))  # Create a 3x1 grid of subplots

# First subplot: Loss
ax.plot(history_cnn.history['loss'], color='b', label="Training loss (CNN)")
ax.plot(history_cnn.history['val_loss'], color='r', label="Validation loss (CNN)")
ax.set_title("Training and Validation Loss")
ax.set_xlabel("Epochs")
ax.set_ylabel("Loss")
ax.legend(loc='best')

# Save the figure
plt.savefig("training_validation_loss_cnn.png")
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
ax.plot(history_bilstm.history['acc'], color='b', label="Training accuracy (BiLSTM)")
ax.plot(history_bilstm.history['val_acc'], color='r', label="Validation accuracy (BiLSTM)")
ax.set_title("Training and Validation Accuracy")
ax.set_xlabel("Epochs")
ax.set_ylabel("Accuracy")
ax.legend(loc='best')
plt.savefig("training_validation_accuracy_bILSTM.png")

import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
ax.plot(history_gru.history['acc'], color='b', label="Training accuracy (GRU)")
ax.plot(history_gru.history['val_acc'], color='r', label="Validation accuracy (GRU)")
ax.set_title("Training and Validation Accuracy")
ax.set_xlabel("Epochs")
ax.set_ylabel("Accuracy")
ax.legend(loc='best')
plt.savefig("training_validation_accuracy_GRU.png")

import matplotlib.pyplot as plt
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
ax.plot(history_cnn.history['acc'], color='b', label="Training accuracy (CNN)")
ax.plot(history_cnn.history['val_acc'], color='r', label="Validation accuracy (CNN)")
ax.set_title("Training and Validation Accuracy")
ax.set_xlabel("Epochs")
ax.set_ylabel("Accuracy")
ax.legend(loc='best')
plt.savefig("training_validation_accuracy_cnn.png")

from tensorflow.keras.models import load_model
loaded_model = load_model('model_bilstm.h5')
```

```
# Save the 20th row to a new DataFrame for testing
test_data = df.iloc[[116000]]

# Save the test data to a CSV file
test_data.to_csv('test_data1.csv', index=False)

# Display the saved test data
print("Test Data:")
print(test_data)

import pandas as pd
import numpy as np
from tensorflow.keras.models import load_model

# Load the pre-trained model
model = load_model('model_bilstm.h5')  # Replace with the actual path to your trained model

# Load the test data
test_data = pd.read_csv('test_data.csv')  # Replace with the actual path to your test data

# Extract features and labels from the test data
X_test = test_data.drop('label', axis=1)
y_test = test_data['label']

# If label encoding was used during training, you may need to encode the labels
# label_encoder = LabelEncoder()
# y_test = label_encoder.transform(y_test)

# Standardize numerical features (if used during training)
# X_test[numerical_columns] = scaler.transform(X_test[numerical_columns])
# Assuming you've reshaped the input data during training
X_test = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)

# Make predictions
predictions = model.predict(X_test)

# Convert predictions to class labels
predicted_class_indices = np.argmax(predictions, axis=1)

# Map indices to original class labels
# inverse_transform if label encoding was used during training
# predicted_labels = label_encoder.inverse_transform(predicted_class_indices)

# Print predictions
print("Predicted Class Indices:", predicted_class_indices)
# print("Predicted Labels:", predicted_labels)

test_data1 = df.iloc[[29]]
test_data1.to_csv('test.csv', index=False)
```

REFERENCES

1. Impact of False Data Injection Attacks on Deep Learning Enabled Predictive Analytics Gautam Raj Mode, Prasad Calyam, Khaza Anuarul Hoque Department of Electrical Engineering & Computer Science University of Missouri, Columbia, MO, USA gmwyc@mail.missouri.edu, calyamp@missouri.edu, hoquek@missouri.edu

2. Intrusion Detection in IoT Using Deep Learning Alaa Mohammed Banaamah and Iftikhar Ahmad

3. Zhang, J.; Pan, L.; Han, Q.-L.; Chen, C.; Wen, S.; Xiang, Y. Deep learning-based attack detection for cyber-physical system cybersecurity: A survey. IEEE/CAA J. Autom. Sin. 2021, 9, 377–391.

4. M. A. der Mauer, T. Behrens, M. Derakhshanmanesh, C. Hansen, and S. Muderack, “Applying sound-based analysis at porsche production: Towards predictive maintenance of production machines using deep learning and internet-of-things technology,” in Digitalization Cases. Springer, 2019, pp. 79–97.

5. Vulnerability Studies and Security Postures of IoT Devices: A Smart Home Case Study Brittany D. Davis, Janelle C. Mason, and Mohd Anwar

6. R. K. Mobley, An introduction to predictive maintenance. Elsevier, 2002.

7. The US Air Force Is Adding Algorithms to Predict When Planes Will Break, Defense One Magazine.” Available: https://www.defenseone.com/business/2018/05/us-air-force-addingalgorithms-predict-when-planes-will-break/148234/.

8. IOT Use Cases and Innovation in IOT,” Available: https://medium.com/@billsoftnet/iot-use-cases-and-innovation-in-iot-6b4e49fbc9dc.

9. R. Caponetto, F. Rizzo, L. Russotti, and M. Xibilia, “Deep learning algorithm for predictive maintenance of rotating machines through the analysis of the orbits shape of the rotor shaft,” in International Conference on Smart Innovation, Ergonomics and Applied Human Factors. Springer, 2019, pp. 245–250.

10. Predictive maintenance benefits for the freight logistics industr,” Available: https://www.ibm.com/downloads/cas/AVNOLWQW.