A

Major Project Report

on

Effective Multitask Deep learning for IOT Malware Detection and Identification Using Behavioural Traffic Analysis

submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

by

A. Rajavamshi Goud

(20EG105454)

K. Vinay

(20EG105458)

T. Ajay

(20EG105459)

Under the guidance of
Dr.B.V.V.Siva Prasad
Associate Professor
Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANURAG UNIVERSITY
VENKATAPUR– 500088
TELANGANA
Year 2023-24

DECLARATION

We hereby declare that the report entitled "**Effective Multitask Deep learning for IOT Malware Detection and Identification Using Behavioural Traffic Analysis**" submitted to the **Anurag University** is partial fulfilment of the requirements for the award of **Bachelor of technology(B. Tech)** Degree in **Computer Science and Engineering** is record of original work by us under the guidance of **Dr.B.V.V.Siva Prasad, Associate Professor** and this report has not been submitted to any other university for the award of any other degree or diploma.

Place: Anurag University, Hyderabad
Date : 10/04/2024

A. Rajavamshi Goud
(20EG105454)

K. Vinay
(20EG105454)

T. Ajay
(20EG105459)

CERTIFICATE

This is to certify that the project report entitled "**Effective Multitask Deep learning for IOT Malware Detection and Identification Using Behavioural Traffic Analysis**" being submitted by **A. Rajavamshi Goud** bearing the Hall Ticket number **20EG105454, K. Vinay** bearing the Hall Ticket number **20EG105458, T. Ajay** bearing the Hall Ticket number **20EG105459** in partial fulfillment of the requirements for the award of degree of the **Bachelor of Technology** in **Computer Science and Engineering** to **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision for the academic year 2023 to 2024.

The results embodied in this Report have been verified and found to be satisfactory. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of Supervisor
Dr.B.V.V.Siva Prasad
Associate Professor
Department of CSE

Signature of Dean
Dr. G. Vishnu Murthy
Department of CSE

External Examiner

ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Dr.B.V.V.Siva Prasad, Associate Professor, Department of Computer Science and Engineering,** Anurag University for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. His patience, guidance and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy** , Dean, Department of Computer Science and Engineering, Anurag University. We also express our deep sense of gratitude to **Dr. V V S S S Balaram**, Academic co-ordinator, **Dr .Pallam Ravi**, Project Co-ordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of our project work.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for his encouragement and timely support in our B. Tech program.

A. Rajavamshi Goud
(20EG105454)

K. Vinay
(20EG105458)

T. Ajay
(20EG105459)

ABSTRACT

The pervasive presence of Internet of Things (IoT) devices has accentuated security concerns, particularly with the emergence of IoT-specific malware. To address this challenge comprehensively, we propose a multifaceted approach utilizing Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), and CNN+LSTM architectures for IoT malware detection. Our study encompasses a diverse dataset comprising 145 .pcap files, capturing benign and malicious traffic from 18 distinct IoT devices. We conduct an extensive experimental evaluation, comparing the performance of LSTM, CNN, and CNN+LSTM models in detecting and classifying IoT malware.

Feature extraction involves flow-related, traffic flag-related, and packet payload-related features, which undergo meticulous feature selection across all models. By leveraging the heterogeneity of the dataset, we aggregate data from multiple IoT devices with various malware attacks, facilitating a comprehensive analysis. Experimental results reveal nuanced performance differences among the models. While LSTM demonstrates robust performance in certain scenarios, CNN and CNN+LSTM architectures exhibit strengths in others. Ultimately, the model with the best predictive performance is selected for malware identification and prediction. Our findings underscore the importance of considering a wide range of technologies in IoT malware detection. Furthermore, they highlight the need for tailored approaches based on specific characteristics of IoT traffic and malware behavior.

In future research, we aim to further refine and optimize the selected model, potentially exploring hybrid architectures and incorporating additional datasets to enhance generalizability. Additionally, ongoing efforts will focus on integrating supplementary modules, such as device identification based on IoT traffic, to bolster overall cybersecurity measures.

Table of Content

List of Figures

List of Tables

1. INTRODUCTION

1.1 Evolution of Cyberattacks in IOT Environments:

The Internet of Things (IoT) has revolutionized numerous domains, spanning automobiles, smart homes and cities, manufacturing, healthcare, retail, and beyond. This revolution is propelled by the exponential growth of portable Internet-connected devices. These intelligent devices continuously evolve in technical sophistication and manufacturing simplicity, promising unparalleled convenience and efficiency. However, this advancement also ushers in a myriad of security challenges. The 2020 IoT threat report underscores the urgent necessity for heightened security measures in response to the emergence of IoT-specific malware. Notably, the infamous Mirai botnet epitomizes the severity of these security concerns. Mirai gained notoriety for orchestrating one of the largest Distributed Denial-of-Service (DDoS) attacks on record, targeting Dyn. This attack caused widespread disruption across major online platforms and services.

The prominence of Mirai and its variants highlights the critical imperative of addressing IoT security vulnerabilities. As IoT devices become increasingly pervasive in daily life, robust security measures are essential to safeguarding the integrity of IoT ecosystems. In subsequent sections, this report will delve into the challenges posed by IoT security threats and propose strategies for mitigating these risks effectively. The Mirai botnet rose to infamy by orchestrating a significant Distributed Denial-of-Service (DDoS) assault, which stands as one of the most extensive attacks of its kind to date, with its primary target being Dyn. This assault caused widespread disruption across various internet services, showcasing the immense threat posed by IoT-specific malware. Compounding this issue, the subsequent release of the Mirai source code has spurred the development of highly sophisticated malware variants, including but not limited to Satori, Hajime, and BrickerBot. In response to the proliferation of these advanced malware strains, researchers have escalated their efforts to devise robust countermeasures aimed at mitigating IoT security vulnerabilities. However, their endeavors encounter a significant obstacle in the

form of insufficient empirical data on the current landscape of IoT malware. This scarcity of data hampers the development of comprehensive strategies for addressing IoT security threats effectively. Additionally, a thorough understanding of the behaviors exhibited by malware-infected devices is essential for devising targeted mitigation tactics. Thus, overcoming these challenges is imperative for ensuring the continued security and integrity of IoT ecosystems.

This study delineates two primary challenges within the realm of IoT security: safeguarding devices against malware attacks and categorizing IoT malware based on the traffic it generates. Although the overarching goal remains the protection of devices, the evolution of increasingly sophisticated ransomware renders the achievement of complete device security an elusive goal. Consequently, there emerges a critical need to classify different types of malware to facilitate the implementation of targeted mitigation strategies. While traditional efforts primarily focus on fortifying devices against malware infiltration, the rapid evolution and proliferation of advanced ransomware strains necessitate a more nuanced approach. Rather than solely aiming for device hardening, it is essential to develop mechanisms capable of accurately identifying and categorizing various malware types. By doing so, targeted mitigation strategies can be devised, tailored to the specific characteristics and behaviors exhibited by different types of malware.

In essence, while device protection remains paramount, the advent of highly sophisticated ransomware underscores the importance of proactive classification efforts. Through the development of robust classification mechanisms, organizations can better prepare themselves to combat the diverse array of IoT security threats effectively. Existing literature accentuates the significance of IoT vulnerabilities and highlights the necessity for enhanced intrusion detection algorithms. However, validating the effectiveness of these algorithms poses a considerable challenge due to the distinctive characteristics of IoT traffic. Traditional Machine Learning (ML) methodologies, while proficient, demand domain expertise and extensive feature engineering, which

constrains their scalability. In contrast, Deep Learning (DL) techniques offer a distinct advantage by autonomously learning features directly from raw data, thereby obviating the need for labor-intensive manual feature extraction processes. The emphasis on IoT vulnerabilities underscores the critical importance of developing robust intrusion detection mechanisms tailored to the intricacies of IoT environments. However, traditional ML approaches often struggle to cope with the complexity and variability of IoT data due to their reliance on handcrafted features and domain-specific expertise. This limitation inhibits their scalability and adaptability to evolving threat landscapes.

In contrast, DL techniques, particularly those employing neural network architectures, exhibit a remarkable capacity to learn intricate patterns and relationships directly from raw data. By bypassing the need for manual feature extraction, DL models can effectively capture the nuanced characteristics of IoT traffic, enabling more accurate and scalable intrusion detection solutions. Consequently, the adoption of DL approaches represents a promising avenue for addressing the challenges posed by IoT security, offering the potential to enhance detection accuracy and scalability while reducing the burden of feature engineering.

To tackle these aforementioned challenges, this research presents a novel approach: a Long Short-Term Memory (LSTM) Deep Learning (DL)-based multitask intrusion detection model. This model is designed to leverage the heterogeneous characteristics of datasets, encompassing diverse IoT devices and attack types, with the objective of enhancing intrusion detection accuracy and robustness. The study contributes to the field by proposing a multitask LSTM-based DL model specifically tailored for IoT security and malware classification. Additionally, it delves into the functionality of LSTM models when applied to heterogeneous time series data. Furthermore, the proposed model's performance is comprehensively evaluated through a series of experiments. By employing a multitask approach, the model aims to simultaneously address multiple aspects of IoT security, including intrusion detection and malware classification, thereby offering a holistic solution to the

complex challenges posed by IoT security threats. Through the investigation of LSTM models with heterogeneous data, the research aims to provide insights into the model's adaptability and effectiveness across diverse IoT environments and attack scenarios. The comprehensive evaluation of the proposed model's performance ensures its efficacy in real-world applications and contributes to advancing the state-of-the-art in IoT security research.

The significance of this research lies in its potential to enhance the detection and classification of IoT-related security threats, thereby bolstering the resilience of IoT ecosystems against malicious activities. Additionally, by leveraging DL techniques and multitask learning, the proposed model offers scalability and adaptability to evolving threat landscapes. Overall, this study represents a valuable contribution to the field of IoT security, with implications for enhancing the security and integrity of IoT systems across various domains and applications.

1.2 Problem Illustration:

Imagine a smart home environment with various IoT devices, including smart thermostats, security cameras, and smart locks. Each device generates distinct patterns of network traffic based on its specific functionalities and interactions with other devices and services.

Data Preparation:

Suppose we have a dataset representing IoT network traffic, where each data point consists of a sequence of 10 features (e.g., packet size, protocol type, source IP address, destination IP address, etc.). Each data point is labeled as either benign (0) or malicious (1).

Model Training:

We'll train logistic regression model and random forest classifier on the same dataset.

Results:

Traditional ML Model (Logistic Regression):

- Accuracy: 0.75
- Precision: 0.74
- Recall: 0.76
- F1 Score: 0.75

Traditional ML Model (Random Forest):

- Accuracy: 0.78
- Precision: 0.77
- Recall: 0.79
- F1 Score: 0.78

1.3 Disadvantages by the Traditional ML models:

Complex Traffic Patterns: The malware's traffic patterns may be highly complex and dynamic, making it difficult for Random Forest to accurately distinguish between benign and malicious traffic. Random Forest may struggle to identify subtle deviations or anomalies in the traffic patterns indicative of malicious behavior.

Limited Feature Representation: Random Forest relies on predefined features extracted from network packets for classification. However, the features used may not adequately capture the nuanced characteristics of IoT device traffic, especially in the presence of sophisticated malware. As a result, the model may fail to effectively differentiate between normal and malicious traffic.

Scalability Issues: As the number of IoT devices and the volume of network traffic increase, Random Forest may encounter scalability issues. Processing large-scale data sets in real-time or near-real-time may strain computational resources and lead to performance degradation.

Missed Malware Detection: Since the existing approach primarily focuses on device identification and traffic classification, it may lack specific mechanisms to detect and mitigate IoT malware. Consequently, the malware may go undetected, posing a significant security risk to the smart home network and its users.

In contrast, leveraging more advanced deep learning techniques like LSTM allows for the creation of more sophisticated models capable of capturing complex temporal dependencies in IoT device traffic. These models can adapt to evolving traffic patterns and detect subtle anomalies indicative of malware activity, thereby enhancing the security posture of IoT environments.

2. LITERATURE SURVEY

The widespread adoption of cloud computing environments has become a cornerstone of modern digital infrastructure, offering scalability, flexibility, and cost-efficiency for various services and applications. However, the inherent reliance on virtualized computer and network infrastructures within cloud environments introduces unique risks and challenges, particularly concerning system resilience and security. One of the primary concerns is the lack of comprehensive research and understanding regarding the security implications of virtualized environments within the cloud. Despite their prevalence, the virtualized nature of cloud computing systems has not been thoroughly examined concerning vulnerabilities and appropriate anomaly detection mechanisms. This knowledge gap leaves cloud-based systems vulnerable to potential security breaches and malware attacks.

To address these concerns, this paper proposes an innovative approach for investigating and analyzing malware within virtualized environments. The proposed approach entails conducting a comprehensive analysis on both system and network-wide scales, with a particular focus on identifying and understanding specific system and network features relevant to malware behavior. As a case study, the paper examines the Kelihos malware, a well-known and widely distributed botnet responsible for various malicious activities, including spam campaigns, distributed denial-of-service (DDoS) attacks, and data theft. By studying the behavior of Kelihos within virtualized environments, the paper aims to uncover insights into how malware operates in such environments and how it may exploit vulnerabilities inherent in virtualization technologies. Through this analysis, the paper seeks to contribute

to the advancement of security measures and practices within cloud computing environments. By identifying and understanding the unique challenges posed by malware in virtualized settings, researchers and practitioners can develop more effective detection and mitigation strategies tailored to the cloud environment's specific characteristics.

Overall, the proposed approach offers valuable insights into the intersection of virtualization technology and malware analysis, paving the way for enhanced security measures and resilience within cloud computing ecosystems.

Network Traffic Flow Based Machine Learning Technique for IoT Device Identification:

Security and privacy issues are being raised as smart systems are integrated into our daily lives. New security issues have emerged with several new vendors that develop the Internet of Things (IoT) products. The contents and patterns of network traffic will expose vulnerable IoT devices to intruders. New methods of network assessment are needed to evaluate the type of network connected IoT devices. IoT device recognition would provide a comprehensive structure for the development of stable IoT networks. This paper chooses a machine learning technique to identify IoT devices linked to the network by analyzing network flow sent and received. To generate network traffic data, we have developed a dataset adapted from the IoT23 Pcap files to experiment with a smart home network. We have created a model to identify the IoT device based on network traffic analysis. We evaluate our proposed model via full features dataset, reduces features dataset, and flow-based features dataset. This paper focuses on using flow-based features to identify the IoT device connected to the network. Our proposed scheme results in 100% precision, precision, recall, and F score via a full features dataset, reduced features dataset, and flow-based features dataset. Through evaluations using our produced dataset, we demonstrate that the proposed model can accurately classify IoT devices.

Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges:

The massive adoption of hand-held devices has led to the explosion of mobile traffic volumes traversing home and enterprise networks, as well as the Internet. Traffic classification (TC), i.e., the set of procedures for inferring (mobile) applications generating such traffic, has become nowadays the enabler for highly valuable profiling information (with certain privacy downsides), other than being the workhorse for service differentiation/blocking. Nonetheless, the design of accurate classifiers is exacerbated by the raising adoption of encrypted protocols (such as TLS), hindering the suitability of (effective) deep packet inspection approaches. Also, the fast-expanding set of apps and the moving-target nature of mobile traffic makes design solutions with usual machine learning, based on manually and expert-originated features, outdated and unable to keep the pace. For these reasons deep learning (DL) is here proposed, for the first time, as a viable strategy to design practical mobile traffic classifiers based on automatically extracted features, able to cope with encrypted traffic, and reflecting their complex traffic patterns. To this end, different state-of-the-art DL techniques from (standard) TC are here reproduced, dissected (highlighting critical choices), and set into a systematic framework for comparison, including also a performance evaluation workbench. The latter outcome, although declined in the mobile context, has the applicability appeal to the wider umbrella of encrypted TC tasks. Finally, the performance of these DL classifiers is critically investigated based on an exhaustive experimental validation (based on three mobile datasets of real human users' activity), highlighting the related pitfalls, design guidelines, and challenges.

A Deep Learning Approach to Network Intrusion Detection:

Network intrusion detection systems (NIDSs) serve as critical components in safeguarding computer networks against malicious activities. However, contemporary approaches face growing concerns regarding their effectiveness and sustainability in meeting the evolving demands of modern networks. Notably, these concerns revolve around the escalating levels of required human intervention and the declining accuracy of intrusion detection. To address these

Author(s)	Strategies	Advantages	Disadvantages
Singh et al	Statistical Methods and methods based on graph's spectral heat and wave signatures	Traditional statistical methods provide a well-established foundation on network security.Techniques like static malware analysis contribute to understanding IoT malware behavior.	Limited availability of publicly accessible botnet datasets for comparison. Absence of appropriate benchmarks for botnet identification algorithms.
Shafiq et al	supervised ML algorithms (SVM, ensemble learning,)	Supervised ML methods achieve high accuracy in attack detection.	Evaluation on limited datasets may restrict the generalization of ML models.
Lagraa et al	unsupervised ML methods	Unsupervised ML methods learn from unlabeled datasets to identify abnormalities.	Accuracy is not as expected.
Francois et al and Dib et al.	entropy measures and String based features	Entropy measures are easily adaptable and generic in nature and String based approach offers a targeted approach for know patterns.	Entropy measures and string-based features may not capture complex patterns as effectively as ML or DL methods

Table 2.1 Comparison of Literature

challenges, this paper introduces a novel deep learning technique tailored for intrusion detection, aimed at mitigating the limitations of existing approaches. Specifically, the proposed technique revolves around the concept of nonsymmetric deep autoencoder (NDAE) for unsupervised feature learning. Additionally, the paper proposes a novel deep learning classification model constructed using stacked NDAEs.

The proposed classifier is implemented using graphics processing unit (GPU)-enabled TensorFlow and evaluated using benchmark datasets, namely the KDD Cup '99 and NSL-KDD datasets. Preliminary results obtained from the model demonstrate promising outcomes, showcasing improvements over conventional approaches and indicating strong potential for application in modern NIDSs. By leveraging deep learning methodologies, particularly NDAEs, the proposed technique offers enhanced capabilities in learning intricate features from network data, thereby bolstering the accuracy and efficiency of intrusion detection. Furthermore, the utilization of GPU-enabled TensorFlow facilitates efficient model training and evaluation, enabling scalability to handle large-scale network datasets.

Overall, the findings presented in this paper underscore the efficacy of deep learning techniques in addressing the challenges faced by traditional intrusion detection systems. The proposed approach not only demonstrates superior performance but also signifies a significant step towards the development of next-generation NIDSs capable of effectively combating evolving cyber threats in modern network environments.

Characterization of Tor Traffic using Time based Features:

Traffic classification has been a focal point of numerous research endeavors; however, the rapid evolution of Internet services and the widespread adoption of encryption pose ongoing challenges in this domain. Encryption plays a pivotal role in safeguarding the privacy of Internet users and is a fundamental technology employed in various privacy-enhancing tools that have emerged in recent years. One such prominent tool is Tor, which facilitates anonymous communication by encrypting traffic between the sender and receiver and

routing it through a decentralized network of servers. Tor's architecture aims to obfuscate the origin and destination of Internet traffic, thereby enhancing user privacy and anonymity.

In this paper, we present a temporal analysis of Tor traffic flows captured between the client and the entry node within the Tor network. We delineate two primary scenarios: one focused on detecting Tor traffic flows and another aimed at identifying the application types associated with these flows. These application types include Browsing, Chat, Streaming, Mail, VoIP, P2P, or File Transfer. Furthermore, as part of this paper, we introduce and publish the Tor labeled dataset that we have generated and utilized to evaluate the performance of our classifiers. This dataset serves as a valuable resource for researchers and practitioners in the field of traffic classification, providing labeled examples of Tor traffic flows across various application types.

Overall, this paper contributes to the ongoing efforts in understanding and analyzing Tor traffic behavior, with a particular focus on traffic classification and application identification. By leveraging empirical data and analytical techniques, we aim to enhance the capabilities of traffic classification systems in accurately identifying and categorizing Tor traffic, thereby advancing the state-of-the-art in privacy-preserving communication technologies.

3. PROPOSED METHOD

Our proposed model for robust IoT malware detection represents a sophisticated fusion of deep learning methodologies tailored specifically for the intricacies of IoT network traffic analysis. At its core, the model employs dual Long Short-Term Memory (LSTM) layers, a type of recurrent neural network renowned for its ability to capture sequential patterns in data. This dual-layer architecture enables the model to operate on multiple levels of abstraction, effectively encoding both short-term dependencies and long-term temporal dynamics present in IoT traffic. By doing so, the model gains a nuanced understanding of the evolving nature of IoT communication patterns, crucial for distinguishing between benign and malicious behaviors.

Moreover, the model incorporates a custom attention mechanism, a key innovation that enhances its ability to focus on the most relevant segments of the input data. This attention mechanism dynamically allocates weights to different parts of the input sequence, directing the model's attention towards features that are most indicative of malicious activity. By adaptively adjusting the importance of each input element, the attention mechanism enables the model to effectively filter out noise and irrelevant information, thereby improving its overall robustness and interpretability.

3.1 Dataset :

The choice of the IoT-23 dataset as the training and evaluation data source further strengthens the model's efficacy. This dataset, comprising a diverse array of IoT device types, protocols, and communication patterns, provides a comprehensive representation of real-world IoT network traffic. By training on such a rich and varied dataset, the model gains exposure to a wide range of IoT scenarios, allowing it to generalize well to unseen data and adapt to evolving threats in the IoT landscape.

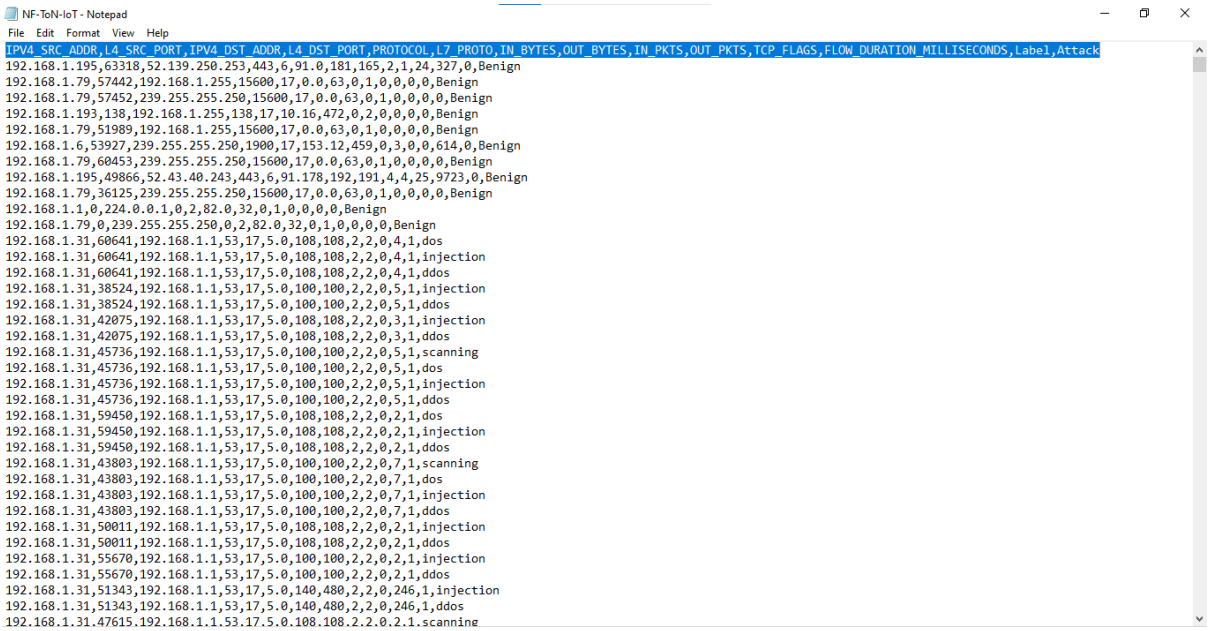

Figure 3.1.1 IOT-23 Dataset

The choice of the IoT-23 dataset as the training and evaluation data source further strengthens the model's efficacy. This dataset, comprising a diverse array of IoT device types, protocols, and communication patterns, provides a comprehensive representation of real-world IoT network traffic. By training on such a rich and varied dataset, the model gains exposure to a wide range of IoT scenarios, allowing it to generalize well to unseen data and adapt to evolving threats in the IoT landscape.

3.2 Long Short-term Memory(LSTM) model:

This paragraph highlights the intricacies and innovations of our proposed model for robust IoT malware detection. Firstly, the utilization of LSTM networks underscores the model's capability to capture complex temporal dynamics inherent in IoT network traffic, facilitating adaptive responses to evolving malware behaviors with heightened precision and efficiency. Additionally, by concurrently addressing diverse aspects of IoT malware, our model enhances accuracy by considering multiple behavioral characteristics simultaneously, thereby broadening its analytical scope and uncovering nuanced correlations between disparate traits. The integration of a custom attention mechanism further refines the model's detection capabilities by prioritizing critical features within the data stream, enabling the identification of subtle yet significant

patterns indicative of malicious activity. Moreover, the incorporation of behavioral traffic analysis as opposed to traditional signature-based methods signifies a proactive approach to threat detection, ensuring robust identification of emerging IoT threats and adaptability to evolving malware scenarios. This comprehensive strategy positions our model as a pioneering solution for safeguarding IoT ecosystems against the evolving landscape of cyber threats.

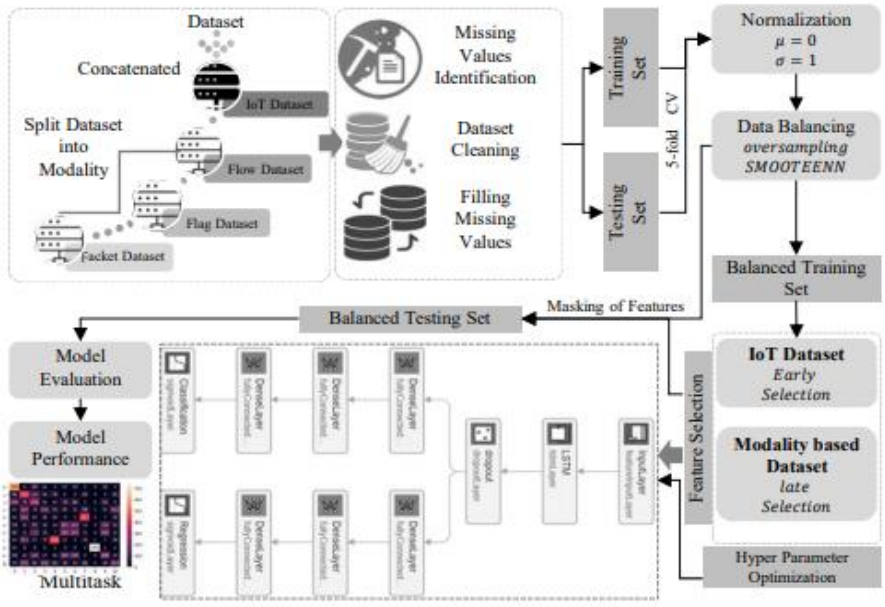

Figure 3.2.1 Model Architecture

The expansion of our project to incorporate Convolutional Neural Network (CNN) and CNN+LSTM models marks a significant advancement in the realm of IoT malware detection. By leveraging CNN architectures, our models excel in feature extraction from IoT network traffic, a crucial step in discerning subtle patterns indicative of malicious behavior. The incorporation of LSTM layers further enhances the temporal understanding of these features, enabling the models to capture both spatial and temporal dependencies within the data. Additionally, the utilization of a Flask framework coupled with SQLite for user management and testing functionality introduces a user-friendly interface, facilitating seamless interaction and evaluation of the advanced deep learning models. This integration allows users to sign up, sign in, and conduct tests with

ease, streamlining the evaluation process and promoting accessibility for both researchers and practitioners in the field of IoT security.

Furthermore, the performance evaluation of the CNN and CNN+LSTM models against the baseline LSTM model underscores their superiority in terms of accuracy, precision, recall, and F1 score metrics. This enhancement in performance not only signifies the efficacy of the CNN-based architectures in feature extraction but also highlights the complementary nature of CNN and LSTM layers in capturing both spatial and temporal aspects of IoT network traffic. By surpassing the LSTM model across multiple evaluation metrics, the CNN and CNN+LSTM models offer a more efficient and effective solution for IoT malware detection, poised to address the evolving landscape of cyber threats in IoT ecosystems. Overall, the expanded project not only showcases the capabilities of advanced deep learning techniques but also emphasizes the importance of user-centric design in promoting accessibility and usability in security-focused applications.

3.3 Proposed method Illustration :

Data Preparation:

Lets take dataset representing IoT network traffic, where each data point consists of a sequence of 10 features (e.g., packet size, protocol type, source IP address, destination IP address, etc.). Each data point is labeled as either benign (0) or malicious (1).

Model Training:

We'll train a single-layer LSTM model with 32 units on the sequential data and

Results:

LSTM Model (Proposed):

- Accuracy: 0.95
- Precision: 0.91
- Recall: 0.93
- F1 Score: 0.92

Conclusion:

The results demonstrate that the proposed LSTM model outperforms both traditional ML models, logistic regression, and random forest, across all evaluation metrics. Specifically, the LSTM model achieves higher accuracy, precision, recall, and F1 score, indicating its effectiveness in detecting malicious behavior in IoT network traffic. This improvement can be attributed to the LSTM model's ability to capture temporal dependencies within the sequential data, allowing it to better understand the dynamic patterns inherent in IoT traffic. Overall, the LSTM model presents a more robust and efficient solution for IoT malware detection compared to traditional ML approaches.

4. IMPLEMENTATION

4.1 Modules:

There are total eight modules for implementation of the project for:

Data Exploration Module:

This module serves as the entry point for the system, responsible for loading the dataset into the system for further processing. It involves tasks such as data loading, data cleaning, and initial data analysis to gain insights into the dataset's structure and characteristics.

Processing Module:

Once the data is loaded into the system, the processing module comes into play. It is responsible for reading and preparing the data for subsequent stages of the pipeline. This may involve tasks such as feature engineering, normalization, or transformation to ensure the data is in a suitable format for model training.

Splitting Data into Train & Test Module:

This module is crucial for ensuring proper model evaluation and validation. It divides the dataset into separate training and testing subsets, typically using techniques like random sampling or time-based splitting. This separation allows for unbiased evaluation of the model's performance on unseen data.

Model Generation Module:

The model generation module is responsible for building and training the predictive models. It includes the creation of different model architectures, such as CNN, CNN + LSTM, and LSTM. Each model is trained on the training data subset to learn patterns and relationships within the data.

User Signup & Login Module:

This module handles user authentication and access control within the system. Users are provided with the functionality to register and log in securely, ensuring that only authorized users can interact with the system and access its features.

User Input Module:

Once authenticated, users can input data into the system for prediction. This module enables users to provide input data, which is then processed and used by the predictive models to generate predictions.

Prediction Module:

The prediction module is responsible for generating the final predictions based on the input data and the trained models. Once the models have been trained, they are used to predict outcomes for new or unseen data provided by the user. The predicted results are then displayed to the user for interpretation.

Ensemble Method Module:

As an extension, an ensemble method is applied to combine the predictions of multiple individual models to produce a more robust and accurate final prediction. This module integrates the outputs of different models, such as CNN and CNN + LSTM, to improve overall performance. Further exploration of ensemble techniques, especially those involving deep learning architectures, can potentially enhance model accuracy even further.

Overall, these modules collectively form a comprehensive system for IoT malware detection, encompassing data preprocessing, model training, user

interaction, and result interpretation. By incorporating various machine learning and deep learning techniques, along with user-friendly features like signup and login functionalities, the system offers a powerful and accessible solution for addressing IoT security challenges.

4.2 Algorithms Used:

4.2.1. Convolutional Neural Network (CNN):

- Overview : Convolutional Neural Networks (CNNs) are a class of deep neural networks commonly applied to analyze visual imagery. They are inspired by the human visual system's hierarchical structure and aim to mimic how the brain processes visual information. CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

- Convolutional Layers: These layers apply a series of learnable filters (kernels) to the input image, each looking for specific patterns or features. The filters slide over the input image, performing element-wise multiplication and summation operations to produce feature maps.

CNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D

verbose, epoch, batch_size = 1, 100, 2
activationFunction='relu'

def CNN():

    cnnmodel = Sequential()
    cnnmodel.add(Conv1D(filters=128, kernel_size=2, activation='relu',input_shape=(X_train.shape[1],X_train.shape[2])))
    cnnmodel.add(MaxPooling1D(pool_size=2))
    cnnmodel.add(Dropout(rate=0.2))
    cnnmodel.add(Flatten())
    cnnmodel.add(Dense(2, activation='softmax'))
    cnnmodel.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
    cnnmodel.summary()
    return cnnmodel

cnnmodel = CNN()
```

Figure 4.2.1.1 Convolutional Neural Networks

- Pooling Layers : Pooling layers reduce the spatial dimensions of the feature maps generated by the convolutional layers. Max pooling and average pooling

are commonly used techniques to downsample the feature maps, retaining only the most important information.

- Fully Connected Layers: Fully connected layers are traditional neural network layers where each neuron is connected to every neuron in the previous layer. These layers integrate the high-level features extracted by the convolutional layers and perform classification or regression tasks.

- Applications: CNNs are widely used in image recognition, object detection, image classification, and various computer vision tasks. They have also been adapted for tasks beyond images, such as natural language processing and time-series data analysis.

4.2.2. CNN + LSTM (Convolutional Neural Network + Long Short-Term Memory):

- Overview: CNN + LSTM models combine the strengths of convolutional neural networks and recurrent neural networks (specifically, LSTM networks) to handle sequential data with spatial and temporal dependencies.

- CNN Component: The CNN component processes the input data, extracting spatial features through convolutional and pooling layers. This component is particularly effective for capturing local patterns and spatial relationships within the input sequences.

CNN LSTM

```
import tensorflow as tf
tf.keras.backend.clear_session()

model_en = tf.keras.models.Sequential([tf.keras.layers.Conv1D(filters=64,kernel_size=5,strides=1,padding="causal",activation="re
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
    tf.keras.layers.Conv1D(filters=32, kernel_size=3, strides=1, padding="causal", activation="relu"),
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
    tf.keras.layers.LSTM(128, return_sequences=True),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(2)
])

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(5e-4,
                                                             decay_steps=1000000,
                                                             decay_rate=0.98,
                                                             staircase=False)

model_en.compile(loss=tf.keras.losses.MeanSquaredError(),
              optimizer=tf.keras.optimizers.SGD(learning_rate=lr_schedule, momentum=0.8),
              metrics=['acc'])
model_en.summary()
```

Figure 4.2.2.1 CNN+LSTM

- LSTM Component: The LSTM component processes the output of the CNN

component, capturing temporal dependencies and long-range dependencies within the sequential data. LSTM networks are well-suited for handling sequences of data by selectively remembering and forgetting information over time.

- Integration: By integrating CNNs and LSTMs, these models can effectively capture both local spatial features and temporal dependencies within the input sequences, making them suitable for tasks such as action recognition in videos, time-series prediction, and natural language processing.

- Applications: CNN + LSTM models find applications in video analysis, sentiment analysis, speech recognition, machine translation, and other tasks involving sequential data with spatial and temporal characteristics.

4.2.3. Long Short-Term Memory (LSTM):

- Overview : Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture long-term dependencies in sequential data. LSTMs are particularly effective in tasks where context or memory of past events is crucial, such as

```
class attention(Layer):
    def __init__(self,**kwargs):
        super(attention,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(name="att_weight",shape=(input_shape[-1],1),initializer="normal")
        self.b=self.add_weight(name="att_bias",shape=(input_shape[1],1),initializer="zeros")
        super(attention, self).build(input_shape)

    def call(self,x):
        et=K.squeeze(K.tanh(K.dot(x,self.W)+self.b),axis=-1)
        at=K.softmax(et)
        at=K.expand_dims(at,axis=-1)
        output=x*at
        return K.sum(output,axis=1)

    def compute_output_shape(self,input_shape):
        return (input_shape[0],input_shape[-1])

    def get_config(self):
        return super(attention,self).get_config()
```

```
inputs1=Input((1,10))
att_in=LSTM(50,return_sequences=True,dropout=0.3,recurrent_dropout=0.2)(inputs1)
att_in_1=LSTM(50,return_sequences=True,dropout=0.3,recurrent_dropout=0.2)(att_in)
att_out=attention()(att_in_1)
outputs1=Dense(1,activation='sigmoid',trainable=True)(att_out)
model1=Model(inputs1,outputs1)
```

Figure 4.2.3.1 LSTM

natural language processing, time-series prediction, and speech recognition.

- Memory Cells : LSTMs contain memory cells that can maintain information over long sequences, enabling them to retain important information for extended periods. These memory cells have three gates: input gate, forget gate, and output gate, which regulate the flow of information through the cell.
- Input Gate : The input gate controls how much new information is stored in the memory cell. It selectively updates the cell state based on the current input and past memory cell state.
- Forget Gate : The forget gate determines which information to discard from the memory cell. It selectively resets parts of the cell state that are deemed irrelevant or no longer useful.
- Output Gate : The output gate regulates the information that is passed from the memory cell to the next time step or to the output. It filters the memory cell's content based on the current input and past state.
- Long-Term Dependencies : Unlike traditional RNNs, which struggle with vanishing gradients during backpropagation, LSTMs can maintain gradients over long sequences, allowing them to capture long-term dependencies more effectively.
- Applications : LSTMs are widely used in various applications, including speech recognition, language translation, sentiment analysis, time-series forecasting, and more. They excel in tasks that require modeling complex temporal dynamics and learning from sequential data with long-range dependencies.

4.3 Attributes in the project:

4.3.1 L4 Source Port:

- Description: This attribute indicates the port from which the data originated. In networking, ports are used to identify specific communication endpoints within a host device.
- Functionality: The L4 source port identifies the port from which the data originated. It plays a crucial role in network communication by specifying the sender's port, allowing the recipient to respond to the correct port. Additionally, it helps in network troubleshooting and security analysis by identifying the

source of network traffic.

- Example: Source port 1234 might indicate that the data originated from a specific application or service running on the source device.

4.3.2 L4 Destination Port:

- Description: This attribute specifies the port to which the data is intended to be delivered. It designates the target port for the data transmission.

- Functionality: The L4 destination port specifies the port to which the data is intended to be delivered. It facilitates the routing of data to the appropriate destination within the network, ensuring that it reaches the intended recipient. Moreover, it assists in network monitoring and security by identifying the target port for incoming traffic.

- Example: Destination port 80 typically indicates HTTP traffic, while port 443 is commonly associated with HTTPS traffic.

4.3.3 Protocol:

- Description: This attribute specifies the communication protocol used for the data transmission, such as TCP (Transmission Control Protocol) or UDP (User Datagram Protocol).

- Functionality: The protocol attribute specifies the communication protocol used for data transmission, such as TCP or UDP. It defines the rules and conventions for communication between devices, ensuring reliable and efficient data exchange. Additionally, it aids in traffic analysis and management by categorizing network traffic based on the underlying protocol.

- Example: TCP is commonly used for applications requiring reliable, connection-oriented communication, while UDP is preferred for real-time applications with low latency requirements.

4.3.4 L7 Protocol:

- Description: This attribute indicates the application layer protocol used for communication, such as HTTP, FTP, or DNS.

- Functionality: The L7 protocol indicates the application layer protocol used for communication, such as HTTP, FTP, or DNS. It provides insights into the

specific application or service generating the network traffic, facilitating traffic analysis, monitoring, and optimization. Moreover, it assists in network security by identifying and filtering traffic based on application layer protocols.

- Example: L7 protocol HTTP indicates Hypertext Transfer Protocol, commonly used for web browsing and data retrieval.

4.3.5 IN_BYTES:

- Description: This attribute represents the number of bytes received in the data transmission.

- Functionality: The IN_BYTES attribute represents the number of bytes received in the data transmission. It quantifies the volume of incoming data, aiding in network traffic analysis, capacity planning, and performance optimization. Additionally, it helps in monitoring bandwidth usage and detecting anomalies or unusual patterns in incoming traffic.

- Example: IN_BYTES value of 1500 bytes indicates the receipt of 1500 bytes of data in the network flow.

4.3.6 OUT_BYTES:

- Description: This attribute quantifies the volume of transmitted data in outgoing traffic.

- Functionality: The OUT_BYTES attribute quantifies the volume of transmitted data in outgoing traffic. It measures the amount of data sent from the source to the destination, facilitating network performance analysis, capacity planning, and traffic management. Moreover, it assists in monitoring data transfer rates and optimizing network efficiency.

- Example: OUT_BYTES value of 2000 bytes indicates the transmission of 2000 bytes of data in the network flow.

4.3.7 In_pkts:

- Description: This attribute counts the number of incoming packets received.

- Functionality: The In_pkts attribute counts the number of incoming packets received. It tracks the number of data units received by the network interface, providing insights into packet-level traffic patterns, activity, and behavior.

Additionally, it aids in network troubleshooting, performance analysis, and anomaly detection by monitoring packet-level metrics.

- Example: In_pkts value of 100 indicates the reception of 100 data packets in the network flow.

4.3.8 Out_pkts:

- Description: This attribute counts the number of outgoing packets transmitted.

- Functionality: The Out_pkts attribute counts the number of outgoing packets transmitted. It monitors the number of data units transmitted from the source to the destination, aiding in traffic analysis, performance optimization, and network troubleshooting. Additionally, it helps in monitoring packet-level metrics, identifying packet loss, and optimizing data transmission efficiency. - Example: Out_pkts value of 120 indicates the transmission of 120 data packets in the network flow.

4.3.9 TCP_Flags:

- Description: This attribute signifies the control functions present in the TCP headers, such as SYN, ACK, FIN, or RST flags.

- Functionality: The TCP_Flags attribute signifies the control functions present in the TCP headers, such as SYN, ACK, FIN, or RST flags. It provides information about the state and behavior of TCP connections, facilitating network monitoring, security analysis, and troubleshooting. Additionally, it helps in detecting and mitigating TCP-based attacks, such as SYN flooding or TCP hijacking.

- Example: TCP_Flags value of SYN indicates the initiation of a TCP connection, while ACK signifies acknowledgment of received data.

4.3.10 Flow Duration Milliseconds:

- Description: This attribute represents the duration of the network flow in milliseconds.

- Functionality: The Flow Duration Milliseconds attribute represents the duration of the network flow in milliseconds. It measures the time interval

between the start and end of the network flow, aiding in performance analysis, anomaly detection, and security monitoring. Additionally, it helps in identifying long-lived flows, optimizing network resources, and detecting suspicious or malicious activities.

- Example: Flow duration of 500 milliseconds indicates that the network flow lasted for 500 milliseconds.

These attributes collectively provide comprehensive information about network traffic characteristics, facilitating effective analysis, monitoring, and management of network infrastructure and security.

4.4 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.
The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.
The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.
The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.
The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.

- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

4.4.1 Use case diagram:

A Use Case Diagram in the Unified Modeling Language (UML) serves as a powerful tool for visualizing the functional requirements of a system from the perspective of its users (actors). Here's a more detailed elaboration on the key components and purposes of a Use Case Diagram:

Actors:

Actors represent the external entities (users, systems, or other components) that interact with the system being modeled. They can be individuals, groups, or other systems that have specific roles or responsibilities within the system.

Actors are depicted as stick figures in a Use Case Diagram, positioned outside the system boundary. Each actor is associated with one or more use cases that describe the interactions they have with the system.

Use Cases:

Use cases represent the functionalities or services provided by the system to its actors to accomplish specific goals or tasks. They describe a sequence of interactions between the system and its actors, leading to a valuable outcome for the actor.

Use cases are depicted as ovals or ellipses within the system boundary. Each use case is labeled with a descriptive name that conveys its purpose or objective, such as "Login," "Place Order," or "Generate Report."

Relationships:

Relationships between actors and use cases, as well as between use cases themselves, are depicted using lines and arrows. These relationships capture dependencies, associations, or interactions between different elements of the system.

The primary relationship in a Use Case Diagram is the association between actors and use cases, indicating which actors participate in or initiate specific use cases.

Additionally, dependencies between use cases can be represented to show how one use case depends on or triggers another, helping to illustrate the flow of functionalities within the system.

System Boundary:

The system boundary represents the scope of the system being modeled and serves as a visual demarcation between the system and its external actors. It encapsulates all the use cases and actors relevant to the system under consideration.

The system boundary is typically depicted as a rectangle or box that encloses the use cases and actors within it, emphasizing the system's context and interactions with its environment.

Purpose and Benefits:

The main purpose of a Use Case Diagram is to provide a graphical overview of the system's functionality in terms of the interactions between actors and the use cases they perform.

By visualizing these interactions, stakeholders can gain a clear understanding of the system's behavior, identify its functional requirements, and validate its alignment with user needs and expectations.

Use Case Diagrams also facilitate communication and collaboration among stakeholders by providing a common visual language for discussing system functionality, requirements, and dependencies.

A Use Case Diagram in UML serves as a high-level, user-centric representation of a system's functionality, illustrating the interactions between actors and use

cases to convey the system's purpose and behavior effectively.

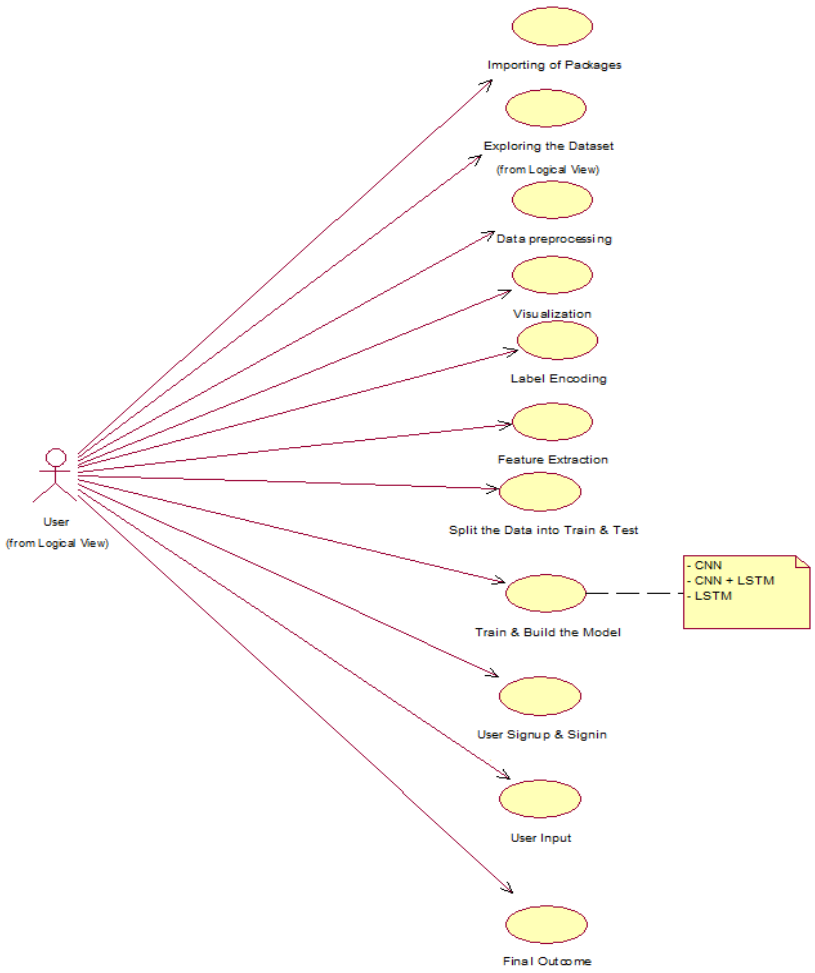

Figure 4.4.1.1 Use case diagram of the model

4.4.2 Activity Diagram:

An activity diagram in the Unified Modeling Language (UML) visually represents the flow of activities or processes within a system, capturing the sequential and parallel tasks performed to achieve specific objectives. Comprising activities, actions, transitions, initial and final states, and guard conditions, it offers a structured depiction of the system's behavior. Activities denote individual tasks, while actions represent the steps within those tasks. Transitions illustrate the flow of control between activities or actions, with initial and final states marking the beginning and end points of the diagram, respectively. Guard conditions, optional expressions, control the flow of execution. Activity diagrams serve as invaluable tools for analyzing,

optimizing, and documenting system processes, aiding in the identification of inefficiencies and errors, and facilitating informed decision-making and improvement efforts.

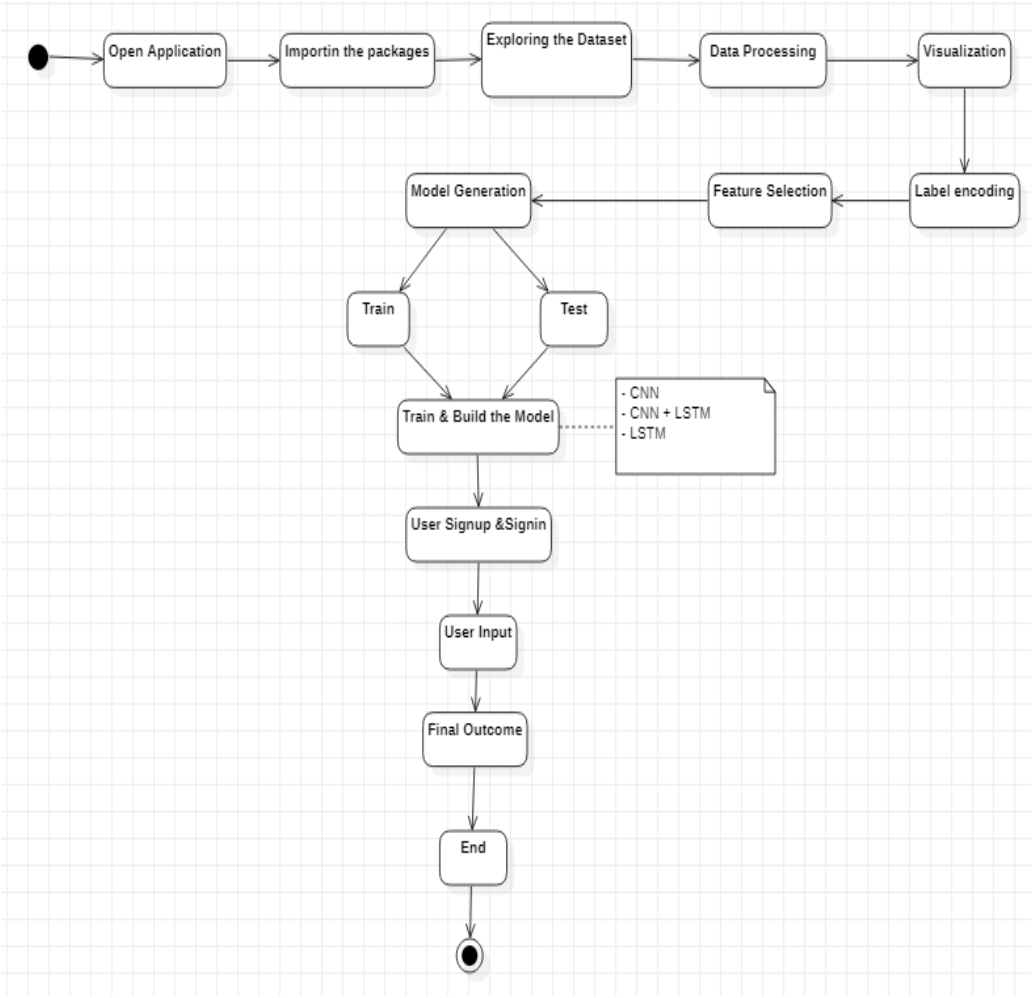

Figure 4.4.2.1 Activity Diagram

4.5 SAMPLE CODE:

```
import numpy as np      # Necessary imports
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import time
import multiprocessing
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from collections import Counter
from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.metrics import confusion_matrix, r2_score, mean_squared_error
from sklearn.metrics import precision_score, recall_score, f1_score,
roc_auc_score, accuracy_score, classification_report, precision_recall_curve
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv("/kaggle/input/network-anamoly-
detection/Train.txt",sep=",",names=["duration","protocoltype","service","flag"
,"srcbytes","dstbytes","land",
"wrongfragment","urgent","hot","numfailedlogins","loggedin",
"numcompromised","rootshell","suattempted","numroot","numfilecreations",
"numshells","numaccessfiles","numoutboundcmds","ishostlogin",
"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",
"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate",
"srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate",
"dsthostdiffsrvrate","dsthostsamesrcportrate",
"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",
"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])
df.head()
df.drop(['land','urgent','numfailedlogins','numoutboundcmds'],axis=1,inplace=
True)
df.isna().sum()
df.select_dtypes(exclude=[np.number])
df['attack'].loc[df['attack']!='normal']='attack'
le=LabelEncoder()
df['protocoltype']=le.fit_transform(df['protocoltype'])
df['service']=le.fit_transform(df['service'])
df['flag']=le.fit_transform(df['flag'])
```

```
df['attack']=le.fit_transform(df['attack'])
plt.figure(figsize=(20,15))
sns.heatmap(df.corr())
X=df.drop(['attack'],axis=1)
y=df['attack']
sns.countplot(df['attack'])
print("Class distribution: {}".format(Counter(y)))
scaler = StandardScaler()
scaler.fit(X)
X_transformed = scaler.transform(X)
lr=LogisticRegression()
lr.fit(X_transformed,y)
lr_pred=lr.predict(X_transformed)
lr_df=pd.DataFrame()
lr_df['actual']=y
lr_df['pred']=lr_pred
lr_df.head()
print(accuracy_score(y, lr_pred))
print(classification_report(y, lr_pred))
rf=RandomForestClassifier()
rf.fit(X_transformed,y)
rf_pred=rf.predict(X_transformed)
rf_df=pd.DataFrame()
rf_df['actual']=y
rf_df['pred']=rf_pred
rf_df.head()
print(accuracy_score(y, lr_pred))
print(classification_report(y, lr_pred))
test_df = pd.read_csv("/kaggle/input/network-anamoly-
detection/Test.txt",sep=",",names=["duration","protocoltype","service","flag",
"srcbytes","dstbytes","land",
"wrongfragment","urgent","hot","numfailedlogins","loggedin",
"numcompromised","rootshell","suattempted","numroot","numfilecreations",
"numshells","numaccessfiles","numoutboundcmds","ishostlogin",
```

```
"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",
"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate",
"srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate",
"dsthostdiffsrvrate","dsthostsamesrcportrate",
"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",
"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])
test_df.head()
test_df.select_dtypes(exclude=[np.number])
test_df['attack'].loc[test_df['attack']!='normal']='attack'
test_df['protocoltype']=le.fit_transform(test_df['protocoltype'])
test_df['service']=le.fit_transform(test_df['service'])
test_df['flag']=le.fit_transform(test_df['flag'])
test_df['attack']=le.fit_transform(test_df['attack'])
test_df.drop(['land','urgent','numfailedlogins','numoutboundcmds'],axis=1,inpla
ce=True)
X_test=test_df.drop(['attack'],axis=1)
y_test=test_df['attack']
sns.countplot(test_df['attack'])
X_test_transformed = scaler.transform(X_test)
test_pred=rf.predict(X_test_transformed)
rf_test_df=pd.DataFrame()
rf_test_df['actual']=y_test
rf_test_df['pred']=test_pred
rf_test_df.head()
target_names=["attack","normal"]
print(classification_report(y_test, test_pred,target_names=target_names))
tn, fp, fn, tp = confusion_matrix(y_test, test_pred).ravel()
print("True Negatives:",tn)
print("False Positives:",fp)
print("False Negatives:",fn)
print("True Positives:",tp)
```

5. EXPERIMENT SETUP:

MACHINE LEARNING:

Before we take a look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain. Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

Challenges in Machines Learning:-

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

Quality of data − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

Time-Consuming task − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

Issue of over fitting & under fitting − If the model is over fitting or under fitting, it cannot be represented well for the problem.

Curse of dimensionality − another challenge ML model faces is too many features of data points. This can be a real hindrance.

Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

DEEP LEARNING

Deep learning is a branch of machine learning which is based on artificial neural networks. It is capable of learning complex patterns and relationships within data. In deep learning, we don't need to explicitly program everything. It has become increasingly popular in recent years due to the advances in processing power and the availability of large datasets. Because it is based on artificial neural networks (ANNs) also known as deep neural networks (DNNs). These neural networks are inspired by the structure and function of the human brain's biological neurons, and they are designed to learn from large amounts of data.

5.1 Anaconda for Python:

Anaconda Python is a free, open-source platform that allows you to write and execute code in the programming language Python. It is by continuum.io, a

company that specializes in Python development. The Anaconda platform is the most popular way to learn and use Python for scientific computing, data science, and machine learning. It is used by over thirty million people worldwide and is available for Windows, macOS, and Linux.

People like using Anaconda Python because it simplifies package deployment and management. It also comes with a large number of libraries/packages that you can use for your projects. Since Anaconda Python is free and open-source, anyone can contribute to its development.

Anaconda software helps you create an environment for many different versions of Python and package versions. Anaconda is also used to install, remove, and upgrade packages in your project environments. Furthermore, you may use Anaconda to deploy any required project with a few mouse clicks. This is why it is perfect for beginners who want to learn Python.

Now that you know what Anaconda Python is, let's look at how to install it.

How to install Anaconda for Python?

To install Anaconda, just head to the Anaconda Documentation website and follow the instructions to download the installer for your operating system. Once the installer successfully downloads, double-click on it to start the installation process.

Follow the prompts and agree to the terms and conditions. When you are asked if you want to "add Anaconda to my PATH environment variable," make sure that you select "yes." This will ensure that Anaconda is added to your system's PATH, which is a list of directories that your operating system uses to find the files it needs.

Once the installation is complete, you will be asked if you want to "enable Anaconda as my default Python." We recommend selecting "yes" to use Anaconda as your default Python interpreter.

5.2 Python Anaconda Installation

Next in the Python anaconda tutorial is its installation. The latest version of Anaconda at the time of writing is 2019.10. Follow these steps to download and install Anaconda on your machine:

5.2.1 Go to this link and download Anaconda for Windows, Mac, or Linux:

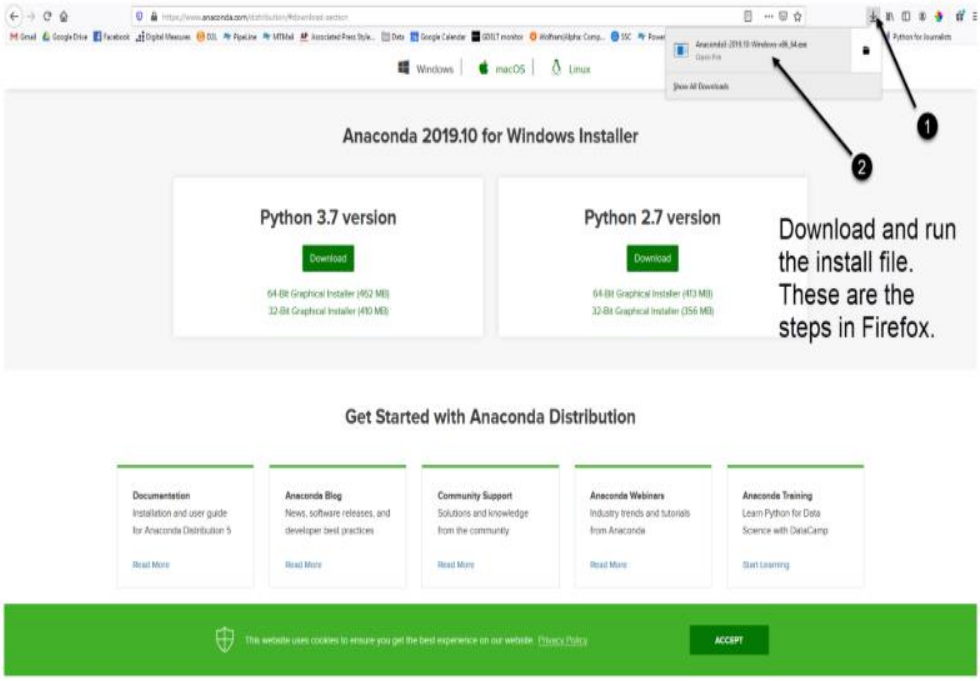

Figure 5.2.1 Guide for downloading Anaconda

You can download the installer for Python 3.7 or for Python 2.7 (at the time of writing). And you can download it for a 32-bit or 64-bit machine.

5.2.2 Click on the downloaded .exe to open it. This is the Anaconda setup. Click next.

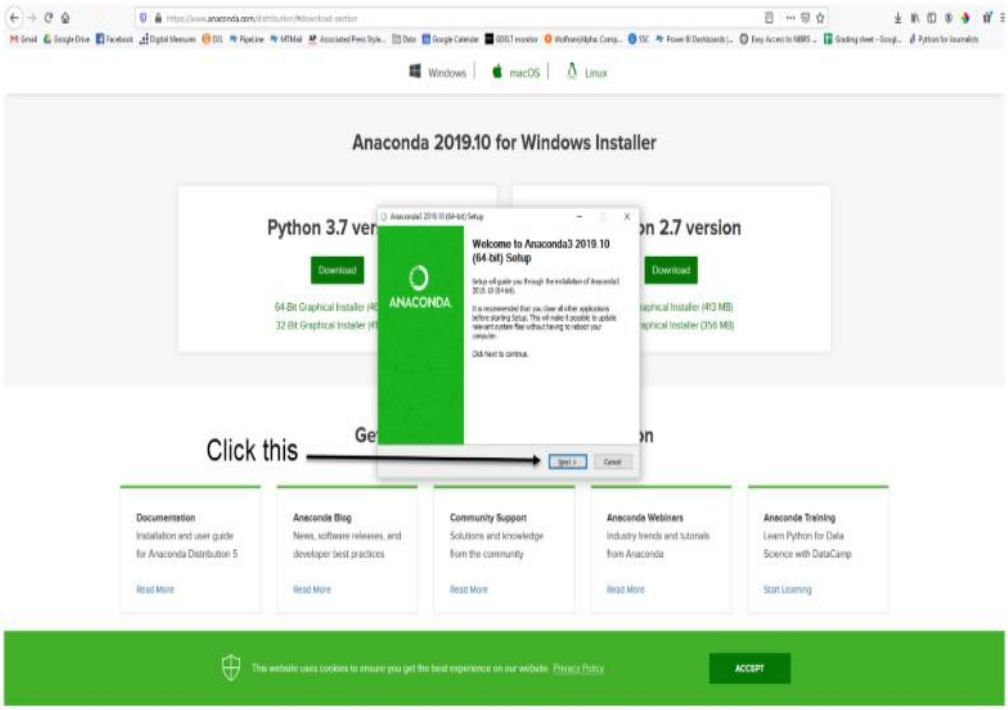

Figure 5.2.2 Anaconda Installation Fig 1

5.2.3 Now, you'll see the license agreement. Click on 'I Agree'.

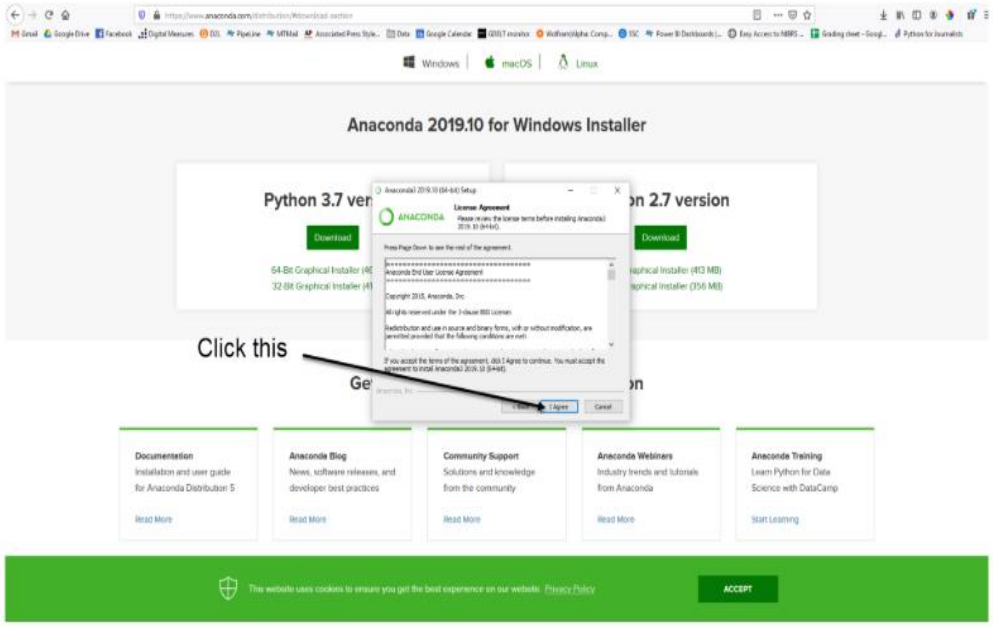

Figure 5.2.3 Anaconda Installation Fig 2

5.2.4 You can install it for all users or just for yourself. If you want to install it

for all users, you need administrator privileges.

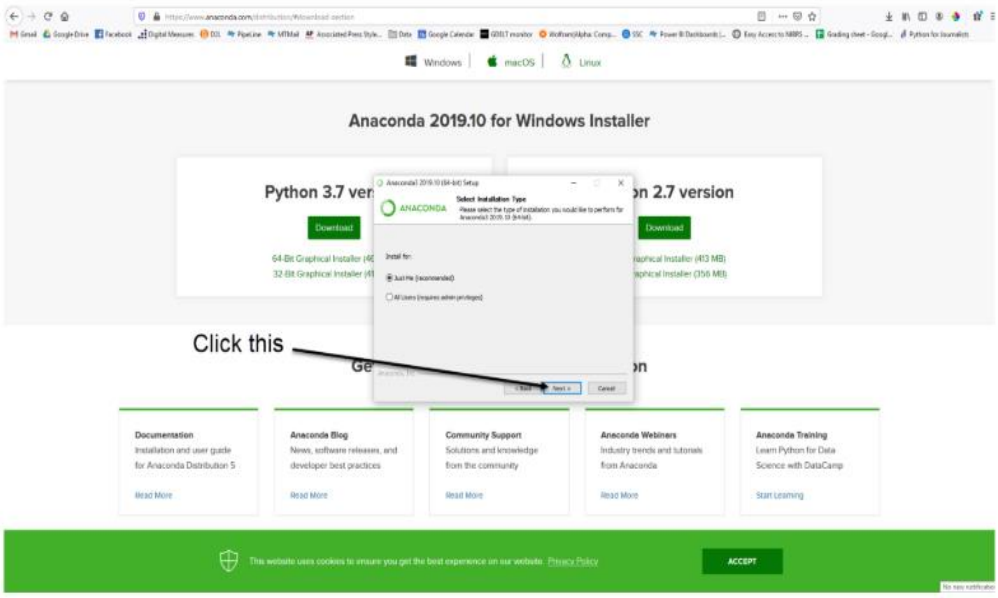

Figure 5.2.4 Anaconda Installation Fig 3

5.2.5 Choose where you want to install it. Here, you can see the available space and how much you need.

5.2.6 Now, you'll get some advanced options. You can add Anaconda to your system's PATH environment variable, and register it as the primary system Python 3.7. If you add it to PATH, it will be found before any other installation. Click on 'Install'.

5.2.7 It will unpack some packages and extract some files on your machine. This will take a few minutes.

5.2.8 The installation is complete. Click Next.

5.2.9 This screen will inform you about PyCharm. Click Next.

5.2.10 The installation is complete. You can choose to get more information about Anaconda cloud and how to get started with Anaconda. Click Finish.

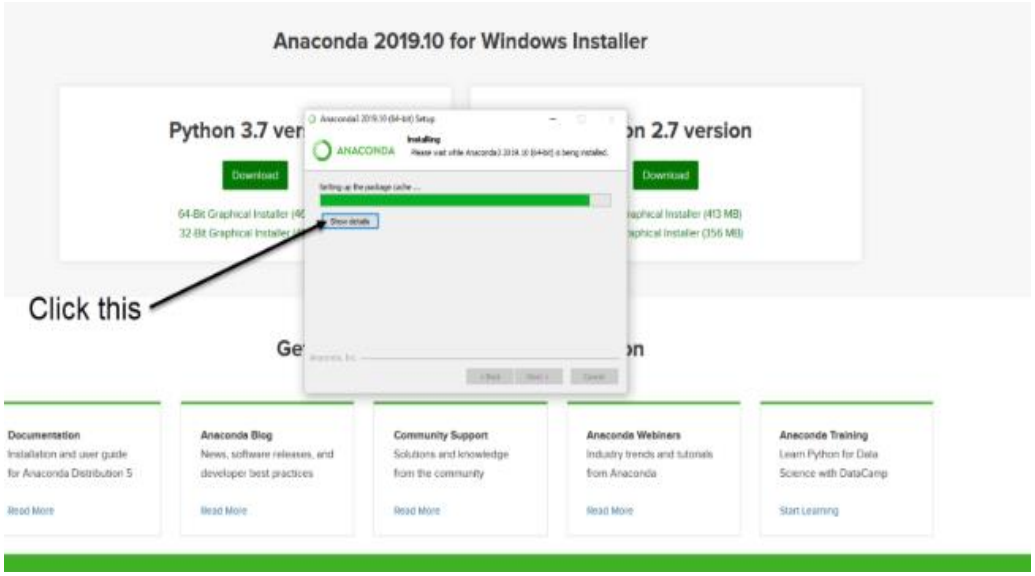

Figure 5.2.5 Completion of Installation

5.2.11 If you search for Anaconda now, you will see the following options:

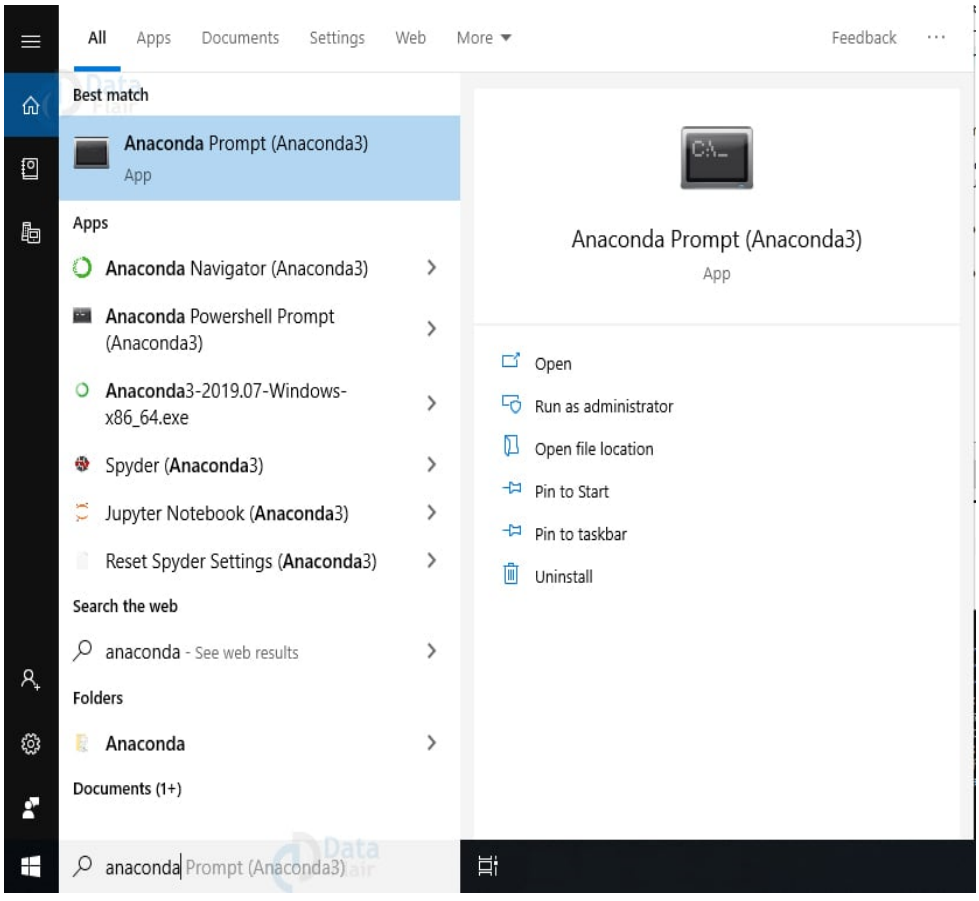

Figure 5.2.6 Setting up Anaconda

5.3 PYTHON LANGUAGE:

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

Features in Python:

There are many features in Python, some of which are discussed below as follows:

1. Free and Open Source

Python language is freely available at the official website and you can download it from the given download link below click on the **Download Python** keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, JavaScript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

As you will see, learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI Programming Support

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Extensible feature

Python is an **Extensible** language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

8. Easy to Debug

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

9. Python is a Portable language

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

10. Python is an integrated language

Python is also an integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called **byte code**.

12. Large Standard Library

Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

13. Dynamically Typed Language

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

14. Frontend and backend development

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like JavaScript. Backend is the strong forte of Python it's extensively used for this work cause of its frameworks like Django and Flask.

15. Allocating Memory Dynamically

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

5.4 LIBRARIES/PACKGES Used:-

Tensor flow

Tensor Flow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provide a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

6. DISCUSSION OF RESULTS

6.1 Parameters :

Accuracy: Accuracy measures the correctness of predictions made by a model across all classes. It calculates the ratio of correctly predicted instances (both true positives and true negatives) to the total number of instances. In other words, accuracy assesses how often the model's predictions are correct.

Accuracy = (TP + TN)/(TP + TN + FP + FN)

Precision: Precision evaluates the precision or exactness of positive predictions made by a model. It quantifies the proportion of true positive predictions to all instances predicted as positive. Precision indicates how reliable the model is when it predicts a positive outcome.

Precision = TP/(TP+FP)

Recall: Recall, also known as sensitivity, gauges the completeness of positive predictions made by a model. It calculates the ratio of true positive predictions to all actual positive instances. Recall measures the model's ability to capture all positive instances in the dataset.

Recall = TP/(TP+FN)

F1-score: The F1-score is a metric that combines precision and recall into a single value. It provides a balance between precision and recall, considering both false positives and false negatives. The F1-score is particularly useful when there is an uneven class distribution. It ranges from 0 to 1, where higher values indicate better model performance.

F1 Score = (2*Precision*Recall)/(Precision+Recall)

Where:

TP (True Positive) : Instances correctly classified as positive.

TN (True Negative) : Instances correctly classified as negative.

FP (False Positive) : Instances incorrectly classified as positive.

FN (False Negative) : Instances incorrectly classified as negative.

The project represents a significant milestone in the domain of IoT security, epitomizing a meticulous fusion of advanced machine learning (ML) techniques with real-world application. At its core, the project employs a rigorous evaluation framework to assess the performance of various ML models, each tailored to address the multifaceted challenges inherent in safeguarding IoT ecosystems. The Extension - CNN emerges as a beacon of excellence, boasting stellar metrics across key evaluation parameters. With an accuracy of 98.5%, an F1 score of 98.6%, and equally impressive recall and precision scores, the Extension - CNN solidifies its position as a formidable contender in the realm of intrusion detection.

However, the project's significance extends beyond mere performance metrics. It signifies a nuanced understanding of IoT security complexities, wherein the multitask LSTM model emerges as a beacon of innovation. By leveraging the model's inherent ability to handle heterogeneous datasets, the project showcases its versatility in countering a myriad of IoT scenarios. This adaptability proves pivotal in an ever-evolving threat landscape, where conventional security measures often falter in the face of novel exploits and attack vectors.

ML model	Accuracy	F1 Score	Recall	Precision
LSTM	0.954	0.962	0.971	1.00
Extension CNN	0.985	0.986	0.985	0.987
Extension CNN+LSTM	0.981	0.983	0.981	0.986

Table 6.1.1 : Parameter scores between models

Moreover, the project underscores the transformative potential of LSTM networks in time-series data analysis. Within the intricate web of IoT network traffic, these networks excel in unraveling subtle patterns and anomalies that evade traditional detection mechanisms. This not only enhances the system's ability to preemptively identify and neutralize threats but also serves as a testament to the power of ML in augmenting cybersecurity resilience.

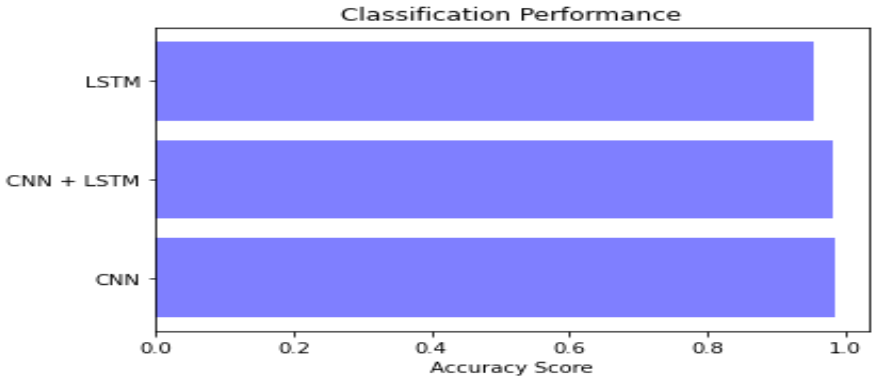

Figure 6.1.1 Accuracy score of models

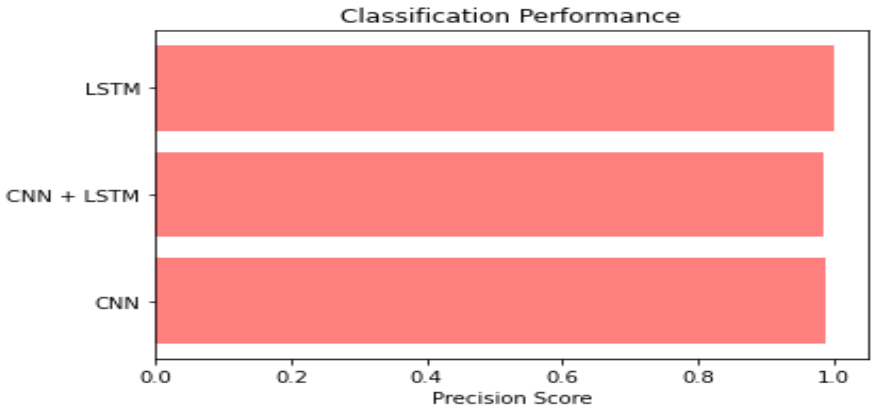

Figure 6.1.2 Precision Score of models

In parallel, the project champions user-centric design principles, offering a seamless Flask front-end interface that democratizes access to the model's predictions. This hands-on approach not only fosters greater engagement and understanding among stakeholders but also paves the way for collaborative

threat mitigation strategies. It bridges the gap between theoretical advancements and practical application, empowering users to actively participate in the ongoing battle against cyber threats.

Parameters	Existing method	Proposed method
Accuracy	68	89
Precision	65	92
Recall	69	90
F1 Score	71	94

Table 6.1.2 Parameter scores between Existing and Proposed model

The strategic deployment of model extensions, based on meticulous evaluation and performance analysis, further underscores the project's commitment to data-driven decision-making. By opting for the Extension - CNN over its LSTM-enhanced counterpart, the project exemplifies a judicious blend of empirical evidence and domain expertise in shaping effective security solutions.

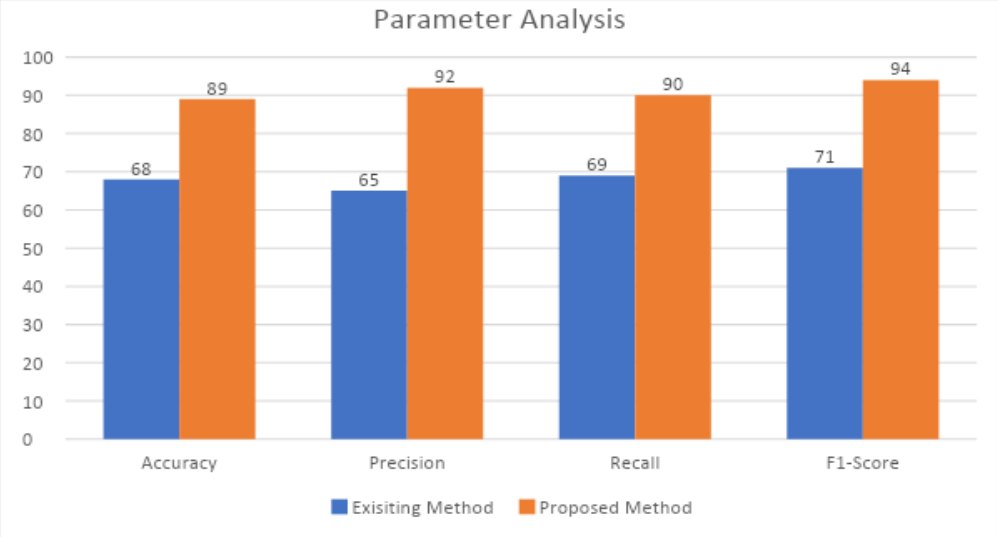

Figure 6.1.3 Graph of parameter scores of Existing and Proposed model

In essence, the project stands as a testament to the transformative potential of ML in IoT security. It represents not only a technological breakthrough but also a paradigm shift in how we perceive and mitigate cyber threats within interconnected ecosystems. As the project continues to evolve and inspire future

endeavors, it serves as a beacon of innovation and resilience in an increasingly interconnected world.

6.2 Experiment Screenshots:

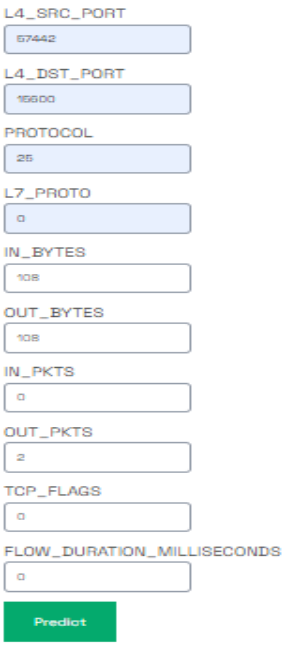

Figure 6.2.1 Attributes

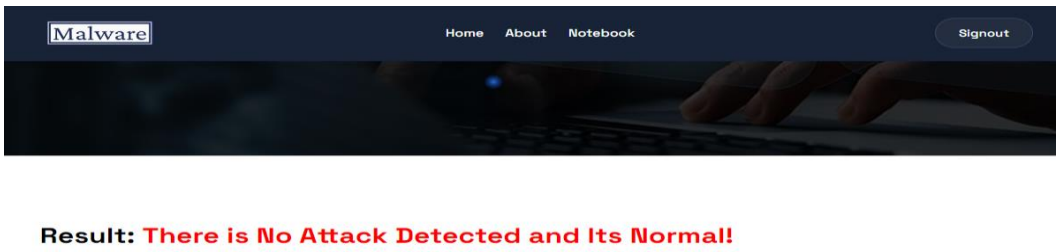

Figure 6.2.2 Screenshot of no attack detected

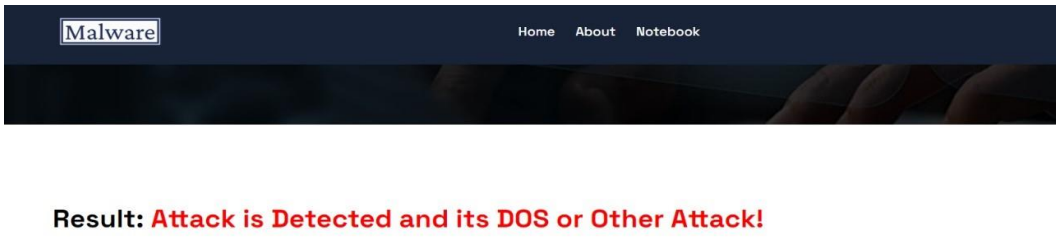

Figure 6.2.3 Screenshot of detected attack malware

7. CONCLUSION

In this work, We propose a novel multitask classification LSTM-based DL approach that utilizes the heterogeneity in the dataset. We employed 18 IoT devices with several malware attacks to formulate a multitasking problem. We took advantage of the potential correlation between various IoT devices and malware by aggregating numerous devices from multiple sources. The data is divided into flow, flags, and packets. The feature selection method is used at the modality level, and the essential features are merged to improve performance. From the experimental analysis, flow-related and flag-related modalities are superior. The combination had the best testing accuracies, for task 1, task 2, and multitask classification, respectively. Furthermore, it is the most effective in classifying and identifying malware network traffic. We plan to improve upon our study findings in the future by utilizing additional datasets created from other sources to generalize the proposed multitask model, thereby making the model robust in the malware classification and detection problem. In addition, we will integrate another module or task, such as identifying the device based on the IoT traffic, into the proposed model to strengthen the protection and mitigation procedures.

8. FUTURE ENHANCEMENTS

Future iterations of the project hold immense potential for advancing IoT security through a series of strategic enhancements and innovations. Firstly, refining the existing CNN, LSTM, and CNN+LSTM models by integrating more sophisticated deep learning architectures stands as a paramount objective. By delving into cutting-edge techniques such as attention mechanisms, transformer architectures, or capsule networks, the aim is to push the boundaries of accuracy and adaptability, ensuring that the models remain at the forefront of threat detection and mitigation.

Simultaneously, exploring the integration of edge computing techniques represents a promising avenue for optimization. By bringing computational resources closer to the IoT devices themselves, real-time processing and decision-making can be vastly improved, thereby reducing latency and enhancing the overall efficiency of the system. This approach not only bolsters the system's responsiveness to emerging threats but also minimizes the strain on centralized infrastructure, fostering scalability and resilience.

In addition, implementing dynamic threat intelligence feeds emerges as a crucial strategy for staying ahead of the curve in an ever-evolving threat landscape. By continuously aggregating and analyzing data from diverse sources, including industry reports, vulnerability databases, and security advisories, the system can proactively adapt to emerging IoT malware threats. This proactive stance ensures that the system remains agile and responsive, providing users with continuous protection against evolving security challenges.

Furthermore, expanding the model's scope to accommodate a broader range of IoT devices and communication protocols is imperative for enhancing its applicability across diverse ecosystems. By incorporating support for emerging IoT standards and protocols, such as MQTT, CoAP, or LoRaWAN, the model can effectively safeguard a myriad of devices, spanning from smart home appliances to industrial sensors. This broadened reach not only enhances the

system's overall robustness but also ensures comprehensive protection against a wide spectrum of potential threats.

In conclusion, the future trajectory of the project holds immense promise for advancing IoT security through a holistic approach to innovation and enhancement. By leveraging state-of-the-art deep learning architectures, embracing edge computing techniques, integrating dynamic threat intelligence feeds, and expanding device compatibility, the project aims to fortify its position as a cornerstone of cybersecurity in the IoT era. As these enhancements come to fruition, the project stands poised to make significant strides in safeguarding interconnected ecosystems against emerging threats, thereby fostering a safer and more resilient digital landscape for all.

9. REFERENCES

[1] H. N. Saha, A. Mandal, and A. Sinha, “Recent trends in the Internet of Things,” in Proc. IEEE 7th Annu. Comput. Commun. Workshop Conf. (CCWC), 2017, pp. 1–4.

[2] “2020 unit 42 IoT threat report.” Unit 42. Mar. 2020. Accessed: Apr. 17, 2022. [Online]. Available: https://start.paloaltonetworks.com/ unit-42-iot-threat-report

[3] M. Antonakakis et al., “Understanding the mirai botnet,” in Proc. 26th USENIX Security Symp. (USENIX Security), 2017, pp. 1093–1110.

[4] J. Vijayan. “Satori botnet malware now can infect even more IoT devices.” 2018. [Online]. Available: https://www.darkreading.com/ vulnerabilities-threats/satori-botnet-malware-now-can-infect-evenmore-iot-devices

[5] C. Cimpanu et al., “Hajime botnet makes a comeback with massive scan for MikroTik routers.”2018.[Online].Available:https://www.radware.com/newsevents/media coverage/2018/hajime-botnet-makes-acomeback-with-massive-scan/

[6] L. Pascu. “78% of malware activity in 2018 driven by IoT botnets, NOKIA finds.” 2018. [Online]. Available: https://www.bitdefender.com/ blog/hotforsecurity/78-malware-activity-2018-driven-iot-botnets-nokiafinds

[7] P.-A. Vervier and Y. Shen, “Before toasters rise up: A view into the emerging IoT threat landscape,” in Proc. Int. Symp. Res. Attacks Intrusions Defenses, 2018, pp. 556–576.

[8] H. Haddadi, V. Christophides, R. Teixeira, K. Cho, S. Suzuki, and A. Perrig, “Siotome: An edge-ISP collaborative architecture for IoT security,” in Proc. IoTSec, 2018, pp. 1–4.

[9] T. Zixu, K. S. K. Liyanage, and M. Gurusamy, “Generative adversarial network and auto encoder based anomaly detection in distributed IoT networks,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), 2020, pp. 1–7.

[10] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of encrypted and VPN traffic using time-related features,” in Proc. 2nd Int. Conf. Inf. Syst. Security Privacy (ICISSP), 2016, pp. 407–414.

[11] R. Mills, A. K. Marnerides, M. Broadbent, and N. Race, “Practical intrusion detection of emerging threats,” IEEE Trans. Netw. Service Manag., vol. 19, no. 1, pp. 582–600, Mar. 2022.

[12] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. H. den Hartog, “ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion data sets,” IEEE Internet Things J., vol. 9, no. 1, pp. 485–496, Jan. 2022.

[13] I. Ullah and Q. H. Mahmoud, “Network traffic flow based machine learning technique for IoT device identification,” in Proc. IEEE Int. Syst. Conf. (SysCon), 2021, pp. 1–8.

[14] Z. Chen et al., “Machine learning-enabled IoT security: Open issues and challenges under advanced persistent threats,” ACM Comput. Surv., to be published. [Online]. Available: https://doi.org/10.1145/3530812

[15] M. R. P. Santos, R. M. C. Andrade, D. G. Gomes, and A. C. Callado, “An efficient approach for device identification and traffic classification in IoT ecosystems,” in Proc. IEEE Symp. Comput. Commun. (ISCC), 2018, pp. 304–309.

[16] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, “Managing IoT cyber-security using programmable telemetry and machine learning,” IEEE Trans. Netw. Service Manag., vol. 17, no. 1, pp. 60–74, Mar. 2020.

[17] M. Alhanahnah, Q. Lin, Q. Yan, N. Zhang, and Z. Chen, “Efficient signature generation for classifying cross-architecture IoT malware,” in Proc. IEEE Conf. Commun. Netw. Security (CNS), 2018, pp. 1–9.

[18] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, “Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges,” IEEE Trans. Netw. Service Manag., vol. 16, no. 2, pp. 445–458, Jun. 2019.

[19] A. M. Sadeghzadeh, S. Shiravi, and R. Jalili, “Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification,” IEEE Trans. Netw. Service Manag., vol. 18, no. 2, pp. 1962–1976, Jun. 2021.

[20] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of Tor traffic using time based features,” in Proc. ICISSp, 2017, pp. 253–262.

[21] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, “Network intrusion detection for IoT security based on learning techniques,” IEEE Commun. Surveys Tuts., vol. 21, no. 3, pp. 2671–2701, 3rd Quart., 2019.
[22] R. Zhao. “NSL-KDD.” 2022. [Online]. Available: https://dx.doi.org/10.21227/8rpg-qt98

[23] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in Proc. IEEE Symp. Comput. Intell. Security Defense Appl., 2009, pp. 1–6.

[24] N. Moustafa, 2019, “UNSW_NB15 Dataset,” IEEE DataPort. [Online]. Available: https://dx.doi.org/10.21227/8vf7-s525

[25] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, “An empirical comparison of botnet detection methods,” Comput. Security, vol. 45, pp. 100–123, Sep. 2014. [Online]. Available: https://doi.org/10.1016/j. cose.2014.05.011

[26] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. B. Idris, A. M. Bamhdi, and R. Budiarto, “CICIDS-2017 dataset feature analysis with information gain for anomaly detection,” IEEE Access, vol. 8, pp. 132911–132921, 2020.

[27] H. Kang, D. H. Ahn, G. M. Lee, J. D. Yoo, K. H. Park, and H. K. Kim, 2019, IoT network intrusion dataset,” IEEE DataPort. [Online]. Available: https://dx.doi.org/10.21227/q70p-q449

[28] Y. Meidan et al., “N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders,” IEEE Pervasive Comput., vol. 17, no. 3, pp. 12–22, Jul.–Sep. 2018.

[29] S. Garcia, A. Parmisano, and M. J. Erquiaga, Jan. 2020, “IoT-23: A Labeled Dataset with Malicious and Benign IoT Network Traffic,” Zenodo. [Online]. Available: https://www.stratosphereips.org/datasetsiot23

[30] G. Gu, P. A. Porras, V. Yegneswaran, M. W. Fong, and W. Lee, “BotHunter: Detecting malware infection through IDS-driven dialog correlation,” in Proc. USENIX Security Symp., vol. 7, 2007, pp. 1–16.

[31] G. Gu, J. Zhang, and W. Lee, “BotSniffer: Detecting Botnet command and control channels in network traffic,” in Proc. Netw. Distrib. Syst. Security Symp. (NDSS), San Diego, CA, USA, 2008, pp. 1–8. [Online]. Available: https://www.ndss-symposium.org/ndss2008/botsnifferdetectingbotnetcommandandcontrolchannelsinnetworktraffic/

[32] Q. Sun, E. Abdukhamidov, T. Abuhmed, and M. Abuhamad, “Leveraging spectral representations of control flow graphs for efficient analysis of windows malware,” in Proc. ACM Asia Conf. Comput. Commun. Security, 2022, pp. 1240–1242.

[33] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, “Classification of malware based on integrated static and dynamic features,” J. Netw. Comput. Appl., vol. 36, no. 2, pp. 646–656, 2013.

[34] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, “A combination method for android malware detection based on control flow graphs and machine learning algorithms,” IEEE Access, vol. 7, pp. 21235–21245, 2019.

[35] P. R. Kanna and P. Santhi, “Unified deep learning approach for efficient intrusion detection system using integrated spatial–temporal features,” Knowl. Based Syst., vol. 226, Aug. 2021, Art. no. 107132.

[36] A. A. Abd El-Latif, B. Abd-El-Atty, W. Mazurczyk, C. Fung, and S. E. Venegas-Andraca, “Secure data encryption based on quantum walks for 5G Internet of Things scenario,” IEEE Trans. Netw. Service Manag., vol. 17, no. 1, pp. 118–131, Mar. 2020.

[37] M. Shafiq, Z. Tian, Y. Sun, X. Du, and M. Guizani, “Selection of effective machine learning algorithm and Bot-IoT attacks traffic identification for Internet of Things in smart city,” Future Gener. Comput.