IoT Threats Detection using Few-Shots Learning

BY

Chua Cheng Han

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: IoT Threats Detection using Few-Shots Learning

Academic Session: 202401

I CHUA CHENG HAN

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

(Author's signature)

Address:
86, Jalan Tempua 3
Bandar Puchong Jaya
47100, Puchong, Selangor

Date: 24/4/2024

Verified by,

(Supervisor's signature)

Aun Yichiet

Supervisor's name

Date: 26/04/2024

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 24/4/2024

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ***Chua Cheng Han*** (ID No: ***20ACB01761***) has completed this final year project/ dissertation/ thesis* entitled "*IoT Threats Detection using Few-Shots Learning*" under the supervision of Aun YiChiet (Supervisor) from the Department of Computer and Communication Technology, Faculty/Institute* of Information and Communication Technology .

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

(*Student Name*)

Chua Cheng Han

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**IoT Threats Detection using Few-Shots Learning**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : Chua Cheng Han

Date : 24/4/2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr. Aun Yichiet who has given me this bright opportunity to engage in the IoT threats detection project. It is my first step to establish a career in the cybersecurity field. A million thanks to you.

To a very special person in my life, Leong Yong Xin, for her patience, unconditional support, and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

Existing IoT threat detection methods lack robustness due to the diverse array of potential attack vectors. Currently, most methods are trained and tested using simulated datasets and do not perform well with unseen samples in real-world applications. In this project, we propose a novel few short learning leveraging Large Language Models (LLMs) to improve model robustness in IoT threat detection. Firstly, we develop two specialized LLM models: a text classification model based on DistilBERT and the few shots learning model using Sentence Transformer Fine-Tuning model (SetFit) framework. The DistilBERT threats detection model method performed well with an accuracy of 99.998% due to better semantics and contextual understanding as compared to existing flow statistical analysis. The few-shot learning model demonstrated remarkable performance with an accuracy of 0.89%, despite being trained on a limited amount of data. For unseen samples, we designed a few-shot retraining (FSR) methodology to adapt and learn new attack vectors across multiple variants using transfer learning. The experimental results showed a 90% improvement in accuracy on unseen threats when implemented in a real-world NIDS.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

DL	Deep Learning
FSL	Few-shot Learning
AI	Artificial Intelligence
IDS	Intrusion Detection System
IoT	Internet of Things
ML	Machine Learning
TL	Transfer Learning
BERT	Bidirectional Encoder Representations from Transformers
SetFit	Sentence Transformer Fine Tuning
Adam	Adaptive Moment Estimation
NLP	Natural Language Processing
RAM	Random Access Memory

Chapter 1 Introduction

1.1 Problem Statement and Motivation

In recent years, the proliferation of Internet of Things (IoT) devices has led to the unprecedented levels of connectivity and convenience to various aspects of our lives. These devices are seamlessly integrated into everyday environments, spanning from smart homes and industrial sectors to healthcare, transportation systems and smart cities. However, the rapid expansion of the IoT landscape has also opened the door to a new wave of security challenges. Traditional security approaches often fall short in addressing the unique and evolving threats posed by IoT environments, leaving critical systems vulnerable to breaches, data leaks, and unauthorized access.

One of the most pressing issues within the realm of IoT security is the efficient and accurate detection of threats. Conventional methods rely on predefined signatures or rules, which struggle to keep pace with the ever-changing tactics employed by malicious actors. Moreover, it is too costly to keep generating millions of datasets for new types of threats in IoT environments and labelling them accordingly to train an AI threat detection model.

The motivation behind this project stems from the urgent need to establish effective and adaptive security mechanisms for IoT ecosystems. Few-shot learning as a subset of machine learning which can train models to recognize IoT threats with only a limited amount of labeled data has emerged as a promising paradigm to provide an adaptive mechanism. The project takes on the challenge to perform IoT threat detection with few-shot learning and the implementation with Transformer-based model. The goal is to seek enhancement for the security mechanism of IoT environments while mitigating the costing issue to label extensive number of datasets.

1.2 Objectives

This project should achieve the following objectives:

I. **Generating unseen network threats to evaluate models' performance in classifying realistic, unseen realistic network threats.**

The subsequent objective is to assess the effectiveness of the developed models in accurately classifying unseen, real-world network threats. Through rigorous evaluation on diverse and authentic threat data, the objective aims to gauge the models' ability to generalize and adapt to different attack. By employing standard performance metrics, this evaluation provides crucial insights into the models' reliability, ultimately informing their suitability for practical deployment in enhancing IoT threat detection capabilities.

II. **Develop an innovative Transformer-based model tailored specifically for IoT threat detection, leveraging advanced Natural Language Processing (NLP) techniques.**

In this research, the focus will be on exploring a methodology for representing network traffic logs as sentences, enabling the classification of datasets using Transformer-based text classification models. The Transformer, a powerful deep learning model renowned for its effectiveness in natural language processing tasks, will be investigated to assess its classification capabilities specifically in the domain of classifying IoT network threats.

III. **Develop the novel few-shot learning model for IoT threat classification.**

The exploration of few-shot learning is an essential component of this research focused on AI-based solutions for classifying IoT threats. The utilization of few-shot learning can help mitigate the need for extensive resources in producing and labelling new datasets specific to IoT threats. As a result, the development and performance evaluation of a few-shot learning model will be undertaken.

IV. **Implement model retraining using transfer learning approaches to enhance prediction accuracy for unseen attacks.**

In the realm of IoT threat classification, the susceptibility of deep learning classifiers to previously unseen attack vectors pose a significant challenge. To mitigate this vulnerability, the exploration of transfer learning methods emerges as a promising avenue. Transfer learning techniques leverage knowledge gained from pre-existing tasks or domains to enhance the performance of classifiers on new, related tasks. By adapting pre-trained models to accommodate the unique characteristics of IoT threat landscapes, transfer learning holds the potential to bolster the resilience of classifiers against previously unseen attacks.

1.3 Project Scope and Direction

This research project focuses on the development and evaluation of two innovative methods for detecting threats in IoT environments. The first method involves using a DistilBERT-based text classification model to analyze text-based network information and classify IoT network threats. The second method explores the use of the Sentence Transformer Fine-Tuning model (SetFit) to enhance the performance of threat classification, particularly when dealing with limited training datasets. By investigating the effectiveness of these approaches, the project aims to improve the accuracy and efficiency of IoT threat detection.

In addition to utilizing publicly available IoT network datasets, this research project will establish an experimental network testbed to gather attack data under controlled conditions. The testbed will simulate a realistic small-scale network environment, enabling the collection of authentic attack traffic logs for assessing model performance across various network architectures and attack types. By employing this controlled data collection approach, the aim is to generate a labeled dataset of IoT threat logs that accurately represent real-world unseen attack scenarios. This dataset will serve as the basis for evaluating the multiclass IoT threats classifier's ability to predict unseen threats. Furthermore, it will be instrumental in the model retraining process, facilitating adaptation to new data and enhancing overall model effectiveness.

In addition to the development and evaluation of the previously mentioned approaches, this research project will explore model retraining methods to mitigate the vulnerability of deep learning classifiers to unseen attacks. Transfer learning techniques will be employed to leverage knowledge from pre-existing models, thereby enhancing

classifier performance on new, related tasks. By adapting pre-trained models to suit the distinct characteristics of IoT threat landscapes, model retraining offers the potential to enhance the resilience of classifiers against previously unseen attacks. This endeavor aims to improve the overall effectiveness of threat detection systems by ensuring their ability to accurately identify emerging threats.

1.4 Contribution

This research significantly advances the field of IoT security by developing a first-of-its-kind few-shot learning model for threat detection using limited labelled data. The proposed approach leverages the powerful natural language processing abilities of Transformer architectures to analyse IoT network traffic logs. This eliminates the need for extensive pre-processing of real-world IoT data, increasing the model's applicability to diverse network environments.

The integration of few-shot learning techniques with a Transformer-based classification framework represents a significant advancement in the field of IoT security. This innovative approach enables the model to achieve remarkably accurate threat detection, even when trained on small, labelled datasets. Few-shot learning techniques empower the model to rapidly adapt and generalize from limited examples, making it highly effective at identifying new and emerging threats in IoT environments. By leveraging the natural language processing capabilities of Transformer architectures, the model gains a deeper understanding of complex patterns in IoT network traffic data, enhancing its ability to discern between normal and malicious activities. Overall, the integration of few-shot learning with Transformer-based classification frameworks represents a significant step forward in advancing the capabilities of AI-driven threat detection systems for IoT ecosystems.

Another novel contribution of the research is the effort on generating network threats logs dataset labelled with contemporary threat classes. This dataset aims to test the model performance on different attacks in real world situations. Importantly, such efforts are relatively rare in the research community, as many existing datasets may not adequately represent the evolving landscape of IoT threats. By filling this gap, the research facilitates more accurate and meaningful evaluations of IoT security solutions,

ultimately contributing to the development of more effective threat detection mechanisms.

Besides that, another novel contribution is the exploration of the adaptability of the models to perform retraining using transfer learning approaches represents a novel and significant contribution to the research. A model that lacks the ability to continually improve and adapt to new network threats is not suitable for practical use. Transfer learning techniques have gained considerable attention in machine learning and deep learning fields for their ability to leverage knowledge from pre-existing tasks or domains to enhance the performance of models on new, related tasks. In the context of IoT security, where the threat landscape is constantly evolving, transfer learning offers a promising approach to improve the robustness and adaptability of threat detection model efficiently.

The powerful yet data-efficient few-shot Transformer model and comprehensive evaluation dataset are expected to create a complete model building ecosystem. Overall, the work establishes a new benchmark for handling limited labelled data scenarios through synergistic applications of NLP and few-shot learning.

1.5 Report Organization

The research consists of several chapters that provide a comprehensive examination of the topic. Chapter 2 delves into related backgrounds by reviewing relevant literature and previous studies, establishing a foundation of knowledge in the field. In Chapter 3, the proposed research methodology is described in detail, outlining the methods or approaches developed to address the research problem. Chapter 4 of the dissertation delves into the illustration of the system model, elucidating the development and utilization of the model. Chapter 5 focuses on experiments and simulation performed to test the developed model. Next, Chapter 6 states all the evaluation on the performance of the IoT threats detection models. Finally, Chapter 7 concludes the research by summarizing the key findings, implications, and contributions.

Chapter 2 Literature Review

2.1 Previous works on Generating IoT Threats Datasets

There are several datasets being published in the past few years for the IoT security domain. These datasets are important for researchers to create and test new AI intrusion detection models that are more robust. Therefore, a high-quality dataset becomes an important factor to ensure the reliability of models. The datasets are generated using different testbed environments and different features are collected in every dataset. Below are some of the works on generating IoT datasets.

2.1.1 ToN-IoT

The ToN-IoT datasets were specifically developed to address the limitations of previous datasets in the field. These datasets are designed using an orchestrated architecture that illustrates the interconnectedness of edge, fog, and cloud layers within an IoT environment [1]. The testbed environment in ToN-IoT is carefully constructed based on interactive network elements, aiming to simulate a realistic representation of IoT and IIoT network configurations [2]. One important aspect of ToN-IoT is its heterogeneity, which is a key property of modern IoT network intrusion detection datasets [3]. The datasets encompass diverse data sources, including telemetry data from IoT and IIoT sensors, datasets from Windows and Linux, as well as network traffic datasets. This heterogeneity enhances the realism and complexity of the datasets, making them more representative of real-world IoT network scenarios. Figure 2.1.1.1 shows the testbed environment for ToN-IoT:

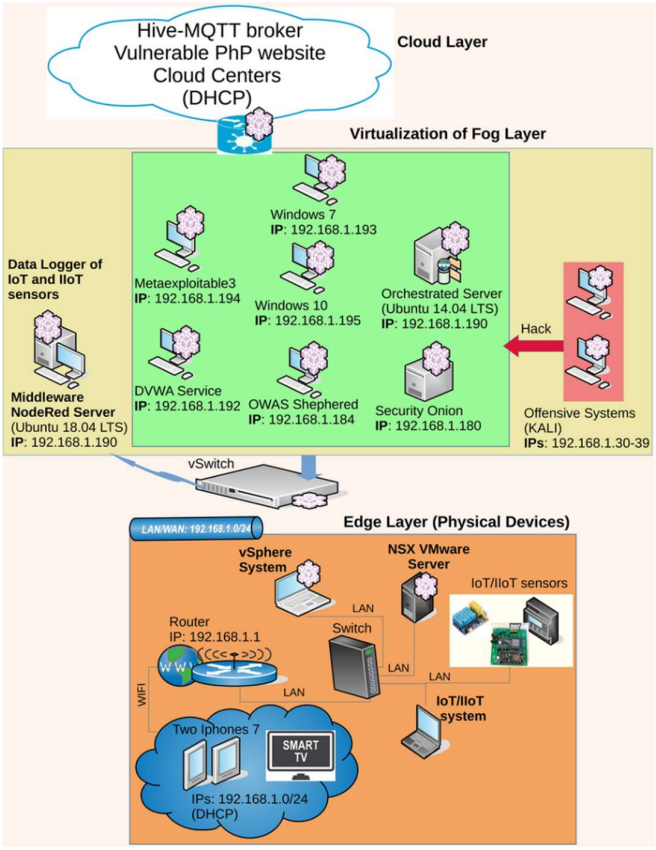


Figure 2.1.1.1: Testbed Environment for ToN-IoT Datasets Generation [1]

ToN-IoT network datasets contain a total of 43 features that description the network packets which contain mainly connectivity features information, layer 4 HTTP and DNS information, and layer 3 SSL information. Besides that, ToN-IoT also contains other types of datasets including Windows log, Linux log, and logs from IoT devices. The ToN-IoT datasets comprise a collection of normal scenarios and eight distinct main attack scenarios, covering a range of common cybersecurity threats encountered in IoT settings. The attacks encompass scanning, cross-site scripting (XSS), denial of service (DoS), ransomware, injection attack, password cracking, backdoor and man-in-the-middle (MITM) attack. However, ToN-IoT datasets consist of simulated environment, it may be questioned whether it can be applied to realistic IoT environments. It does not implement IoT devices as attackers which is one of the important threats that IoT environments face.

2.1.2 CICIoT2023

CICIoT2023 dataset is a new IoT attack dataset which is created from a massive amount of real IoT devices. The topology of the network used in the research consists of a total of 105 IoT devices. Out of these devices, 67 IoT devices were involved in the attacks, while the remaining 38 devices were Zigbee and Z-Wave devices connected to 5 hubs. The configuration of the network closely resembles a real-world smart-home environment, complete with various smart home devices, sensors, cameras, and micro-controllers [4]. The novel part of this dataset is that IoT devices are being used as malicious agents where other datasets seldom perform this.

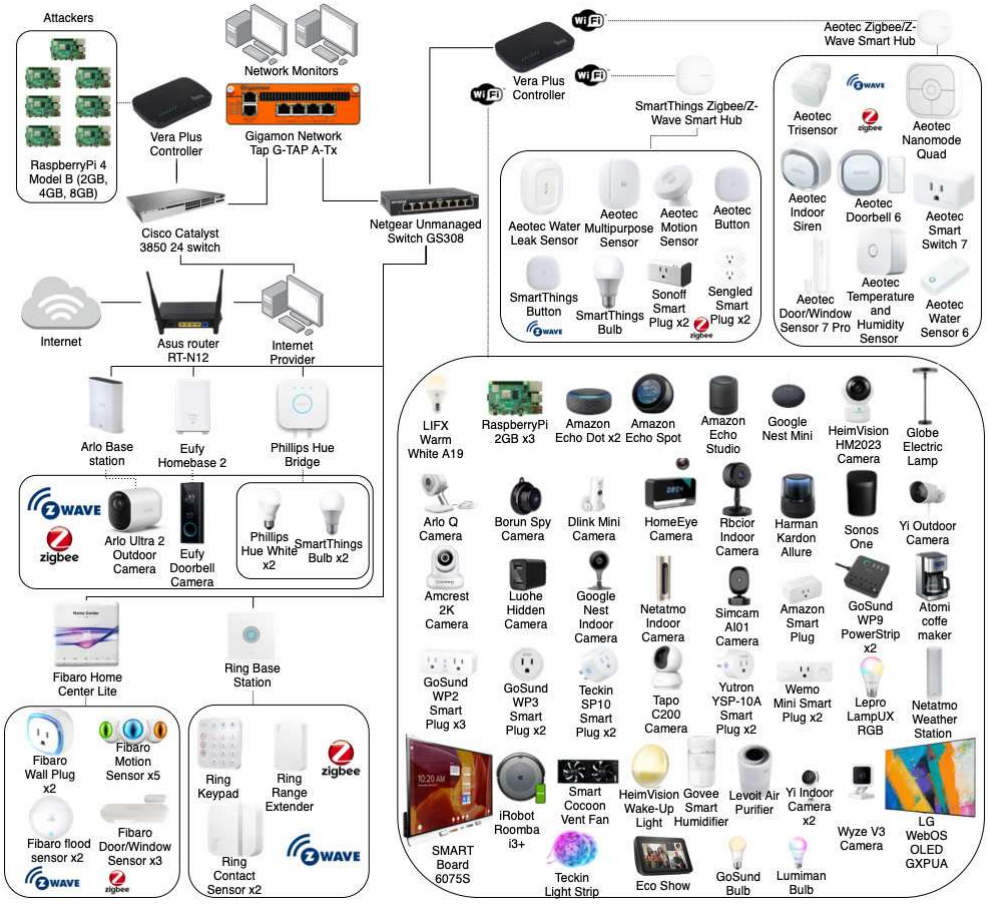


Figure 2.1.2.1: Topology Chart of CICIoT2023 [4]

While the basic attack framework can be seen in the Figure 2.1.2.2:

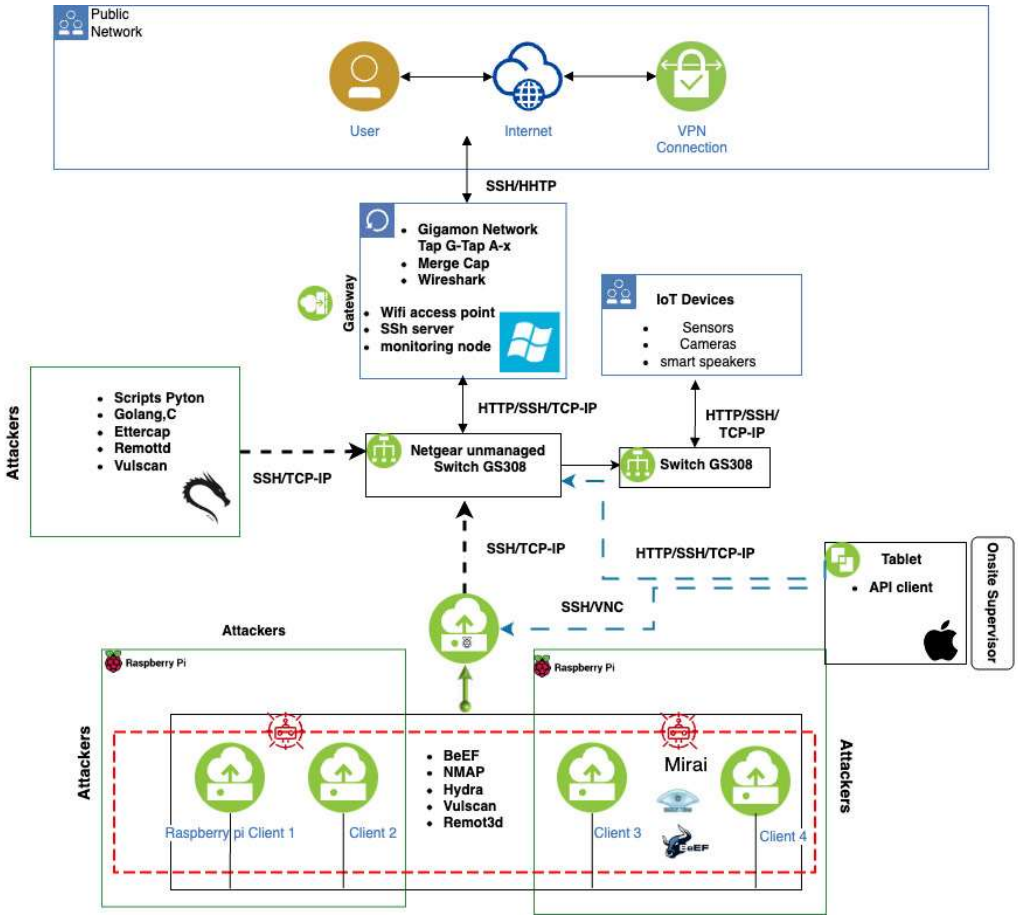


Figure 2.1.2.2: Basic Attack Framework for CICIoT2023 [4]

This dataset has collected 46 different features which consists of mainly of protocol information and some packet information. 33 attacks are performed during the production of dataset, and it is classified into 7 main classes. The 7 main classes include DDoS, Brute Force, Spoofing, DoS, Recon, Web-based and Mirai. However, the limitation of this datasets is that they do not include cloud and fog layer in their topology.

2.1.3 IoT-23

The IoT-23 dataset was specifically developed to provide researchers with a substantial dataset containing real instances of IoT malware infections as well as benign IoT traffic. The dataset serves as a valuable resource for the development and evaluation of AI-based threat detection models [5]. In the process of capturing the traffic data for the dataset, three IoT devices were utilized: a Philips HUE smart LED lamp, an Amazon Echo home intelligent personal assistant, and a Somfy smart door lock. This hardware ensures that the real network behaviour is captured and not simulated network behaviour. Figures below shows the hardware that is used in the dataset generation.

Figure 2.1.3.1: Amazon Echo Device Used in IoT-23 [5]

Figure 2.1.3.2: Philips Hue Device Used in IoT-23 [5]

Figure 2.1.3.3: Somfy Door Lock Device Used in IoT-23 [5]

There are 20 malware attacks performed in the datasets collection. The dataset is specifically labelled after the malware captures analysis to provide a detailed information of the malwares. The data captures are classified with 10 different labels which is attack, benign, Mirai, FileDownload, C&C, HeartBeat, DDoS, Okiru, PartofAHorizontalPortScan, and Torii. The limitation for this dataset is that it does not shows high heterogeneity because its testbed devices are limited.

2.1.4 Edge-IIoTSet

Edge-IIoTSet is also a realistic cybersecurity dataset of IoT and IIoT which aims to design a dataset that suits the training for centralized and federated models [6]. Other datasets seldom cover the Industrial Internet of Things (IIoT) field whereas Edge-IIoTSet does. Its testbed is designed to consists of 7 interconnected layers which is edge layer, Blockchain layer, fog layer, NFV layer, cloud computing layer, SDN layer, and IoT/IIoT perception layer. Figure 2.1.4.1 shows about the testbed architecture for Edge-IIoTSet and the proposed dataset generation framework.

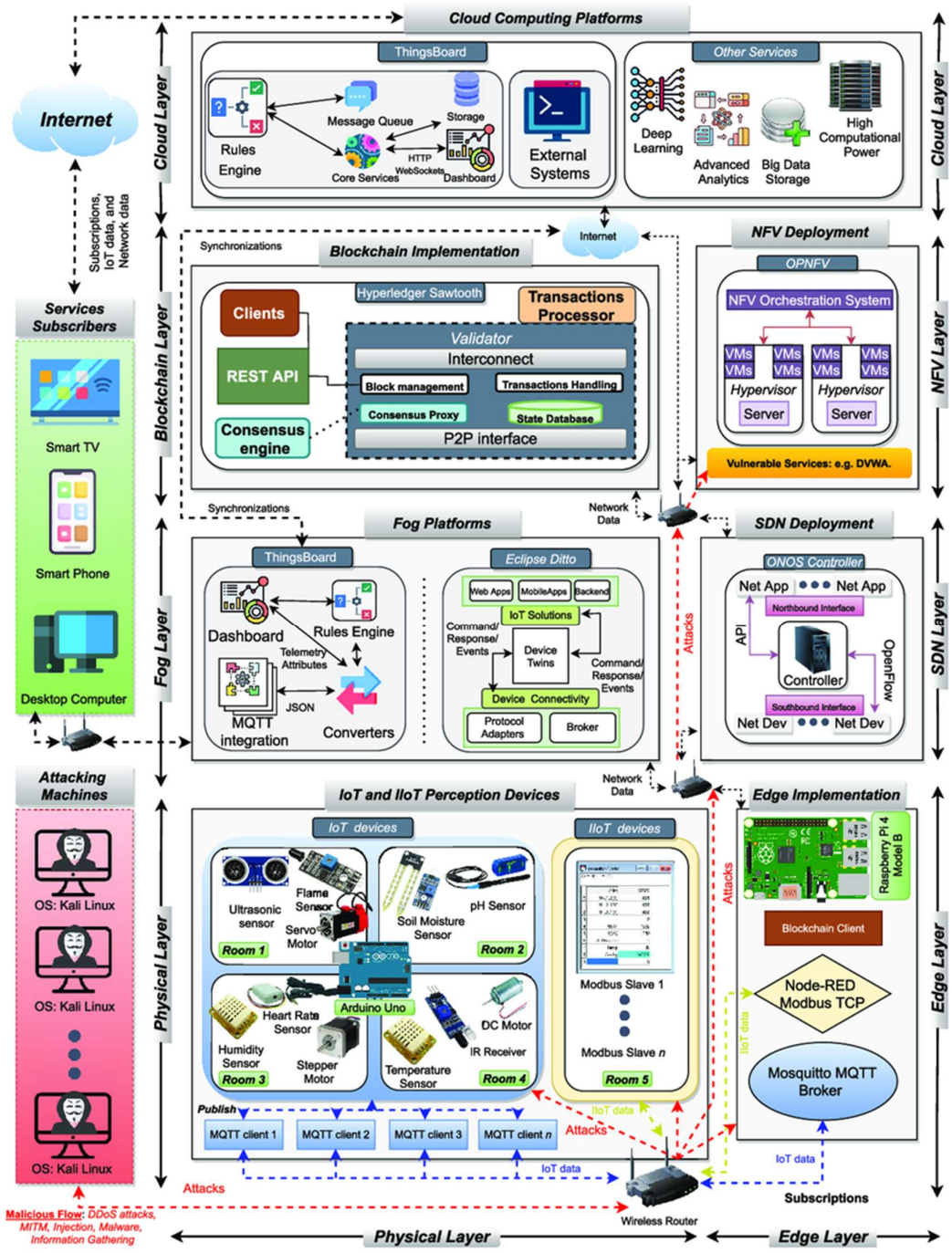


Figure 2.1.4.1: Testbed Architecture of Edge-IIoTSet [6]

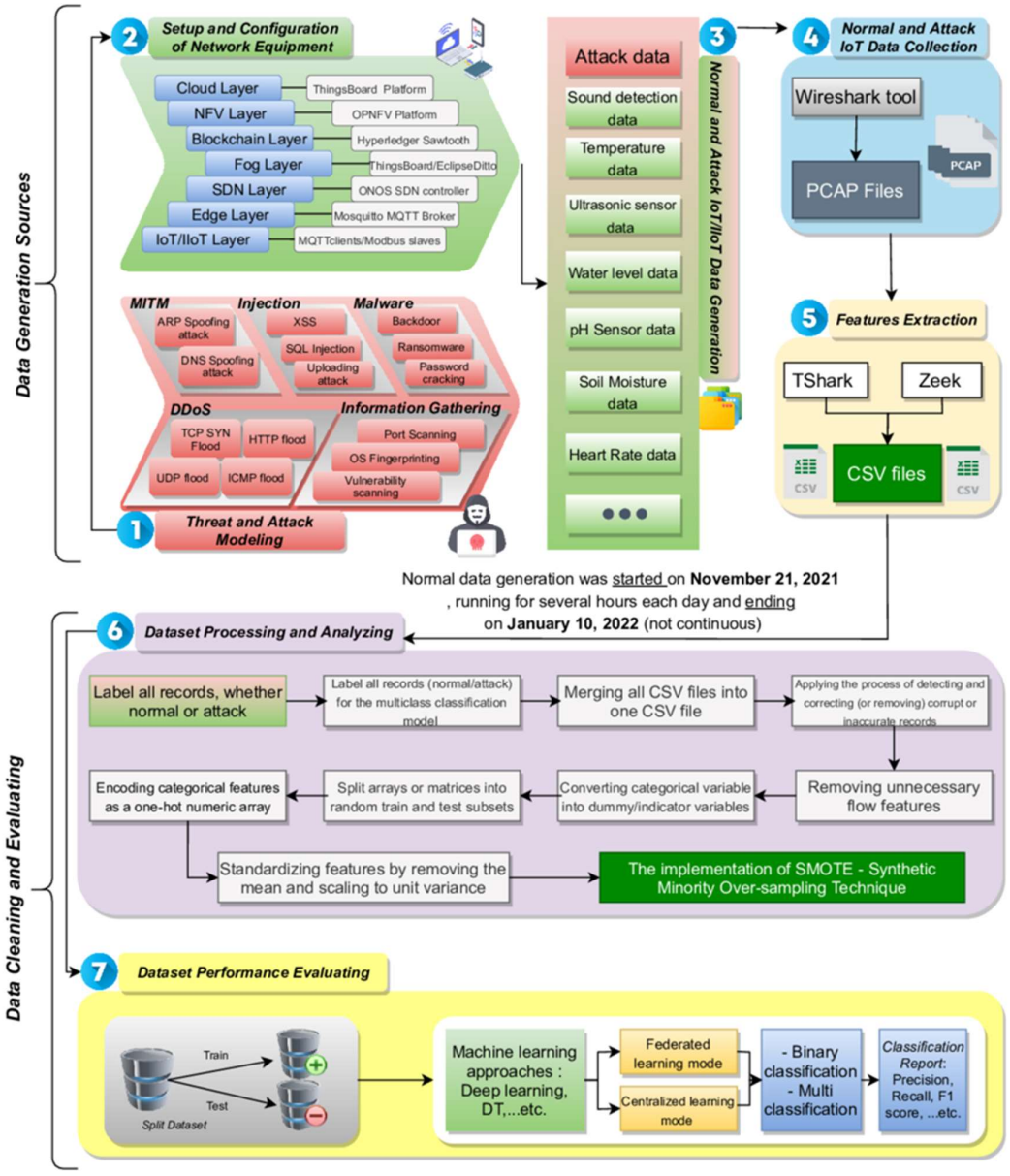


Figure 2.1.4.2: Dataset Generation Framework of Edge-IIoTSet [6]

There are a total 14 different classes of attacks being executed in the collection of datasets and it can be specified into 5 main classes which is DDoS, Injection, MITM, Malware, and Scanning. The dataset undergoes feature selection and there are 61 features left in the csv files that consists of mainly layer featured information. However, the number of malicious scenarios in the datasets is lesser when compared to other datasets.

2.2 Previous Works on IoT Threat Detection Model

Machine learning and deep learning methods are continuously discovered by researchers in their abilities on IoT threat detection. IoT threat detection models that are built from various AI techniques will be discussed in the section below.

2.2.1 Machine learning Methods

There are various machine learning models that were developed and tested on IoT threat detection datasets such as ToN-IoT, Edge-IIoTSet, and IoT-23. Machine learning techniques are discovered in the papers [2], [7], [8], [9], [10].

In [2], Support Vector Machine (SVM), k-Nearest Neighbour (kNN), Random Forest (RF), Classification and Regression Trees (CART), Logistic Regression (LR), Linear Discriminant Analysis (LDA),Naïve Bayes (NB), and Long Short-Term Memory (LSTM) are performed and evaluated. It uses all of the models stated to perform classification on the IoT datasets. Both binary classification and multiclass classification are evaluated in the paper with k-fold cross-validation method (k = 4). The Classification and Regression Trees (CART) model outperformed al others model in the paper and get an accuracy of 0.88 in binary classification and 0.77 accuracy in multiclass classification. However, this is an approach of Machine Learning in the IoT log datasets but not a network dataset. It gets a lower accuracy compared with the network dataset approach with ToN-IoT.

In the paper [10], the authors also used ML techniques to perform multiclass classification on the ToN-IoT network dataset. In this paper, only 14 out of 46 features are used in the model after the feature extraction. All of the feature selected is about the connection features of network packets only. Random Forest (RF), Naïve Bayes (NB) and k-Nearest Neighbour (kNN) are trained and validated. Among the ML techniques used, k-Nearest Neighbour (kNN) had outperformed the other models and get an accuracy of 98.2%. Whereas Naïve Bayes (NB) was the model with worst accuracy. Password cracking threats were among the hardest threats to be recognized compared to other threats. The limitation of this approach is that there is no binary classification being performed and there is no proper data preprocessing features to handle the imbalance dataset.

The authors in [9] suggests 8 ML models which is Support Vector Machine (SVM), Decision Tree (DT), Logistic Regression (LR), k-Nearest Neighbour (kNN), naïve bayes (NB), Random Forest (RF), AdaBoost and XGBoost to classify the ToN-IoT network dataset. The authors address the class imbalance issue and feature selection issue in the ToN-IoT dataset. Therefore, Synthetic minority oversampling technique (SMOTE) and Chi^2 are being adopted and tested for their effectiveness. 4 types of processed data were being fed to the model for training which includes original featured data, data after Chi^2 operation, data after SMOTE operation and data after both Chi^2 and SMOTE were applied. For binary classification, XGBoost model with original featured data outperformed other models and achieved an accuracy of 0.991%. While XGBoost model with SMOTE also get 0.990% which is similar, but it had proven that SMOTE and Chi^2 do not improve the accuracy of model well. For multiclass classification, XGBoost model Was still the best performer by achieving 0.983% in accuracy for original featured data. Similarly, it proved that SOMTE and Chi^2 do not improve the accuracy of model.

2.2.2 Deep Learning Methods

In the paper [11], the authors had proposed a new deep learning model to perform cyber threat classification task. 3 different trending IoT threat datasets which are Edge-IIoTset, ToN-IoT, and UNSW_NB15 are selected to perform validation on the proposed deep learning model. The proposed model contains 3 blocks which are the temporal representation block (TRB), the residual-based spatial representation (RSR) block, and the detection block (DB).

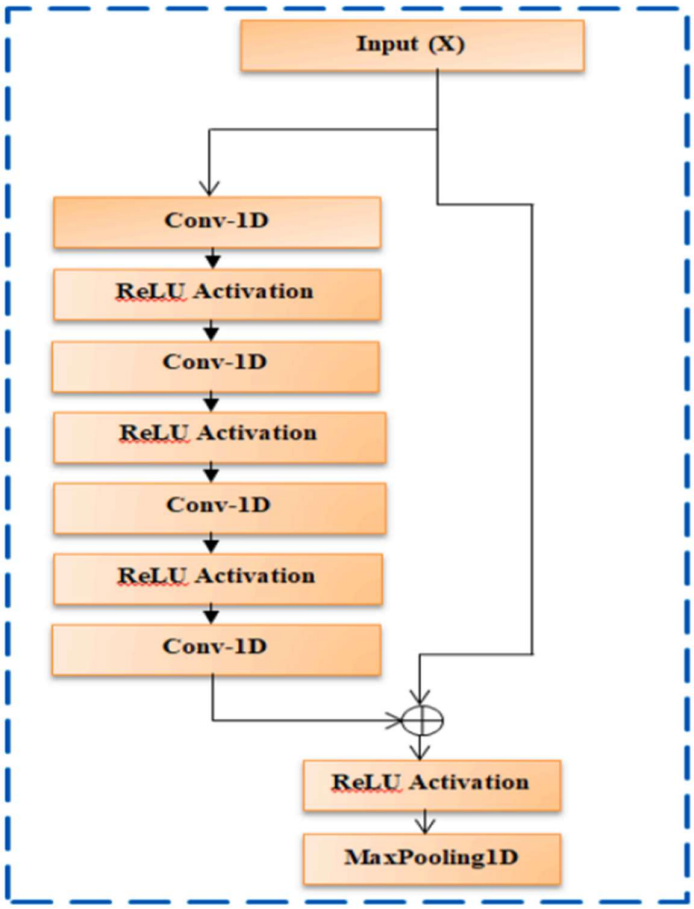


Figure 2.2.2.1: Residual Block of the Proposed Model [11]

The residual-based spatial-representation block is composed of five residual blocks, each containing four parallel convolutional layers. In addition, it incorporates a skip connection to mitigate the vanishing gradient problem. This block functions as a mechanism to extract spatial representations from the output of the preceding layer, effectively capturing significant spatial features that are inherent in the data.

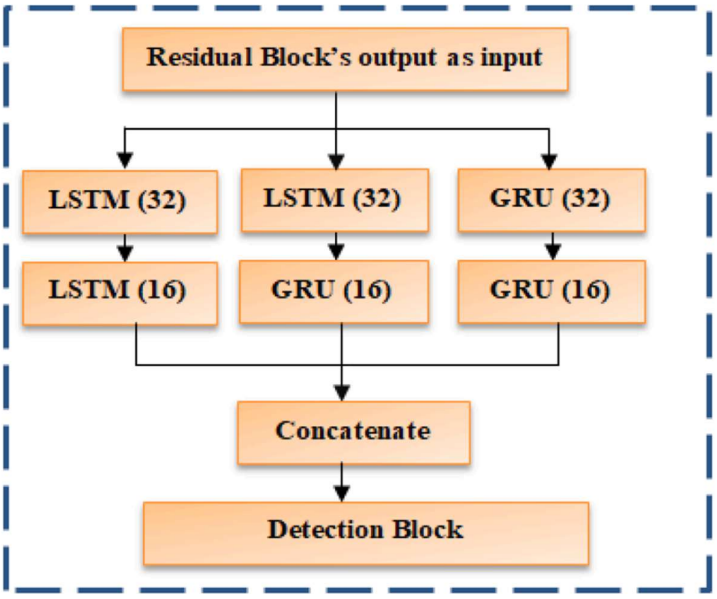


Figure 2.2.2.2: Temporal Representation Block of DeepAK-IoT Model [11]

The temporal representation block takes the output from the residual-based spatial-representation (RSR) block and feeds it into three parallel paths. This block is designed to learn and capture temporal representations, allowing for more accurate detection of cyber threats.

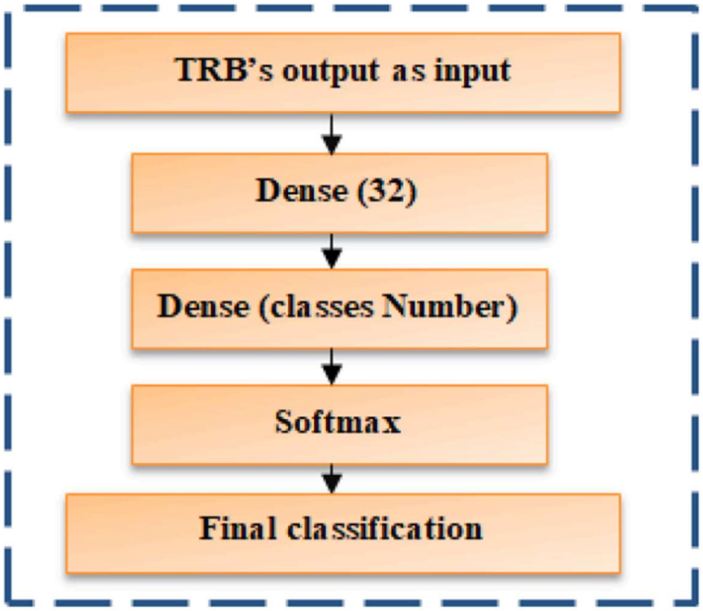


Figure 2.2.2.3: Detection Block of DeepAK-IoT Model [11]

The detection block, the final component of the model, is comprised of two layers. The first layer is a fully connected layer with 32 neurons, followed by a second fully connected layer responsible for estimating the cyber threat class. These layers are connected to a SoftMax activation function, which outputs the probability score for each class.

The model has achieved impressive accuracy rates, with a reported accuracy of 90.57% for the ToN-IoT dataset, 94.96% for the Edge-IIoT dataset, and 98.41% for the UNSW-NB15 dataset. These results surpass the performance of alternative models, such as the 1DCNN model, CNN-LSTM model, and LSTM, indicating the effectiveness and superiority of the proposed model in detecting and classifying cyber threats in IoT and IIoT environments.

In the research described in [12], two CNN models, namely the Inception Time model and the DenseNet model, were proposed and evaluated using three distinct datasets focused on IoT threats: ToN-IoT, Edge-IIoT, and UNSW-NB15. The DenseNet model, specifically the state-of-the-art DenseNet121 model, was utilized. DenseNet differs from traditional CNNs by incorporating direct connections between any two layers, facilitating improved information flow within the network. This model requires fewer parameters compared to conventional CNNs, making it more efficient, and allows for reusability of features.

The DenseNet architecture involves a sequence of connected layers called a dense block. In this block, the dimensions of the feature maps remain consistent, while the number of filters used varies. Each dense block consists of multiple interconnected layers, where each layer comprises a 1D convolution, batch normalization, and ReLU activation. Transition layers are positioned between adjacent dense blocks to link them together. These transition layers serve the purpose of reducing the dimensions of the feature map, followed by average pooling. Their role is to gather the outputs or blocks from the preceding block.

The architecture concludes with fully connected softmax and output layers, which are responsible for generating the final predictions based on the learned features. This comprehensive architecture, leveraging DenseNet121, provides an effective and robust approach for detecting and classifying IoT threats in the datasets considered in the research. The architecture of DenseNet121 is illustrated in Figure 12.

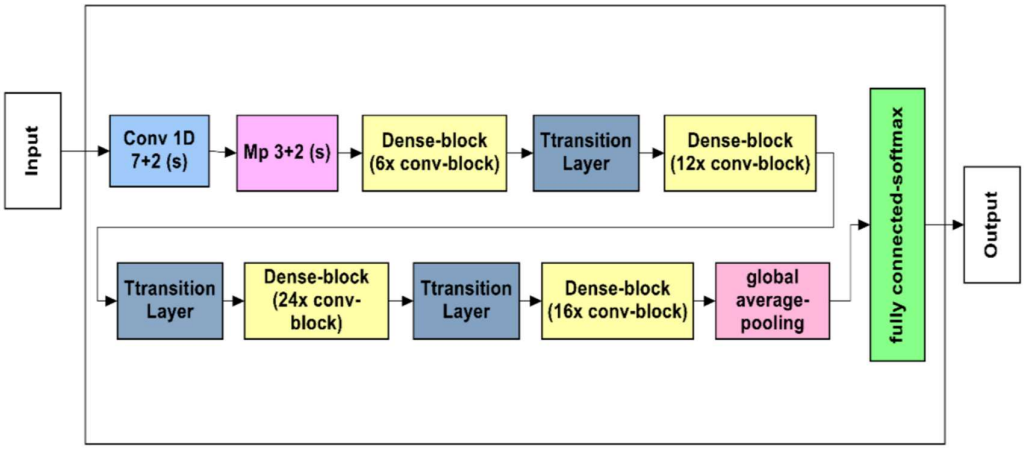


Figure 2.2.2.4: Architecture diagram of DenseNet121 [12]

Next, in the study described in [12], the Inception Time model was also investigated. This model optimizes computational resources while minimizing additional computational burdens. It achieves high-performance output by extracting features from input data across multiple scales using convolutional filters of varying sizes. The Inception Time model is employed in two configurations within the inception network: one with 1D input vector shape and another with 2D input shape obtained through the sliding window technique for time-series data.

Each block in the Inception Time model consists of three inception modules. To reduce input dimensions, a bottleneck layer is utilized, which helps decrease parameters and computational costs. A 1D convolutional layer acts as a sliding filter, enabling discrimination of regions in the time series and enhancing training speed and generalization. The output from the bottleneck layer is then fed into three one-dimensional convolutional layers with kernel sizes of 10, 20, and 40, respectively.

In addition, the inputs of the inception module pass through a max-pooling layer with a size of 3, facilitated by the bottleneck layer. The outputs from the four convolutional layers are concatenated along the depth dimension using a depth concatenation layer. All layers, except the concatenation layer, share the same stride and padding. Each convolutional layer employs 32 filters, and residual connections are incorporated into every third inception module.

The network consists of successive inception modules, followed by a batch normalization layer, a GlobalAveragePooling1D layer, and a dense layer with softmax activation for

classification purposes. This architecture enables effective feature extraction and classification of IoT threats using the Inception Time model.

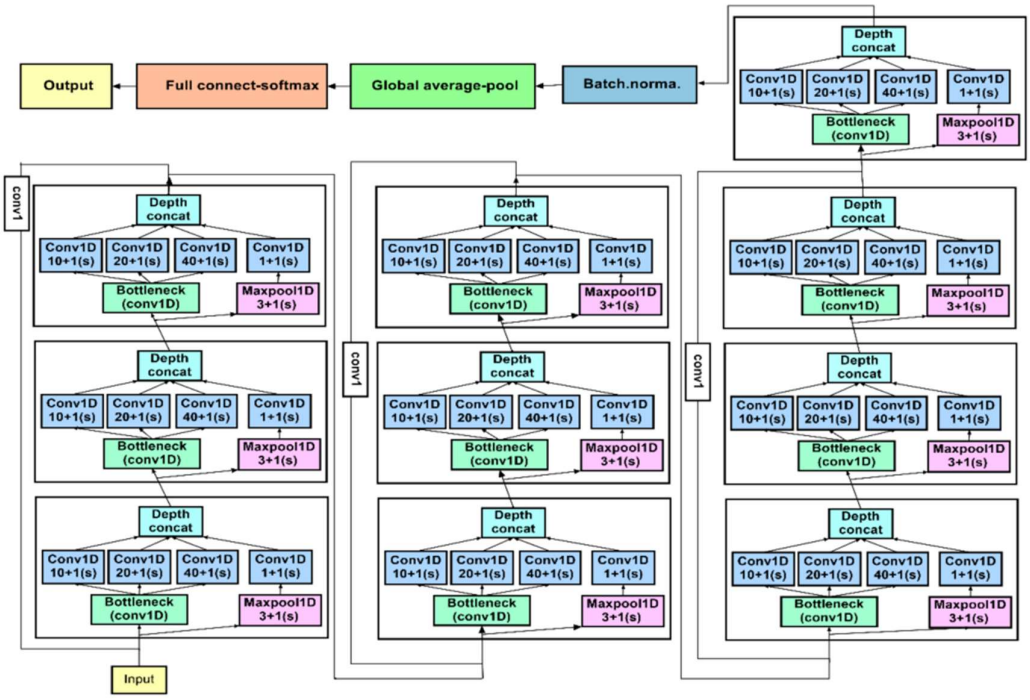


Figure 2.2.2.5: Architecture of Inception Time model [12]

Both of the models are evaluated using ToN-IoT datasets which includes windows 7, windows 10, network and combination of windows 10 and network datasets, UBSW-NB15 dataset, and Edge-IIoT dataset. Multiclass classification was being performed and evaluated for all the datasets. The Inception Time model achieved the highest, 100% accuracy on combination of Windows 10 and Network dataset. The results of the validation will be summarized in the Table 2.2.3.1. However, Inception Time model had outperformed the DenseNet model in all of the results.

In [13], a deep learning-based intrusion detection system (IDS) with a transfer learning approach is proposed. The paper utilizes transfer learning by freezing most of the layers and training only the last layers using a convolutional neural network (CNN) on the Bot-IoT and ToN-IoT datasets. The CNN-based IDS is initially trained on the Bot-IoT dataset, and then the ToN-IoT dataset is used to update the model.

The update operation involves freezing the pre-trained convolutional base and using its output as input to the classifier. The classifier is then retrained on a combined training dataset, which comprises 50% of the ToN-IoT dataset and 10% of the Bot-IoT dataset. This approach helps prevent the model from being solely influenced by the behavior of the new dataset.

The model architecture is characterized by several components. It starts with an input layer consisting of 16 input neurons, corresponding to the number of features. The model includes five hidden layers which is the Convolution1D layer, MaxPooling1D layer, Flatten layer, ReLU layer, and Dense layer. Finally, it concludes with an output layer.

During the initial training phase, the model undergoes 10 epochs, which means the complete dataset passes through the neural network 10 times. Each training iteration involves batches of 32 samples, with the batch size indicating the number of samples processed in each iteration. The neural network comprises a total of 16 input neurons, aligned with the number of features. It consists of four intermediate (hidden) layers, with the following number of neurons in each layer: 16 in the Convolution1D layer, 8 in the MaxPooling1D layer, 256 in the Flatten layer, 256 in the ReLU layer, and 44 in the Dense layer. The output layer consisted of 4 neurons, corresponding to the multiclass classification requirement, as visually depicted in Figure 2.2.2.6.

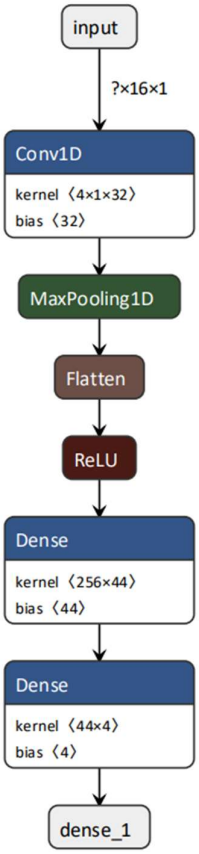


Figure 2.2.2.6: Architecture of the original IDS model [13]

During the updating operation, the convolutional base which includes the stack of Convolution1D layer, MaxPooling1D layer, Flatten layer, and ReLU layer will be frozen. Only the 2 Dense layers at the bottom will be retrained. This model is only created to predict 4 classes of threats which are DDoS, Dos, Reconnaissance, and Theft. A validation dataset which consists of 15% of the TON-IoT dataset concatenated with 5% of BoT-IoT dataset. The updated model can achieve an accuracy of 99.47% on the validation test. In this transfer learning approach, it also decreases the time needed to train the model. Updating the model only require 170 seconds for 10 epochs while training the whole model takes 31590 seconds for every epoch.

2.2.3 Deep Learning with Transformer Architecture

In the research presented in [14], the application of the Transformer model in IoT intrusion detection was investigated. The proposed system, called TransIDS, leverages the Transformer's capabilities to adaptively adjust attention to IoT network traffic features and effectively utilize

deep global information from these features. To enhance the generalization ability of the model, Label Smoothing Regularization was implemented in the IDS framework. This technique adds fuzzy noise to the training label samples, helping the model to better handle uncertainties and improve its performance. The paper also analyzes and compares the effectiveness of different hyperparameters in the TransIDS framework. Specifically, the multi-headed attention mechanism and self-attention mechanism are studied to determine their impact on the model's performance and intrusion detection capabilities.

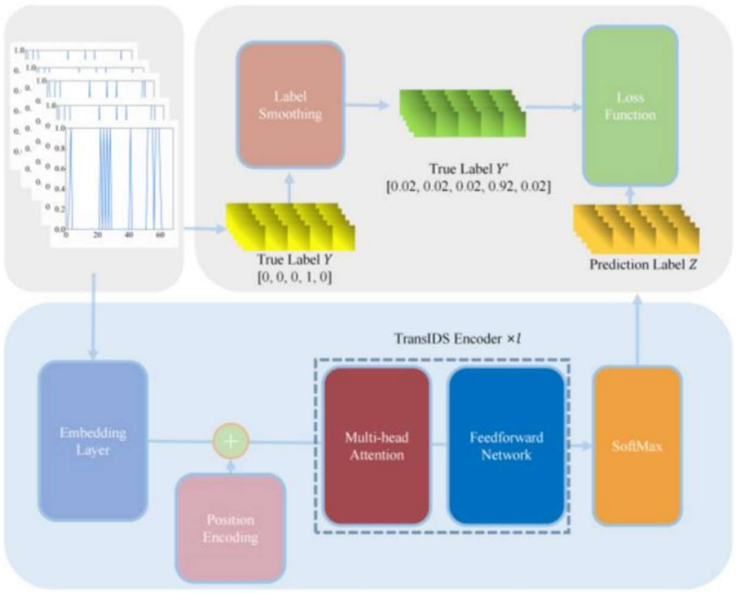


Figure 2.2.3.1: Architecture of TransIDS model [14]

Transformers use an encoder to extract deep features from time series data. The encoder computes the correlation between time steps using self-attention. However, it cannot effectively capture the positional information within the time series. Since the ordering and position of each time step is important for time series data, transformers rely on positional encodings. By embedding positional encodings for each time step, the self-attention mechanism can learn correlations that account for positional information when processing the time series. This allows it to effectively model the time series while also considering the ordered position of each time step within the sequence. The multi-head attention mechanism extracts multiple global temporal patterns from the time series through parallel self-attention networks. This provides a more abundant set of temporal features. To preserve the original features, the multi-head attention mechanism in TransIDS utilizes the outputs from parallel

self-attention networks. Residual connections and layer normalization are then applied after the multi-head attention step. This allows the model to retain important information from previous layers and normalize the outputs. Finally, a feed-forward neural network processes the normalized outputs, capturing complex patterns and relationships in the data. The multi-layer encoder stack processes the inputs in this way to produce deep global features as the output. A softmax function in equation 7 then calculates the probability of each feature belonging to each category. The non-numeric features in the dataset are converted into numerical data with one-hot-encoding and label encoding for data preprocessing. Min-max normalization was also being implemented to convert all the data between 0 and 1. The processed data is then used to train and evaluate the Transformer model. The limitation of this approach is that data is still being converted numerically same as the traditional approaches.

Not only that, the authors in [15] proposed a BERT-based Transformer model for IoT threat classification for textual data. This transformer approach concatenates all the textual data and does not perform preprocessing on textual data to encode and normalize it. While LightBGM model are proposed for numerical data. Figure 2.2.3.2 shows the complete framework for a secured IoMT network.

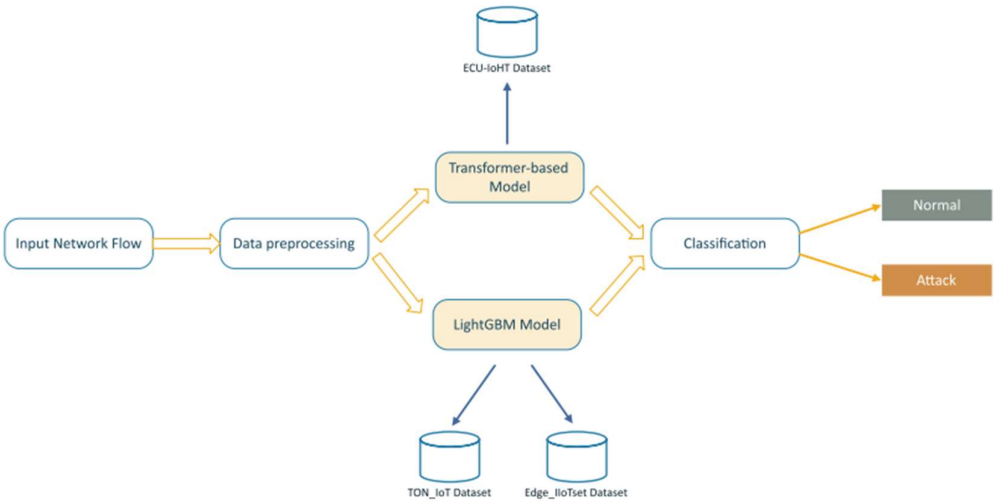


Figure 2.2.3.2: IoMT intrusion detection proposed in [15]

As a complex IoMT environment, it will have heterogeneous network flow. Therefore, the authors have classified it into 2 different types which network flows that consists textual features and network flows that consists mainly of numerical features and categorical features. 3 network datasets are being used in this paper to evaluate all of the models which includes ECU-IoHT, ToN-IoT, and Edge-IIoT. In the evaluation, BERT-based transformers had achieved 100% accuracy on the classification of ECU-IoHT because its network information is in textual form. However, LightBGM performs just slightly better than BERT on ToN-IoT dataset. Besides that, LightBGM also performed better than BiLSTM and DNN in comparison with Edge-IIoT dataset. This paper had proved that Transformer models are robust in text data classification, and it can be applied to threat detection field also.

In [16], the authors propose a Transformer-based IoT Network Intrusion Detection System (NIDS) method. They process the data from the ToN-IoT dataset by combining IoT telemetry data with network data. This combined data is then utilized to evaluate the FT-Transformer-based model. To prepare the data for the model, it is split into numerical data and categorical data. These two types of data are tokenized separately using different functions in the feature tokenizer. This tokenization process transforms the data into learnable embeddings, which can be understood by the model. The output of the feature tokenizer, consisting of the numerical and categorical embeddings, is then fed into multiple Transformer encoder blocks. In this

particular study, the authors employ N=6 stacked encoder blocks. Each encoder block includes a multi-head attention (MHA) layer and a fully connected feed-forward network. After each MHA layer and feed-forward network, a residual connection and normalization layer are applied. The key component of the Transformer encoder is the MHA layer. It allows the model to dynamically learn information from different feature embeddings. The MHA layer calculates M=8 heads of self-attention, which is also known as scaled dot-product attention. The input embedding is used to compute query (Q), key (K), and value (V) vectors. The weighted sums of the value vectors are then calculated by taking the dot product of Q with K and applying the softmax function to obtain the weights.

For classification, only the learned embedding from the input is fed to the final MLP layers. As an independent third-party embedding extracted via attention, it is more suitable for final classification compared to the numerical or categorical feature embeddings.

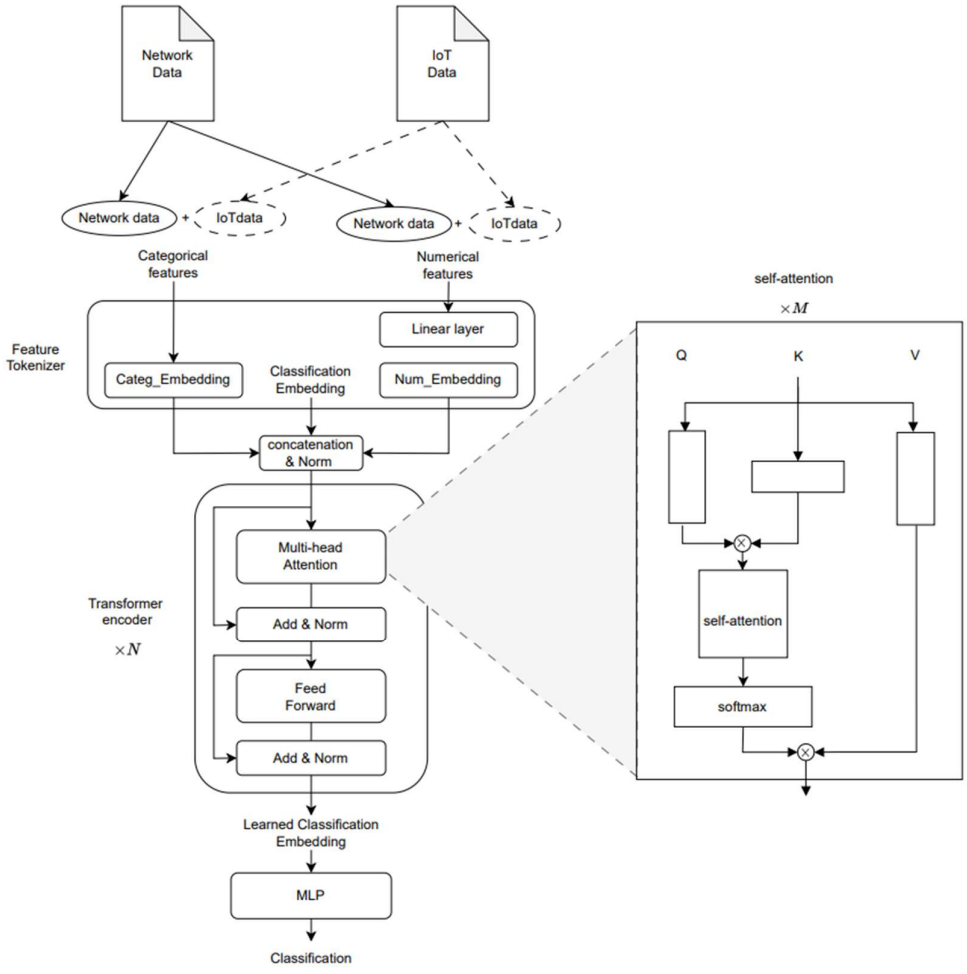


Figure 2.2.3.3: Overview of FT-Transformer proposed in [16]

The Transformer model is being evaluated with binary and multi-classes classification with ToN-IoT pure network data with combination of network and telemetry sensor data. The results were evaluated using accuracy, precision, recall, false alarm and F1 score. The accuracy of multiclass classification is tabled in table 2.2.4.1.

2.2.4 Summarization

Table 2.2.4.1: Comparison of Reviewed Model

Citation	Datasets	Methods	Accuracy (%)
[2]	ToN-IoT IoT Dataset	Logistic Regression (LR)	0.61
		Linear Discriminant Analysis (LDA)	0.68
		k-Nearest Neighbour (kNN)	0.84
		Classification and Regression Trees (CART)	0.85
		Random Forest (RF)	**0.88**
		Naïve Bayes (NB)	0.62
		Support Vector Machine (SVM)	0.61
		Long Short-Term Memory (LSTM)	0.81
[10]	ToN-IoT Network Dataset	Random Forest	94
		Naïve bayes	70
		K-Nearest Neighbour	**98.2**
[9]	ToN-IoT (Network)	Logistic Regression (LR)	**77.7**
		Naïve Bayes (NB)	71.2
		Decision Tree (DT)	93.4
		Random Forest (RF)	93.7
		K-Nearest Neighbour (kNN)	97.9
		Support Vector Machine (SVM)	77.9
		AdaBoost	39.9
		XGBoost	**98.3**
[11]	ToN-IoT	DeepAK-IoT	90.57
	Edge-IIoTset	DeepAK-IoT	94.96
	UNSW-NB15	DeepAK-IoT	**98.41**
[12]	ToN-IoT (Network)	DenseNet	98.57
		Inception Time	99.65
	ToN-IoT (Win10)	DenseNet	97.87
		Inception Time	98.30
	ToN-IoT (Win7)	DenseNet	98.36
		Inception Time	99.21
	ToN-IoT (Win10–N)	DenseNet	99.95
		Inception Time	**100**

	UNSW-NB15	Inception Time	98.60
	Edge-IIoT	Inception Time	94.94
[14]	ToN-IoT (Network)	TransIDS	99.46
[15]	ECU-IoHT	LightBGM	98.4182
		BERT	100
		BiLSTM	97.8301
	ToN-IoT (Network)	LightBGM	99.9934
		BERT	99.9852
		BiLSTM	97.3293
	Edge-IIoTset	LightBGM	100
		BERT	99.5659
		BiLSTM	100
[16]	ToN-IoT (Network)	FT-Transformer	95.78
	ToN-IoT (Combined Network & IoT)	FT-Transformer	**97.06**

2.2.5 Unknown Threat Detection

The paper[17] tested the unknown threat detection using binary IoT threat classifiers. We can note that although the known threats classification gets an 100% accuracy, the unknown threats classification does not perform well at an average accuracy of 46 % (which is the prediction on IoT-2 until IoT-8).

Dataset	Accuracy	Precision	Recall	F1-score
IoT-1	100	100	100	100
IoT-2	51	100	51	67.5
IoT-3	46.2	100	46.2	63.2
IoT-4	40.4	100	40.4	57.5
IoT-5	42.4	100	42.4	59.5
IoT-6	39.4	100	39.4	56.6
IoT-7	55	100	55	71.4
IoT-8	49.2	100	49.2	65.9

Figure 2.2.5.1: Result on detecting unseen threats for binary classification [17]

Chapter 3 Research Methodology

3.1 Research Methodology Diagram

Stage 1: Research on IoT Threat Classification Models

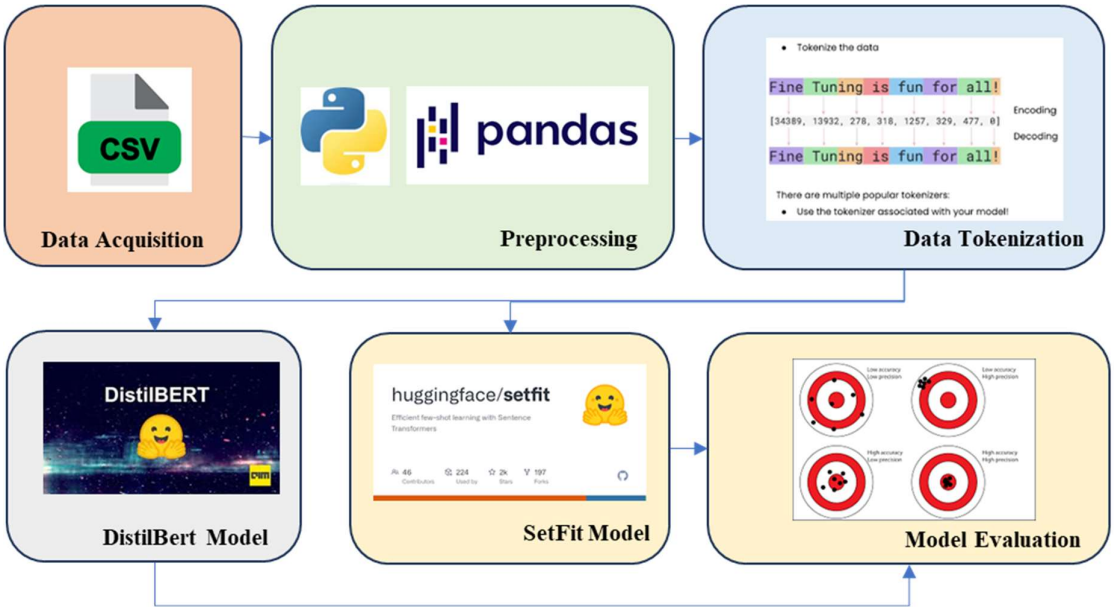


Figure 3.1.1 Flowchart for Research Stage 1

Stage 2: Generating Unseen Dataset for Model Evaluation

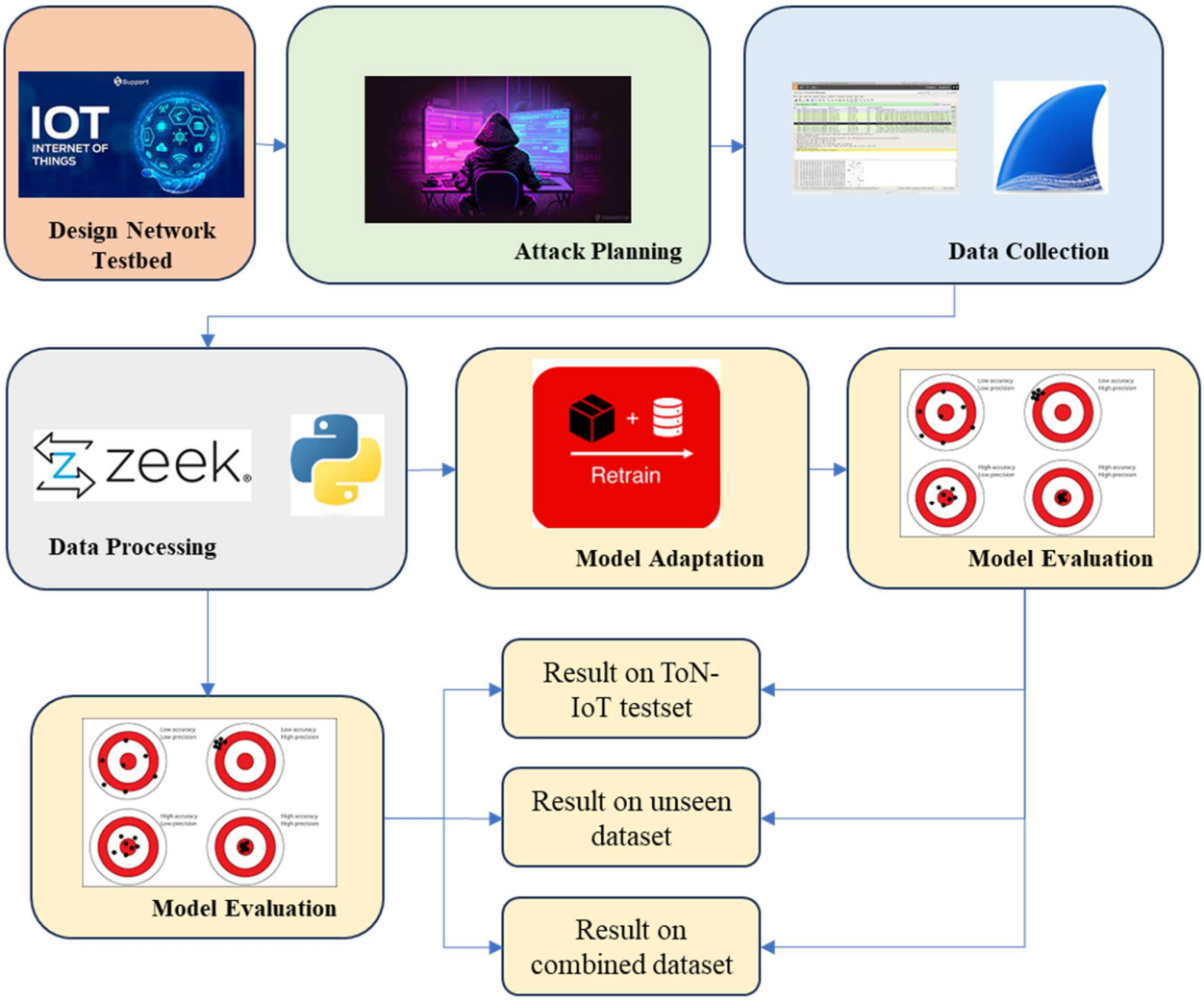


Figure 3.1.2 Flowchart for Research Stage 2

3.2 System Methodology Explanation

3.2.1 Dataset Acquisition

The ToN IoT dataset represents the next generation of Industry 4.0/Internet of Things (IoT) datasets, derived from a methodical testbed within a laboratory setting. It encompasses a variety of data sources, including sensor data, network data, and log data, all gathered from a single expansive and authentic network environment. This diverse nature of the ToN IoT dataset accurately mirrors the complexities inherent in IoT environments.

The ToN-IoT dataset has been chosen as the primary dataset for the model training task in this project due to its various advantages. One notable advantage is its resilience in terms of

heterogeneity. In contrast to numerous other datasets that transform textual information into numerical data and standardize the data, the ToN-IoT dataset preserves the original textual information from its network dataset. This characteristic facilitates a more thorough analysis of the dataset and safeguards against the loss of information during the transformation.

Additionally, the ToN-IoT dataset stands out for its inclusion of benign network data and 9 different types of attacks: Scanning, Dos, Injection, Ddos, Password, Xss, Ransomware, Backdoor, and Mitm. To represent these attack types, two attributes have been created in the dataset. The first is a binary column that indicates whether a record is a threat, and the second is a string value denoting the specific attack type. This comprehensive coverage of attack types, including the incorporation of ransomware attacks, sets the ToN-IoT dataset apart from many other datasets that lack such diversity.

Considering these factors, the ToN-IoT dataset is an ideal choice as the initial dataset for this project. It offers the advantages of preserving textual information, providing a comprehensive representation of attack types, providing a well-prepared train test dataset and including unique attack types like ransomware. These qualities make the ToN-IoT dataset well-suited for the model training task and enhance its value for the research project.

In our research, we employ the network data sourced from the ToN-IoT dataset, specifically from the Train_Test_Network.csv file. This dataset is a condensed version extracted from the larger datasets. The purpose of using this simplified version is to mitigate the consumption of excessive resources and time that would be required when working with the complete dataset. Table 3 provides a summary of the number of records for each classification type, offering a concise overview of the dataset's composition.

Table 3.2.1.1: Attack types and counts

Type	Value Count
Normal	300000
Scanning	20000
Dos	20000
Injection	20000
Ddos	20000
Password	20000

Xss	20000
Ransomware	20000
Backdoor	20000
Mitm	1043

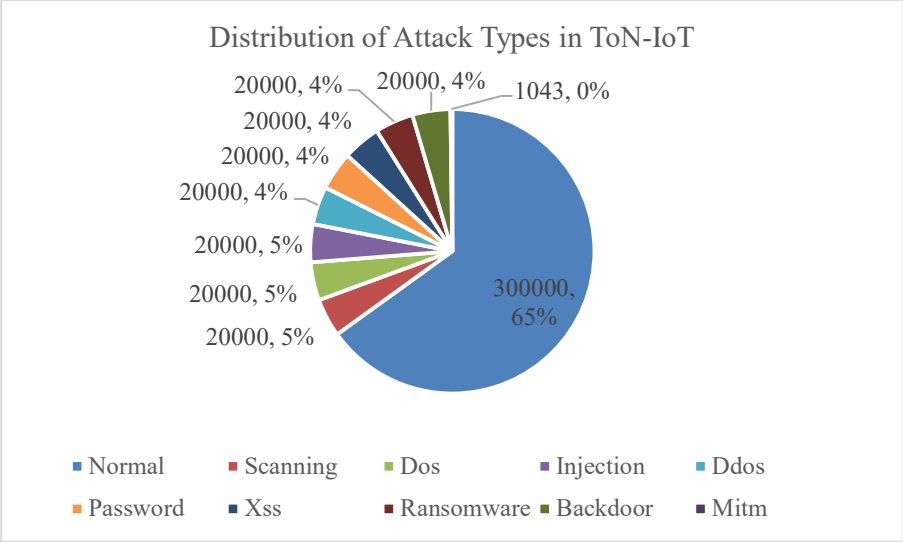


Figure 3.2.1.1: Distribution of Attack Types in ToN-IoT

Besides that, the dataset contains a total of 43 features and 2 label columns. The features will be recorded on table 3.2.1.2.

Table 3.2.1.2: Features in ToN-IoT Datasets

No.	Feature	Data Type	Description
1	ts	int64	Timestamp of connection between flow identifiers.
2	src_ip	object	IP addresses originating endpoints' IP addresses.
3	src_port	int64	TCP/UDP ports originating endpoint's traffic.
4	dst_ip	object	Destination IP addresses responding to endpoint's IP addresses.
5	dst_port	int64	Destination ports which respond to endpoint's TCP/UDP ports.

6	proto	object	Transport layer protocols of flow connections.
7	service	object	Dynamically detected protocols like DNS, HTTP, and SSL.
8	duration	float64	Time of the packet connections, estimated by subtracting the 'time of the last packet seen' from the 'time of the first packet seen'.
9	src_bytes	int64	Source bytes which are originated from payload bytes of TCP sequence numbers.
10	dst_bytes	int64	Destination bytes which are responded payload bytes from TCP sequence numbers.
11	conn_state	Object	Various connection states such as S0, S1, and REJ.
12	missed_bytes	int64	Number of bytes that are absent in content gaps.
13	src_pkts	int64	Number of original packets estimated from source systems.
14	src_ip_bytes	int64	Total length of IP header field of source systems.
15	dst_pkts	int64	Number of packets sent from the destination.
16	dst_ip_bytes	int64	Total number of bytes sent from the destination.
17	dns_query	Object	Domain name subjects of DNS queries.
18	dns_qclass	int64	Values specifying DNS query classes.
19	dns_qtype	int64	Value specifying DNS query types.
20	dns_rcode	int64	Response code values in DNS responses.
21	dns_AA	object	DNS authoritative answer flag (True denotes server is authoritative for query).
22	dns_RD	object	DNS recursion desired flag (True denotes request recursive lookup of query).
23	dns_RA	object	DNS recursion available flag (True denotes server supports recursive queries).

24	dns_rejected	object	DNS rejected flag (True denotes DNS queries are rejected by the server).
25	ssl_version	object	SSL/TLS version offered by the server.
26	ssl_cipher	object	SSL/TLS cipher suite used by the connection (if applicable).
27	ssl_resumed	object	SSL flag indicating the session that can be used to initiate new connections.
28	ssl_established	object	SSL/TLS established session flag.
29	ssl_subject	object	Subject of the X.509 certificate offered by the server.
30	ssl_issuer	object	SSL/TLS issuer name of the certificate (if applicable).
31	http_trans_depth	object	Depth of the HTTP transaction (i.e., how many redirects were followed).
32	http_method	object	HTTP request method (e.g., GET, POST, PUT).
33	http_uri	object	URIs used in the HTTP request.
34	http_version	object	HTTP protocol version used by the client and server.
35	http_request_body_len	int64	Actual uncompressed content sizes of data transferred from the HTTP client.
36	http_response_body_len	int64	Actual uncompressed content sizes of data transferred from the HTTP server.
37	http_status_code	int64	HTTP status code returned by the server (e.g., 200, 404, 500).
38	http_user_agent	object	User agent string sent by the client.
39	http_orig_mime_types	object	Original MIME types of the HTTP request and response bodies.
40	http_resp_mime_types	object	Response MIME types of the HTTP request and response bodies.
41	weird_name	object	Name of the weird event detected by Zeek (e.g., " DNS_RR_unknown_type").
42	weird_addl	object	Additional information about the weird event.

43	weird_notice	object	Indication of whether a violation or anomaly has been converted into a notification or alert.
44	label	int64	Label assigned to tag normal and attack records (e.g., "0", "1").
45	type	object	Type of attack class.

3.2.2 Dataset Preprocessing

Removing Features

The initial step in the process involves identifying and eliminating specific features that could potentially impede the model's ability to generalize effectively. For instance, IP addresses, port numbers, and timestamps are prone to change within different network environments. By excluding these variables from the dataset, we can minimize any noise or irrelevant information that may impact the model's performance in real-world deployments.

Furthermore, a substantial portion of the dataset's features were found to contain null values for over 80% of the records. Recognizing the computational burden associated with processing such features, we opted to remove them from consideration. Specifically, features related to DNS, SSL, HTTP, and violation information were eliminated to streamline the dataset and reduce computational complexity. We will only use connection features as the dataset input.

In summary, our approach involves training models based on two distinct sets of features, allowing us to explore and evaluate the impact of feature selection on model performance. This strategic approach aims to enhance the efficiency and effectiveness of our models in addressing real-world challenges in IoT threat detection.

Table 3.2.2.1: Features Comparison for Set 1 and Set 2

No.	Feature	Set 1 Features	Set 2 Features
1	ts		
2	src_ip		
3	src_port		
4	dst_ip		
5	dst_port		
6	proto	✓	✓
7	service	✓	✓
8	duration	✓	✓
9	src_bytes	✓	✓
10	dst_bytes	✓	✓
11	conn_state	✓	✓
12	missed_bytes	✓	✓
13	src_pkts	✓	✓
14	src_ip_bytes	✓	✓

15	dst_pkts	✓	✓
16	dst_ip_bytes	✓	✓
17	dns_query	✓	
18	dns_qclass	✓	
19	dns_qtype	✓	
20	dns_rcode	✓	
21	dns_AA	✓	
22	dns_RD	✓	
23	dns_RA	✓	
24	dns_rejected	✓	
25	ssl_version	✓	
26	ssl_cipher	✓	
27	ssl_resumed	✓	
28	ssl_established	✓	
29	ssl_subject	✓	
30	ssl_issuer	✓	
31	http_trans_depth	✓	
32	http_method	✓	
33	http_uri	✓	
34	http_version	✓	
35	http_request_body_len	✓	
36	http_response_body_len	✓	
37	http_status_code	✓	
38	http_user_agent	✓	
39	http_orig_mime_types	✓	
40	http_resp_mime_types	✓	
41	weird_name	✓	
42	weird_addl	✓	
43	weird_notice	✓	

Encode Categorical Labels

The dataset contains 10 distinct classes of categorical labels, each representing different categories or classifications within the data. To effectively utilize this categorical information for analysis or model training, it is essential to convert these labels into a numerical format. This transformation enables computational algorithms to process and interpret the categorical information, which is crucial for performing classification tasks. By converting categorical labels into numerical representations, we ensure compatibility with machine learning algorithms, thereby enabling more accurate predictions and insights to be extracted from the dataset. To achieve this transformation, tools such as Label Encoder from libraries like scikit-learn in Python are employed. This process ensures that the categorical labels are appropriately encoded into numerical values, facilitating the subsequent analysis and modelling steps.

Table 3.2.2.2: Numerical Representations of Attack Classes

Attack Type	Numerical representation
Backdoor	0
Ddos	1
Dos	2
Injection	3
Mitm	4
Normal	5
Password	6
Ransomware	7
Scanning	8
Xss	9

Forming Sentences using Dataset Records

In the next stage of preprocessing, we proceed with forming coherent sentences from the remaining network feature data. This involves concatenating all relevant features of a record together and separating them with white space. By doing so, we consolidate the diverse aspects

of the network data into a unified sentence-like structure. This approach simplifies the information for the model to process and comprehend effectively. By presenting the data in a structured and cohesive manner, we enhance the model's ability to extract meaningful patterns and insights from the dataset. This step plays a crucial role in preparing the data for subsequent analysis and model training, facilitating more accurate and reliable predictions of network threats.

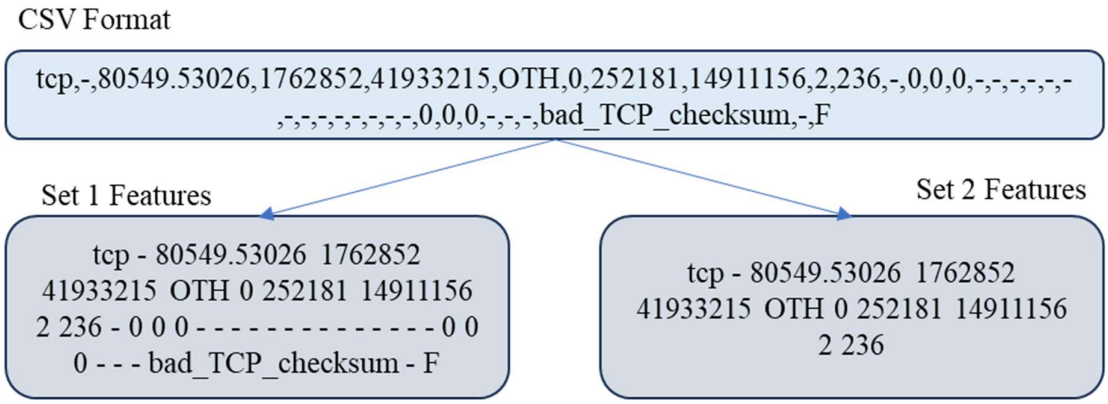


Figure 3.2.2.1: Visualization of Preprocessing

Partitioning Datasets

DistilBert Model

For the DistilBERT-based classification model, the dataset is partitioned into three distinct sets: the training set, validation set, and testing set. This partitioning is essential to facilitate the model's training with cross validation, evaluation, and testing processes, ensuring robust performance and generalization.

SetFit Model

The dataset is partitioned into training set, validation set, and testing set also for the training process. On the other hand, the SetFit model, requiring only a small number of records for training, follows a different approach. To optimize accuracy, various numbers of shots are attempted, specifically 8, 16, 32 and 64 shots. The remaining records not used for shots are allocated as testing data for the SetFit model.

By automating these preprocessing steps, we streamline the dataset and create a more refined input for the subsequent stages of the modeling process. This systematic approach ensures

efficient utilization of data and enhances the model's performance across different architectures and training methodologies.

3.2.3 Tokenizer

DistilBert Model

The textual data obtained from the previous preprocessing stage cannot be directly fed into the text classification model. To make the data compatible with the model, an additional step is required, which involves tokenization. Tokenization involves splitting the text into smaller units called WordPieces. This process allows the text to be converted into embeddings, which are numerical representations that the model can process.

In addition to tokenize the text, special classification tokens are added to the sentences to provide contextual information. These tokens include [CLS] for sequence classification and [SEP] for separating paragraphs. By including these classification tokens, the model can better understand the structure and context of the text.

For example, the record of

"[CLS] 1554287870, 57546, 15600, udp, -, 0. 0, 0, 0, 0, S0, 0, 1, 63, 0, 0, -, 0, 0, 0, -, -, -, -, -, -, - [SEP]"

is converted into:

([101, 14168, 2575, 22407, 2575, 2581, 21619, 1010, 4583, 26187,

2575, 1010, 3770, 1010, 22975, 2361, 1010, 8299, 1010, 1014,

1012, 5890, 24087, 2692, 2683, 1010, 14168, 1010, 4891, 1010,

16420, 1010, 1014, 1010, 1019, 1010, 18034, 1010, 1018, 1010,

10630, 2692, 1010, 1011, 1010, 1014, 1010, 1014, 1010, 1014,

1010, 1011, 1010, 1011, 1010, 1011, 1010, 1011, 1010, 1011,

1010, 1011, 1010, 102])

Furthermore, the tokenizer assigns unique integer IDs to each token and generates an attention mask. The attention mask is a binary mask that indicates which tokens should be attended to

and which ones can be ignored during processing. This information, along with the tokenized text, is then converted into embeddings, which are numerical representations of the tokens that capture their meaning and relationships.

Ultimately, the tokenized text, integer IDs, and attention mask are ready to be inputted into the DistilBERT model for further processing and classification. By following this tokenization process, we ensure that the textual data is appropriately prepared for effective analysis and classification by the model.

SetFit Model

In the case of the SetFit model, there is no need to manually create a tokenizer as it is already integrated within the model's Sentence Transformer module. The Sentence Transformer library provides a comprehensive set of functionalities, including a tokenizer that can automatically tokenize and embed the sentences before the sentences is inputted to the Transformer block.

When the model is running, it automatically processes the sentences and generates embeddings for each sentence using the pretrained tokenizer. This eliminates the need for manual tokenization and embedding steps, as the tokenizer handles these tasks seamlessly within the SetFit model.

By utilizing the capabilities of the Sentence Transformer library and its built-in tokenizer, you can streamline the preprocessing phase and directly obtain sentence embeddings for your input data. This simplifies the overall workflow and ensures that the text is appropriately processed and ready for further analysis or classification using the SetFit model.

3.2.4 DistilBert-based IoT Threats Classification Model

During the preprocessing stage, we encounter challenges with certain features in the dataset, particularly those containing random string values that are difficult to encode. For instance, the "weird_name" feature may contain arbitrary strings like "bad_TCP_checksum," which cannot be easily transformed into numerical representations using traditional methods like Label Encoder. Moreover, Label Encoder lacks the capability to update its dictionary to accommodate new string values, such as with the introduction of a new feature like "protocol." As a result, we seek to explore the potential of language models in the field of IoT threat classification.

DistilBert is a compact and efficient Transformer model developed through the distillation of Bert base. With 40% fewer parameters compared to google-bert/bert-base-uncased, it offers a

lightweight alternative that runs 60% faster while retaining over 95% of BERT's performance on the GLUE language understanding benchmark. This makes DistilBert an attractive option for various natural language processing tasks where speed and resource efficiency are crucial especially on IoT environment.The DistilBert pretrained-model will be utilized to build a multiclass IoT threats classifier. The tokenized embeddings will be inputted into the model for classification. The model will determine which class of threats is the network data records belongs to.

The model will be built using the PyTorch module, which is a popular deep learning framework known for its flexibility and ease of use. PyTorch provides a wide range of tools and functionalities that make it suitable for building and training neural network models. PyTorch offers a dynamic computational graph, allowing for easy model construction and modification. The PyTorch module also offers GPU acceleration, enabling efficient training and inference on compatible hardware. This can significantly speed up the training process, especially for models with large parameter sizes or complex computations.

During the training of the model, the cross-entropy loss function will be utilized as the objective function. The cross-entropy loss is a commonly used metric for multiclass classification tasks because it quantifies the disparity between the predicted class probabilities and the actual class labels. It is particularly suitable when models generate class probabilities as their output. Minimizing the cross-entropy loss allows the model to assign higher probabilities to the correct classes, thereby improving its classification performance.

To prevent overfitting and ensure the model's generalization ability, an early stopping mechanism will be implemented. This mechanism involves monitoring the validation loss after each epoch of training. If the validation loss fails to improve or starts to increase consistently, the training process can be stopped early to prevent the model from overfitting to the training data.

3.2.5 SetFit-based Few-Shot Learning IoT Threats Classification Model

To enhance the original framework, a few-shot learning technique will be explored. This technique aims to improve the model's performance when dealing with limited training data. The SetFit architecture, an efficient few-shot learning approach that leverages the Sentence Transformer architecture, specifically tailored for IoT threat classification will be explored. This innovative framework aims to enhance the model's ability to generalize and classify IoT threats accurately, even with limited labelled data. By utilizing the advanced capabilities of the

Sentence Transformer architecture, SetFit offers a promising solution to address the challenges associated with traditional classification methods in the context of IoT security.

Sentence transformers are a type of model in natural language processing that aims to encode variable length sentences into fixed size numeric vectors in a way that captures the overall semantic meaning and intent of the text. This differs from traditional word embeddings which only embed individual tokens. By fine-tuning powerful pretrained language models like MPNet as encoders, sentence transformers are able to analyse the contextual relationships between words in a sentence to understand the concepts being conveyed. These encoders are then typically combined with pooling operations to condense the final hidden states into a single representative vector for each unique text. The end goal is to be able to measure similarity between sentences based on the cosine distance between their embedding vectors, rather than just relying on surface form word overlap. This allows applications in areas like semantic search, clustering texts by topic, and ranking sentences according to semantic relevance.

SetFit leverages the capabilities of Sentence Transformer to generate dense embeddings based on pairs of sentences. In the Sentence Transformer fine-tuning stage, contrastive training is employed to create positive and negative pairs using a limited amount of labelled textual network data. The Sentence Transformer model is then trained on these pairs, generating dense vectors for each example.

In the classification head training stage, the classification head is trained using the encoded embeddings. This step ensures that the model can map the dense vectors to the corresponding classes or labels. During inference, unseen examples are passed through the fine-tuned Sentence Transformer, which generates embeddings. These embeddings are then fed to the classification head for classification.

The SetFit framework enables few-shot training for the IoT threats classification model. It requires less training data compared to the DistilBERT text classification model mentioned earlier. Additionally, SetFit reduces the training time, making it more efficient for training with limited labelled data.

By leveraging Sentence Transformer's ability to generate dense embeddings and combining it with contrastive training and a classification head, SetFit demonstrates improved performance in few-shot learning scenarios. This framework enables efficient and accurate classification of IoT threats, making it a valuable approach for threat detection and mitigation.

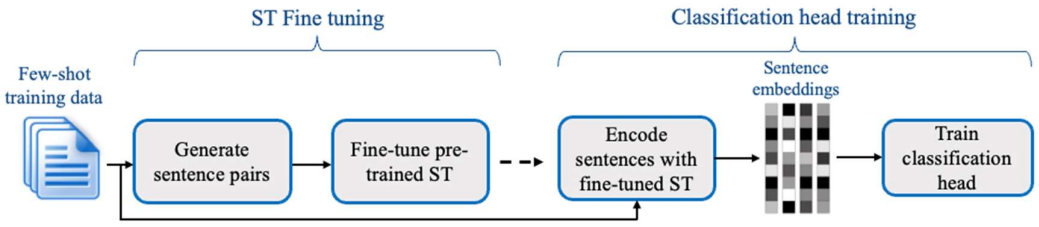


Figure 3.2.5.1: SetFit Framework [18]

SetFit is a library that facilitates the implementation of the framework without the need for extensive model customization. The focus of the work with this model is to fine-tune it by identifying the best parameters and shot configurations. The trainer class provided by the authors is utilized to train the entire SetFit model, automating the training and testing processes. Since SetFit is a few-shot learning method, the model is trained with different numbers of shots. Shots refer to the labeled data provided for training the model in each class.

A higher number of shots generally leads to higher accuracy. However, the objective is to find the optimal number of shots that can achieve high performance without requiring a significant increase in the amount of labeled data. In this context, basic shots being explored are 8, 16, and 32. For instance, 8 shots mean that only 8 records from the dataset are used to generate the sentence pairs and vice versa.

By experimenting with different shot configurations, the goal is to determine the appropriate balance between accuracy and the amount of labeled data required. The aim is to identify the shot configuration that provides high performance without a considerable increase in the number of labeled data instances. It is recommended to have a minimum of 8 shots to ensure sufficient data for the model's training.

Besides that, the number of iterations is also an important parameter to consider in the SetFit framework. It determines the number of pairs that will be generated during the contrastive learning phase. During contrastive learning, positive and negative pairs of sentence embeddings are created to train the Sentence Transformer model. The model learns to bring the embeddings of positive pairs closer together in the embedding space while pushing the embeddings of negative pairs further apart. The number of iterations defines how many pairs are generated and used for training during this contrastive learning process. Each iteration

typically involves randomly sampling positive and negative pairs from the available labeled data.

The trainer of the model separates the training step into 2 stages which is the fine tuning of the Sentence Transformers and the final classifier. Sentence Transformer is being trained by using the generated sentence pairs in which contain true pairs and false pairs. The Cosine Similarity loss function is being used in the Sentence Transformer training and Adam is chosen as the optimizer. While for the classification head uses Cross Entropy Loss as the loss function for the training of final classifier. While Adam is also being used as the optimizer.

In the SetFit framework, the trainer class divides the training process into two stages: fine-tuning the Sentence Transformers and training the final classifier.

The Cosine Similarity loss function is typically employed in this training stage. It measures the similarity between the embeddings of the positive pairs, aiming to maximize it, while minimizing the similarity between the embeddings of the negative pairs. By optimizing this loss function, the Sentence Transformers learn to generate embeddings that capture the semantic relationships between the sentences. The Adam optimizer is chosen for fine-tuning the Sentence Transformers.

In the second stage, the final classifier is trained using the embeddings generated by the Sentence Transformers. Cross Entropy Loss is commonly used as the loss function for training the final classifier. The cross-entropy loss measures the dissimilarity between the predicted class probabilities and the true class labels. The objective is to minimize this dissimilarity, as it corresponds to improving the accuracy of the classification task. Similarly, in the fine-tuning stage, the Adam optimizer is utilized for training the final classifier.

3.2.6 Model Evaluation

In the model evaluation phase, we will comprehensively assess the performance of all four models based on key metrics including accuracy, precision, recall, and F1-score. These metrics serve as reliable indicators of the models' effectiveness in accurately classifying IoT threats.

Our analysis will delve into the strengths and limitations of each model, providing valuable insights into their respective capabilities and areas for improvement. By examining the performance of each model across multiple metrics, we can gain a holistic understanding of their overall efficacy in threat detection. Moreover, we plan to implement a KNN (K-Nearest

Neighbors) classifier alongside our developed model. This will allow us to assess the performance of our model relative to a widely-used baseline classifier in the field of machine learning.

Stage 2: Implementing model on unseen threats

During this stage, we are actively generating realistic network attacks to assess the performance of the deep learning models. This marks a pioneering effort in research, as it represents the first instance of implementing such a step and investigating the feasibility of the model in real-world implementation.

3.2.7 Designing IoT Network Testbed

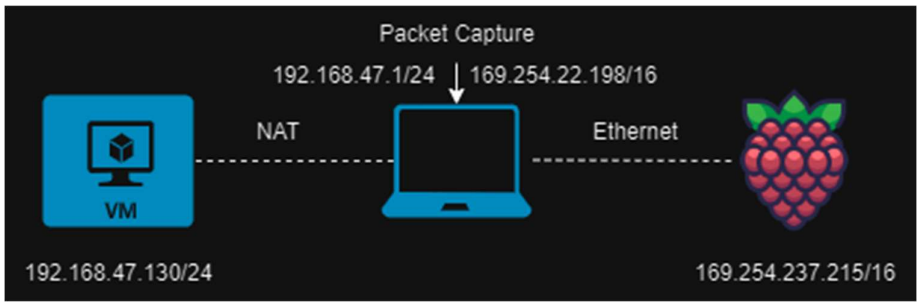


Figure 3.2.7.1: IoT Network Testbed

Our network architecture is designed to simulate a realistic IoT environment where a Raspberry Pi serves as the IoT device hosting a vulnerable web application, the DVWA (Damn Vulnerable Web Application), susceptible to various cyber attacks. The architecture also includes an attacker node, represented by a Kali Linux virtual machine, responsible for launching simulated attacks, and a packet collector node, hosted on a separate computer, to monitor and analyse network traffic.

Components:

Attacker (Kali VM):

The Kali Linux virtual machine is configured to function as the attacker node in our network. It hosts a suite of penetration testing tools and is responsible for launching simulated attacks against the IoT device.

This node is used to initiate various attack scenarios, including DDoS attacks, SQL injection, and cross-site scripting, to assess the vulnerability of the IoT device.

Victim (Raspberry Pi with DVWA):

The Raspberry Pi serves as the victim device in our network, emulating a typical IoT device running a vulnerable web application.

It hosts the Damn Vulnerable Web Application (DVWA), a deliberately insecure web application used for testing purposes.
The Raspberry Pi is physically connected to the network and is accessible from the attacker node for exploitation.

Packet Collector (Host Computer):

Since the Kali VM is hosted in the computer and is connected through NAT with the host computer. The packet collector is hosted on the computer and is responsible for capturing and analysing network traffic generated during the attack simulations.
It is equipped with network monitoring tools such as Wireshark to capture packets traversing the network.

Connectivity:
The host computer and the victim device (Raspberry Pi with DVWA) are connected to the same local area network (LAN) through Ethernet.
The attacker Kali VM is connected to the host with NAT selection in VMware therefore it will share the same IP address with the host computer.
Communication between the attacker and victim nodes occurs over the LAN, allowing the attacker to exploit vulnerabilities in the IoT device's web application.

Software Used:
Kali Linux: Used as the operating system for the attacker node, providing a suite of pre-installed penetration testing tools.
Raspbian OS: Installed on the Raspberry Pi to support the DVWA web application.
DVWA (Damn Vulnerable Web Application): Installed on the Raspberry Pi, serving as the target for simulated attacks.
Wireshark: Utilized on the packet collector node to capture and analyze network traffic.
This network architecture facilitates the simulation of realistic attack scenarios in an IoT environment, enabling the evaluation of few-shot learning techniques for threat detection.

3.2.8 Discovering IoT vulnerabilities and Intrusion Planning

In the context of our specialized network testbed, where documentation from ToN-IoT is unavailable, we are tasked with designing and executing network attacks to simulate real-world threats. These attacks are tailored to be unseen, meaning they are not present in the existing dataset and therefore represent novel challenges for our threat detection models.

Port Scanning Attack

One of the attacks we will conduct is a port scanning attack using the Nmap tool. This attack aims to identify open ports on the target device (169.254.237.215) which could potentially serve as entry points for unauthorized access. The command nmap 169.254.237.215 will be executed to scan for open ports and gather information about the target's network configuration.

Tools: Nmap

Command: nmap 169.254.237.215

Password Attack

Another attack in our arsenal is a password attack, which involves attempting to gain unauthorized access to the target device by brute-forcing login credentials. We will utilize the Hydra tool to perform HTTP-GET-FORM and SSH brute force attacks. These attacks involve systematically attempting different username-password combinations until a successful login is achieved.

Tools: Hydra

Hydra SSH brute force

hydra 169.254.237.215 -l pi -P /home/kali/passwordList.txt ssh

XSS Attack

Additionally, we will execute a cross-site scripting (XSS) attack using the xsser tool. This attack targets web applications running on the target device (at the URL "http://169.254.237.215/vulnerabilities/xss_r/?name=XSS#") and exploits vulnerabilities to inject malicious scripts.

Tools: xsser

Command: sudo xsser --url "http://169.254.237.215/vulnerabilities/xss_r/?name=XSS#" --cookie="PHPSESSID=b6i30vnrgtpfqdn57mp02so790; security=low" –auto

*cookie value must be changed according to session

3.2.9 Perform attack and capture the attack network flow

The planned attack is currently underway, and we are actively capturing the network flow using Wireshark. Wireshark, a powerful network protocol analyser, allows us to monitor and capture network traffic in real time, providing detailed insights into the data packets traversing the network during the attack.

As the attack progresses, Wireshark continuously captures and records various network activities, including packet transmissions, protocol interactions, and communication between devices. This captured network flow data will serve as valuable input for our analysis and evaluation of the attack's impact on the network infrastructure. The network flow will be saved in pcap format.

3.2.10 Generating Network Logs with Zeek and Converting to CSV

The Zeek network analysis tool will be utilized to capture and log network traffic data from pcap files generated from the simulated attacks. Subsequently, Python scripts were employed to convert the generated logs into a structured CSV format suitable for further analysis and model inference process.

Generating Network Logs with Zeek

Deployment: Zeek was deployed on Ubuntu Linux VM. The network flow data generated during the simulated attacks are transferred into the VM for Zeek analysis and logging. Zeek passively analyse each packet captured by the Wireshark program and generate comprehensive logs containing detailed information about network connections, protocol activity, and security events. Zeek logs various types of network activity, including connection summaries (conn.log), HTTP requests (http.log), DNS queries (dns.log), SSL/TLS handshakes (ssl.log), and more, providing a holistic view of the network traffic.

Converting Zeek Logs to CSV Using Python Scripts

Before converting the Zeek logs to CSV, we preprocess the logs to extract relevant fields and filter out unnecessary information based on the requirements of the model. Since we are unable to reproduce the way ToN-IoT combining the logs, we only extract the logs information from conn.log files which consists of all connection information.

We developed custom Python scripts to parse the Zeek logs and convert them into structured CSV files. These scripts utilize libraries such as Pandas for data manipulation and for file I/O operations.The script reads the conn.log file into a Pandas DataFrame, selectively extracting pertinent columns such as timestamps, IP addresses, port numbers, protocol types, connection durations, and packet counts. Subsequently, the extracted data is transformed into a new DataFrame, tailored to include only the essential information relevant to network connections. Finally, the script exports this refined DataFrame into a CSV file, allowing for seamless integration with various data analysis tools and workflows. By automating the conversion process, the script facilitates efficient data preprocessing, The table below shows the features that is extracted from the conn.log, it is exact same with the set 2 feature.

Table 3.2.10.1: Features Visualization for Generated Dataset

No.	Features
1.	ts
2.	src_ip
3.	src_port
4.	dst_ip
5.	dst_port
6.	proto
7.	service
8.	duration
9.	src_bytes
10.	dst_bytes
11.	conn_state
12.	missed_bytes
13.	src_pkts
14.	src_ip_bytes
15.	dst_pkts
16.	dst_ip_bytes

Dataset Filtering and Labelling

The csv files generated will then be analysed and filtered out to remove noises from the normal network flow. Only the malicious network flow will be remained in the dataset.

After that, the ground truth labels will be assigned to each record in the CSV file. These labels correspond to the true classification or category of each data instance, providing a reference point for training and evaluating machine learning models.

Combining csv into datasets

We will produce 3 different datasets for evaluation to represent different situation.

1) ToN-IoT testset

   20% of the ToN-IoT data will be randomly sampled out for testing.

2) Unseen Logs dataset

   50 samples for each class from the self-generated unseen logs will be randomly extracted to form the unseen logs dataset for testing.

3) Combined dataset

   50 samples for each class will be sampled out from ToN-IoT and 5 samples for each class in the unseen log dataset will be combined to form the combined dataset.

3.2.11 Evaluation of Model Performance on Generated Data

To assess the effectiveness of the trained model in detecting unseen IoT threats, we conducted an evaluation of its performance on the generated data. We utilized the trained models on the set 2 features to make inferences on the network log data previously converted into CSV format. The model analysed each network connection represented in the CSV file, leveraging its learned representations of normal and anomalous network behaviour to classify connections as benign or malicious. Subsequently, we evaluated the model's predictions against ground truth labels. By comparing the model's predictions with actual observations, we gained insights into its ability to effectively identify and classify security threats in the IoT environment. This evaluation process serves as a critical step in validating the model's performance and assessing its readiness for real-world deployment in enhancing network security and threat detection capabilities.

3.2.12 Performing retraining of model with combined dataset (Transfer Learning)

In our exploration of model capabilities, a significant aspect we delved into was the potential of model adaptability towards unseen threats. Transfer learning, a technique widely employed in machine learning, involves leveraging knowledge gained from training on one task to improve performance on a different but related task. By fine-tuning the previously trained model on the unseen self-generated dataset, we aimed to evaluate its adaptability and

effectiveness in detecting threats across different IoT environments. Through this process, we sought to uncover insights into the model's ability to generalize and transfer learned knowledge, ultimately enhancing its utility in addressing diverse cybersecurity challenges in IoT ecosystems. This investigation into transfer learning capabilities represents a crucial step in harnessing the full potential of machine learning models for cybersecurity applications, paving the way for more robust and adaptable threat detection systems.

The DistilBert-based model will undergo retraining using a hybrid dataset comprised of samples from the ToN-IoT training set and training samples from the unseen dataset. The ToN-IoT training set will be randomly subsampled to include 50 samples per class, which will then be combined with manually sampled unseen data containing 5 samples per class to augment the training dataset for the SetFit model. The retraining process follows the standard training methodology outlined earlier, with the exception of reducing the number of training epochs to 5 for both models

Chapter 4 System Model

4.1 System Implementation Modeling

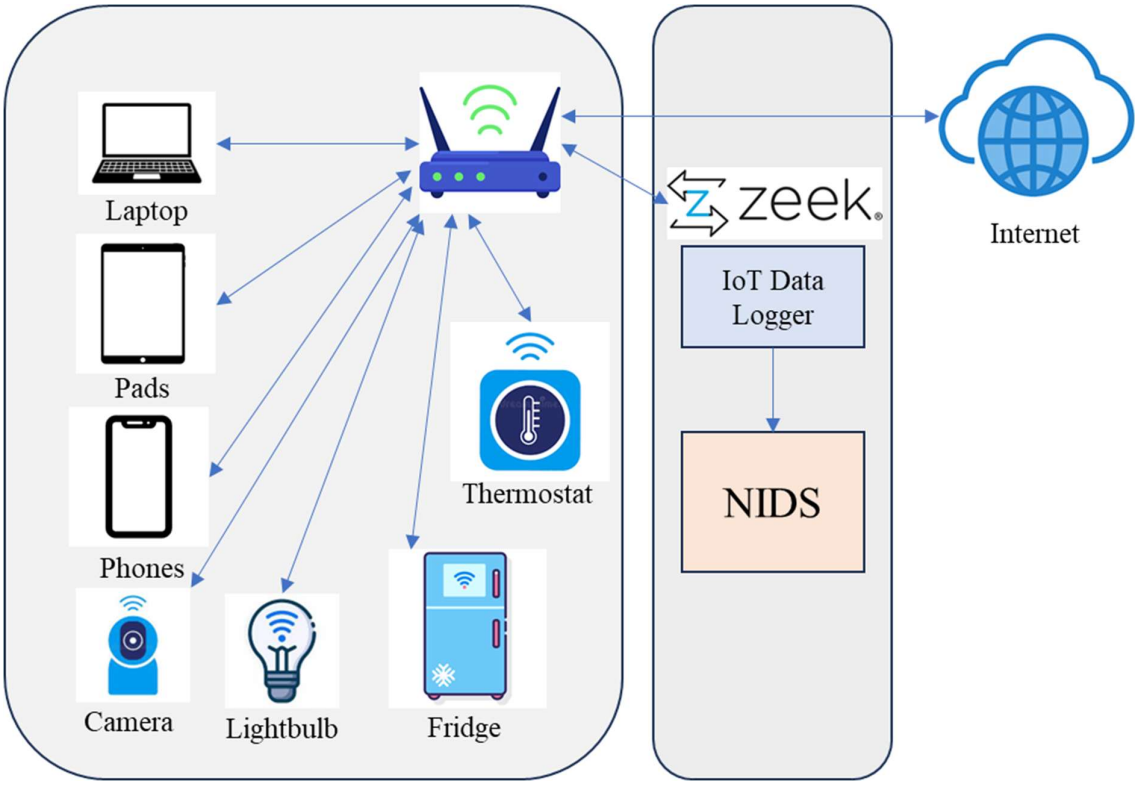


Figure 4.1.1: System Implementation Model

The model that has been created will play a crucial role within the Network Intrusion Detection System, specifically tailored for detecting potential threats within IoT network setups. Its function is to analyze the connection logs received from the Zeek logging server, a platform designed for network traffic analysis. Upon receiving these logs, the model will apply its classification capabilities to assess each connection and determine whether it represents normal network activity or poses a potential security threat. This process enables the system to swiftly identify and respond to any suspicious or malicious behavior occurring within the IoT network environment, thereby enhancing overall network security.

4.2 IoT Threat Detection Model Flow

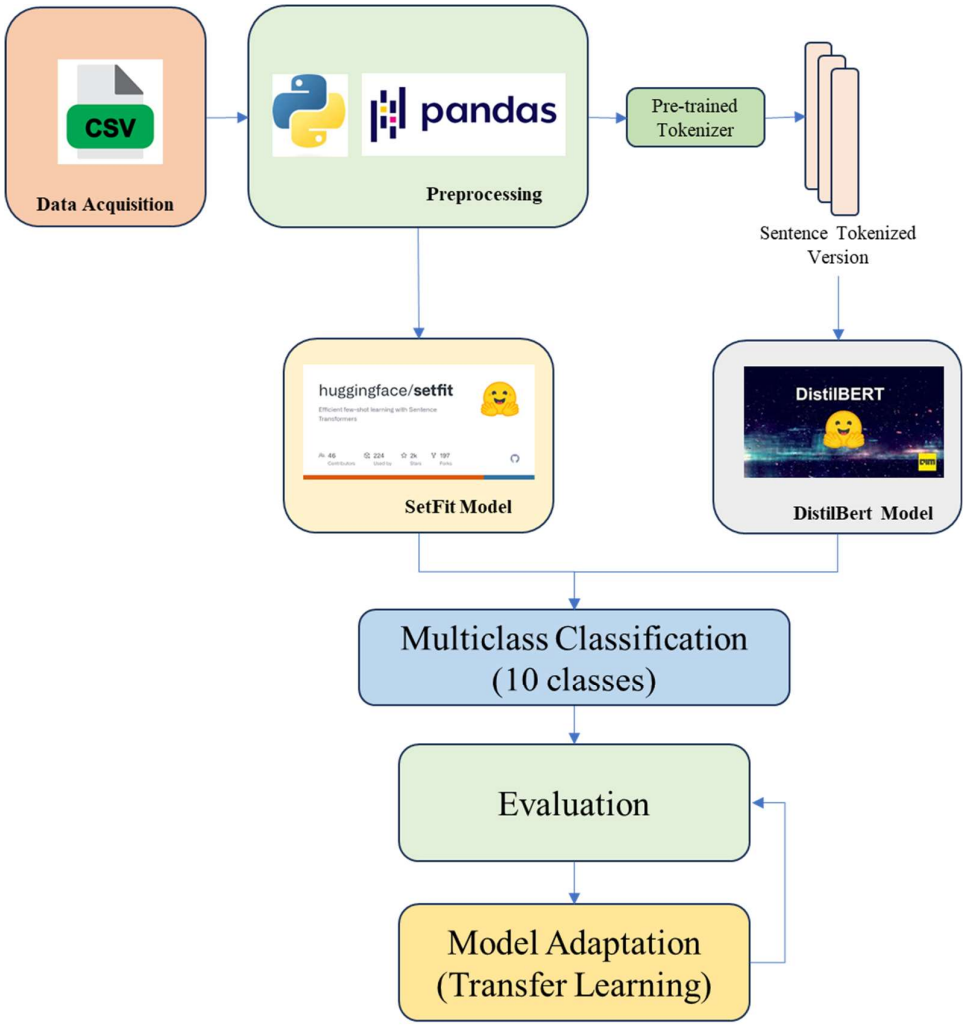


Figure 4.2.1: IoT Threat Detection Model Flow

The acquired network log data undergoes preprocessing to extract relevant features essential for the models. These features are then concatenated into a sentence-like structure. For the DistilBert model, tokenization of the sentences precedes their input into the model.

This research develops two distinct models: SetFit, a few-shot learning model, and DistilBert, which lacks few-shot learning capability. Both models engage in multiclass classification of the log sentences to determine whether they belong to attack classes or represent normal traffic flow.

Following model classification, evaluation ensues to scrutinize model performance. In cases where new, unrecognized attacks emerge, model adaptation is initiated. This adaptation involves employing transfer learning on the existing model using a combined dataset containing the new, unseen log samples and the old samples. Through model retraining, the model efficiently learns the patterns of new threats, enabling it to classify them accurately post-adaptation.

4.3 Models Architecture

4.3.1 DistilBert-based IoT Threat Classification Architecture

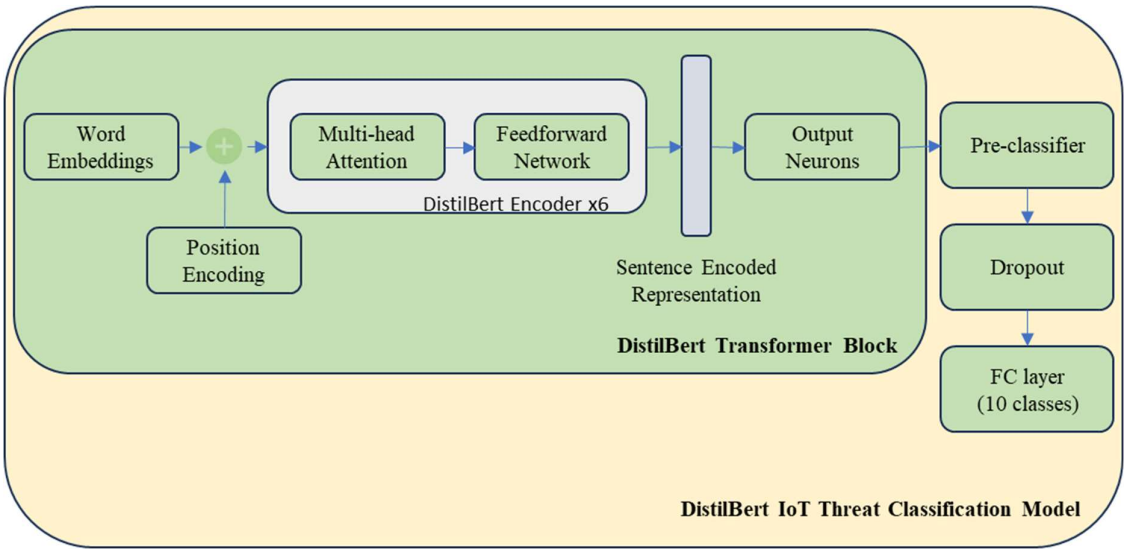


Figure 4.3.1: DistilBert-based IoT Threat Classification Model Architecture

The threat classification model consists of 4 main components which is the DistilBert layer, pre-classifier layer, dropout layer and classifier layer. The DistilBert layer will encode the input sequences into a fixed length vector. The vectors will then be input into the pre-classifier, dropout layer and classifier to get the prediction.

The DistilBert layer consists of 2 main components which are the Embeddings block and the Transformer block. Embeddings block is responsible for handling the input tokenization and embedding processes. It consists of 4 main components which is the word embeddings, position embeddings, a normalization layer and dropout layer. Word embeddings layer will represent every token by a vector space of (30522, 768) where 30522 is the number of unique tokens in

the vocabulary and 768 is the dimensionality of the embedding space. While the position embeddings will represent the position in the sequence by a vector of (512, 716) where 512 is the maximum sequence length of every token and 716 is the dimensionality of the embeddings space. Next, the normalization layer helps to improve the stability and speed up the training process and dropout layer prevents overfitting.

Next, the input is then pass through the DistilBert encoder. It is used to encode the input sequences into fixed-length vectors. There are 4 main components in the DistilBert encoder block which consists of the Multi-Head Self Attention layer, normalization layer, feed-forward network layer, and output normalization layer. The multi-head self-attention layer is a mechanism allows the model to attend to different parts of the input sequence simultaneously, capturing dependencies and relationships between tokens. It consists of linear transformations for query (q), key (k), and value (v) projections, followed by dropout and output linear transformations. The output will then pass to a normalization layer. The feedforward neural network consists of two linear layers with a GELU activation function in between. It transforms the attention outputs to a higher-dimensional space and then back to the original dimensionality. The output is then pass through the normalization layer again before pass to the next layer for classification.

After being process by the DistilBERT Model, the output vector will then be pass to the 3 additional classification layers which is the pre-classifier layer, dropout layer, and classifier layer. The data will be pass through a pre-classifier first before inputted into the dropout layer which randomly drop some activation during training to prevent overfitting and improve generalization ability of the model. Finally, the final classifier will output the predicted class probability.

4.3.2 SetFit-based IoT Threat Classification Architecture

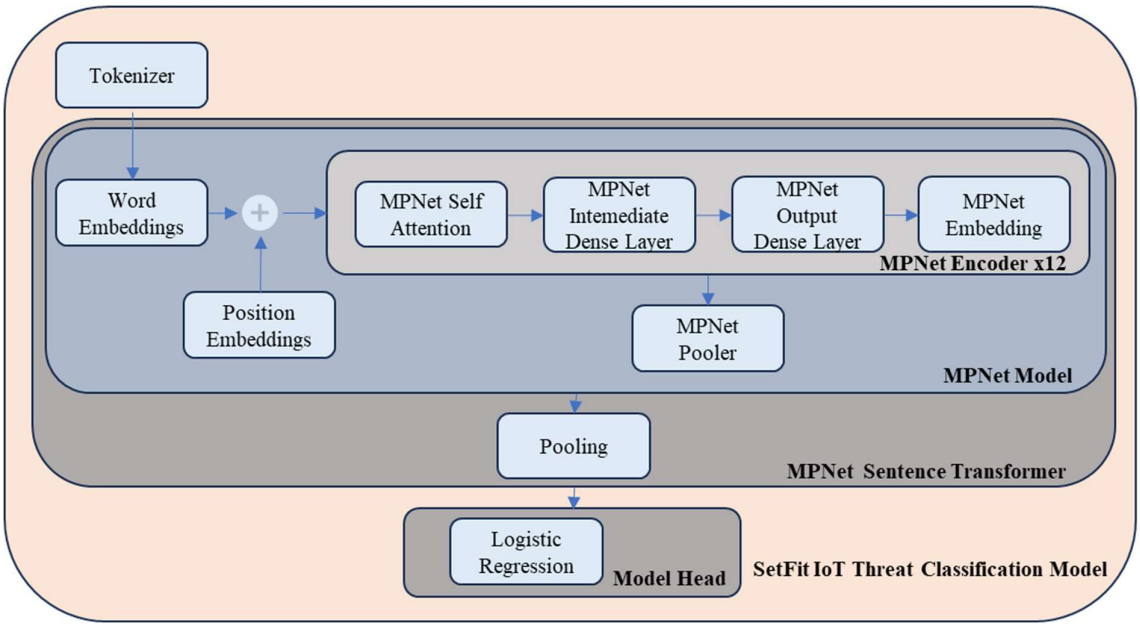


Figure 4.3.2: SetFit-based IoT Threat Classification Model Architecture

The SetFit framework comprises two main layers: the MPNet Sentence Transformer and the classification head.

The model body consists of two components: the Tokenizer and the MPNet Transformer model. The MPNet Sentence Transformer is a transformer-based model that learns contextual representations of the input sentences. It captures the relationships and dependencies between words or tokens, enabling it to generate meaningful dense embeddings. The MPNet Sentence Transformer first tokenizes the input text into individual tokens and maps them to dense vector representations using word embeddings. These embeddings are then processed through multiple layers of Transformer blocks, which incorporate self-attention and feedforward neural networks to capture semantic relationships and contextual information. Additionally, a mean pooling layer is applied to aggregate the token-level representations into a single fixed-length vector.

The classification head is a machine learning model that performs the multiclass classification task. In the SetFit framework, Logistic Regression has been chosen as the classification model. Logistic Regression is a widely used model for binary or multiclass classification problems. It learns a set of weights and biases to map the dense embeddings generated by the Sentence Transformer to the corresponding threat classes. By training the Logistic Regression model on

the encoded embeddings, it becomes capable of accurately classifying the threats based on the learned representations.

These embeddings are then processed through multiple layers of Transformer blocks, which incorporate self-attention and feedforward neural networks to capture semantic relationships and contextual information.

Chapter 5 Experiment

5.1 Hardware Setup

Several hardware had been used to conduct the experiment to implement it on self-generated network attacks.

1) Laptop

Table 5.1.1: Specifications of laptop

Description	**Specifications**
Model	HP Pavilion Laptop 15-cs3137tx
Processor	Intel Core i7-1065G7
Operating System	Windows 11
Graphic	NVIDIA GeForce MX250 4GB DDR3
Memory	16GB DDR4 RAM
Storage	512 GB NVME SSD

2) Raspberry Pi 3 Model B

Table 5.1.2: Specifications of Raspberry Pi 3 Model B

Description	**Specifications**
Model	Raspberry Pi 3 Model B
Processor	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
Operating System	Raspbian Buster
Memory	1GB RAM

3) RJ45 Network Cable

5.2 Software Setup

Software on Laptop

1) VMWare Workstation 17 Player
2) Kali Linux VM
    a. Hydra
    b. Nmap

c. xsser

3) Wireshark
4) Zeek Network Analyzer Tools installed in Ubuntu VM
5) RealVNC Viewer

Software on Raspberry Pi

1) Damm Vulnerable Web Application (DVWA)
2) Docker

5.3 Setting and Configuration

5.3.1 Setting Up Vulnerable IoT Device

The Raspberry Pi 3 Model B will be used as the victim in our network testbed. We can access to the Raspberry Pi interface with RealVNC viewer through the IP address of 169.254.237.215. This setup enables remote interaction with the Raspberry Pi, allowing for monitoring, management, and troubleshooting tasks to be performed seamlessly.

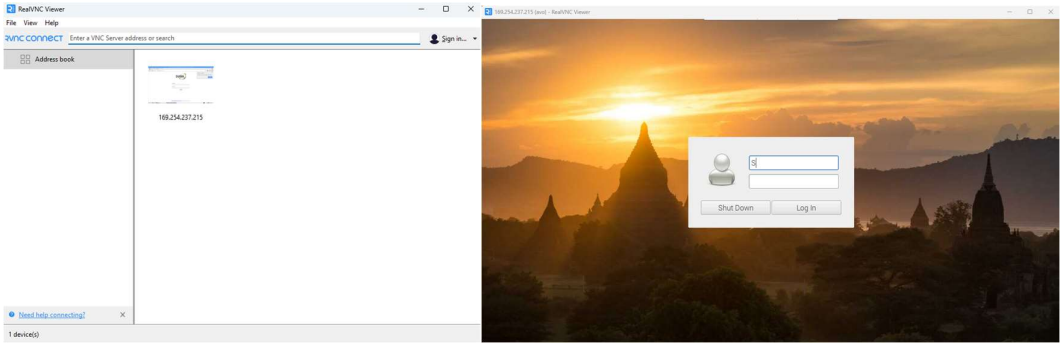


Figure 5.3.1.1: Setting Up Vulnerable IoT Device 1

Initially, Docker will be installed on the Raspberry Pi to simplify the process of hosting the Damn Vulnerable Web Application (DVWA). Subsequently, DVWA will be executed within a Docker container, leveraging the containerization technology provided by Docker. This

approach streamlines the deployment of DVWA on the Raspberry Pi, ensuring efficient resource utilization and ease of management throughout the hosting process.

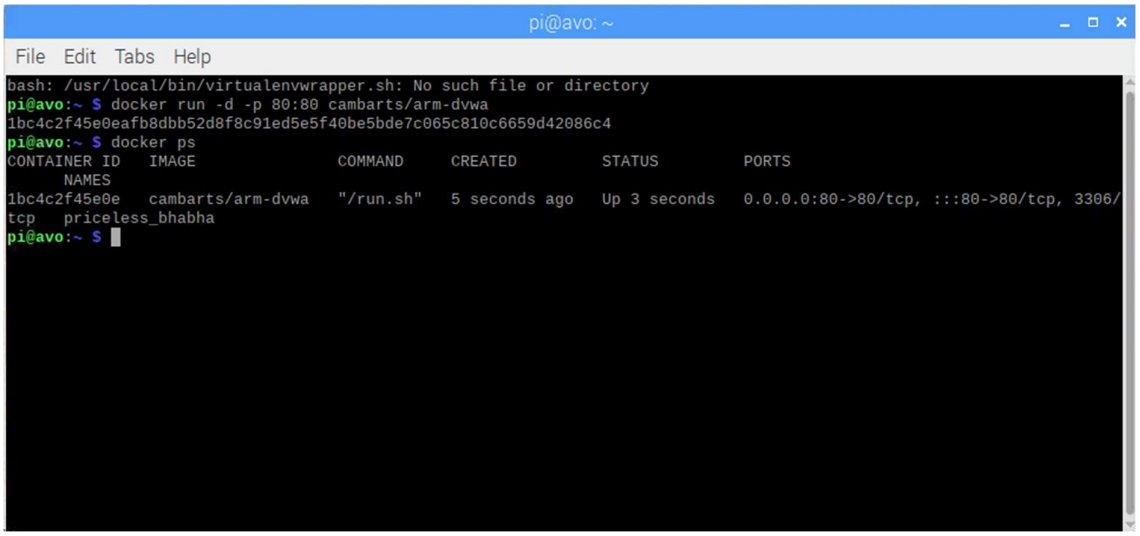


Figure 5.3.1.2: Setting Up Vulnerable IoT Device 2

DVWA web page can be access through localhost IP port 80 through web browser.

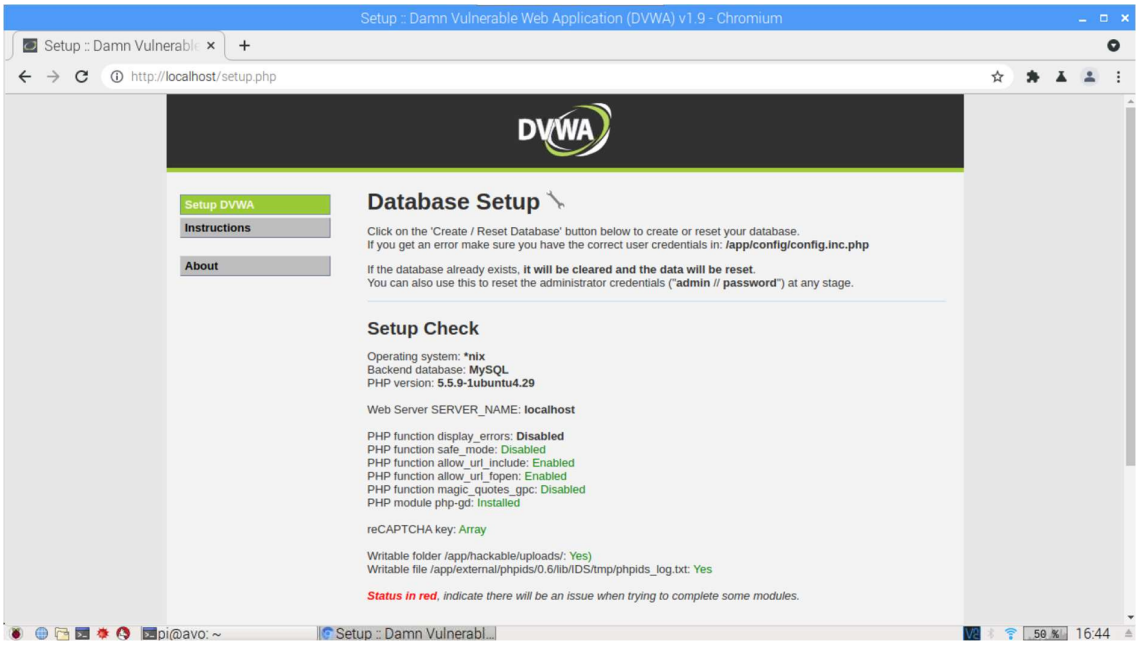


Figure 5.3.1.3: Setting Up Vulnerable IoT Device 3

Next, we will need to create a database for the DVWA.

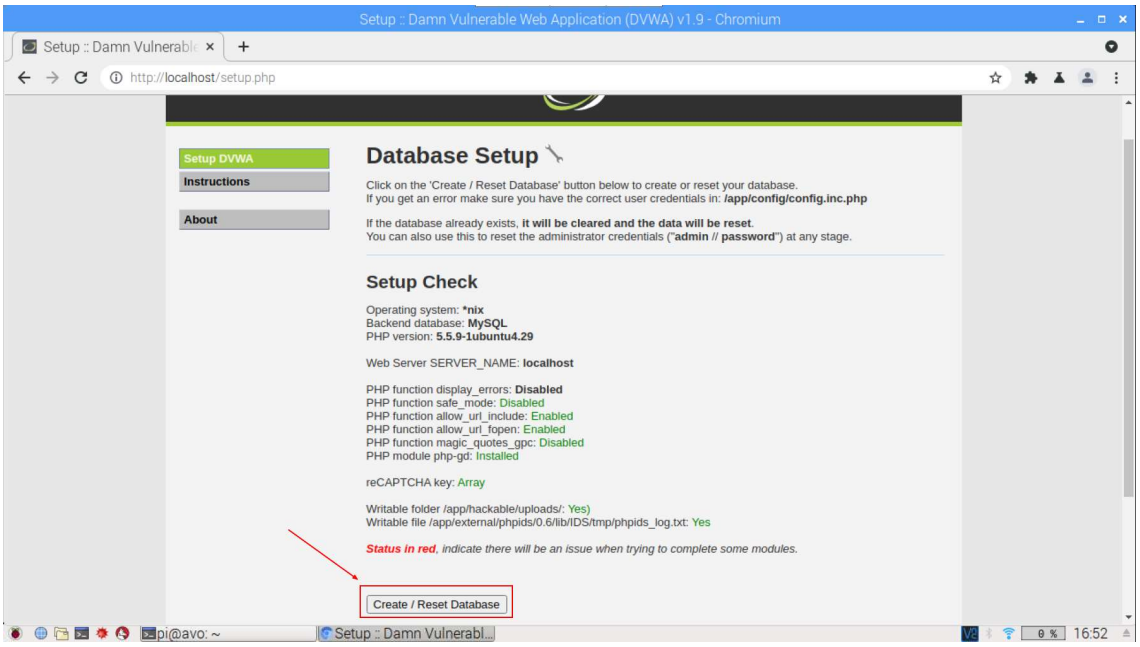


Figure 5.3.1.4: Setting Up Vulnerable IoT Device 4

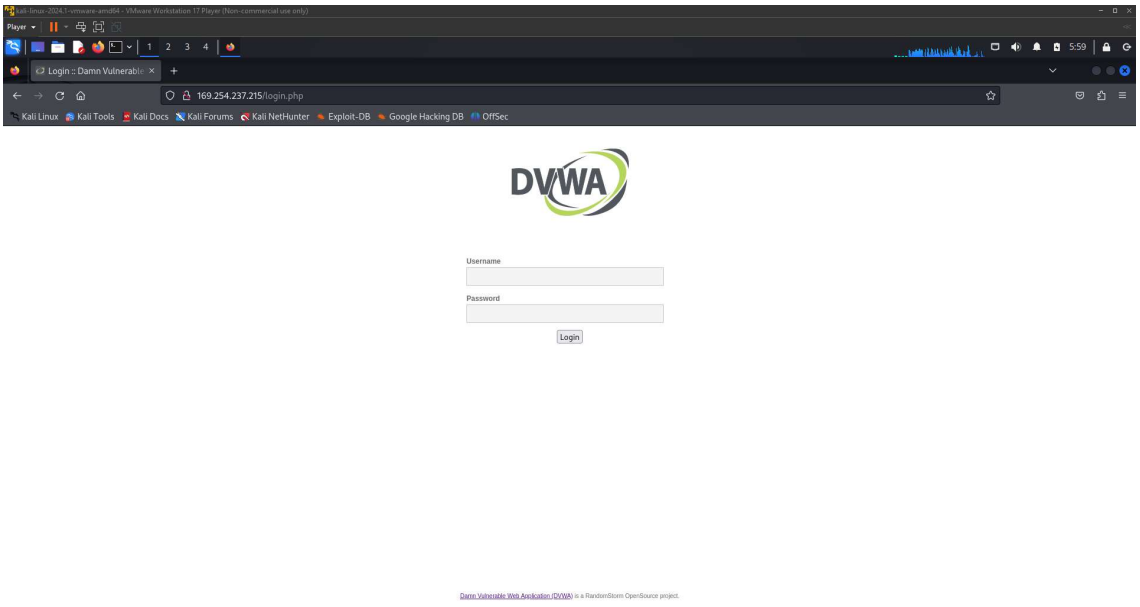


Figure 5.3.1.5: Setting Up Vulnerable IoT Device 5

5.3.2 Setting Up Kali Linux VM

1) Download pre-built VMware Kali Image

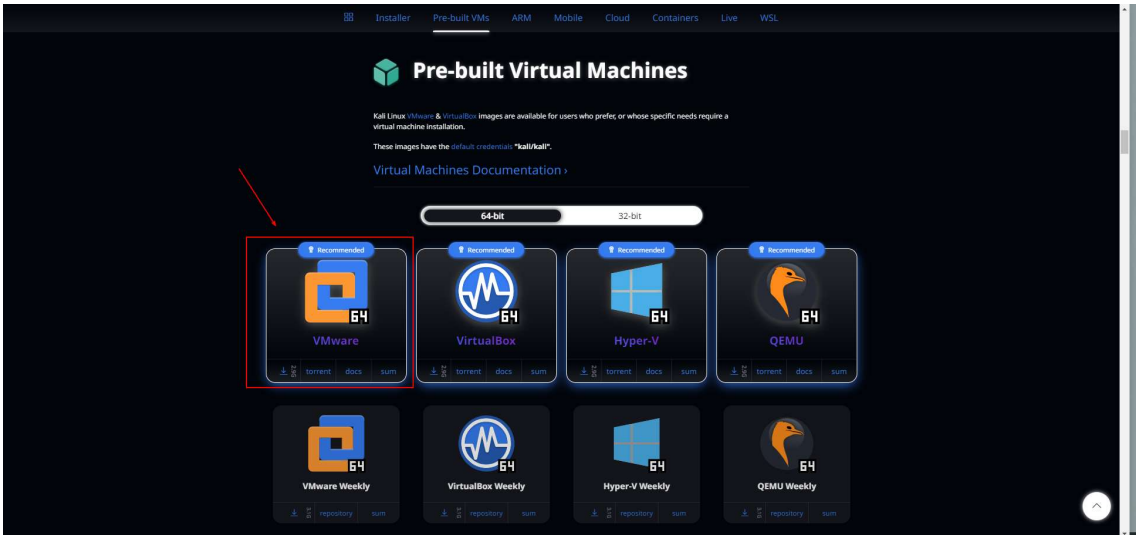


Figure 5.3.2.1: Setting Up Kali Linux VM 1

2) Open the Kali VM downloaded

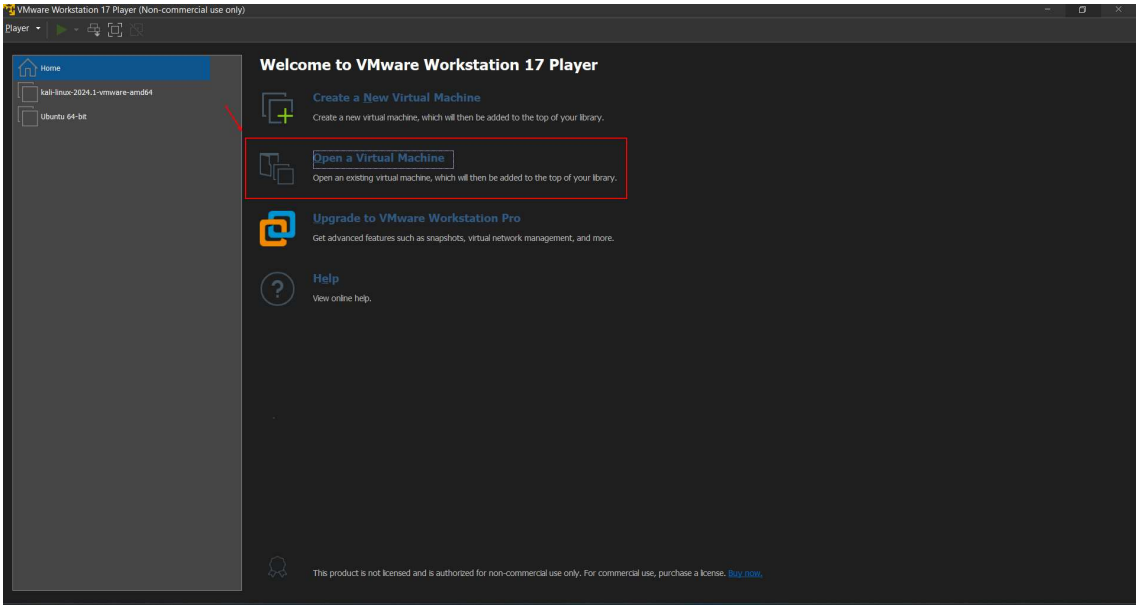


Figure 5.3.2.2: Setting Up Kali Linux VM 2

3) Double click to turn on the VM

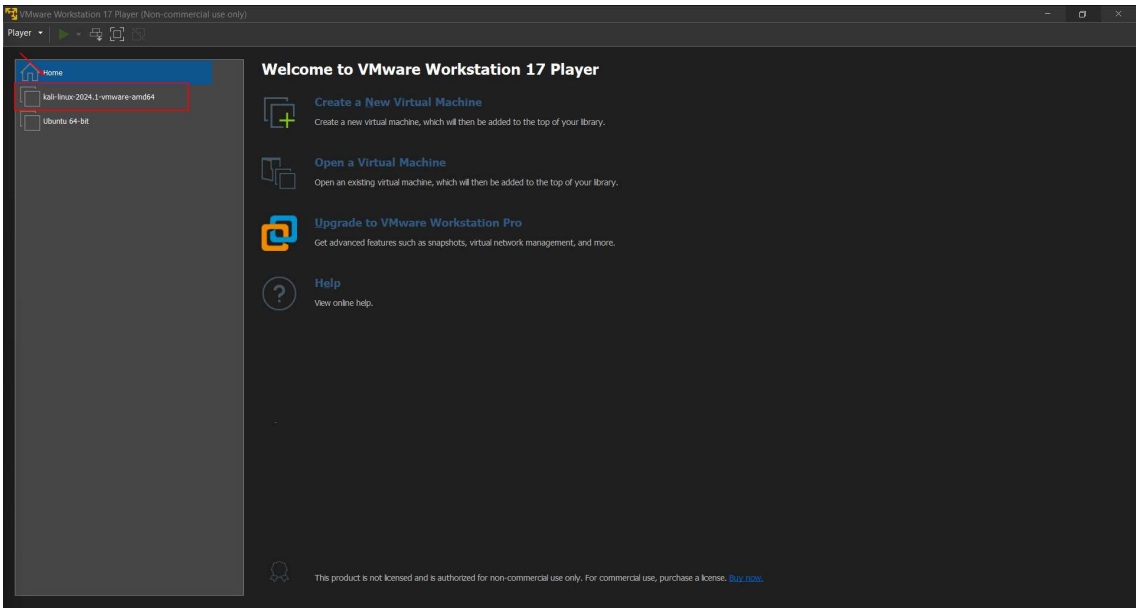


Figure 5.3.2.3: Setting Up Kali Linux VM 3

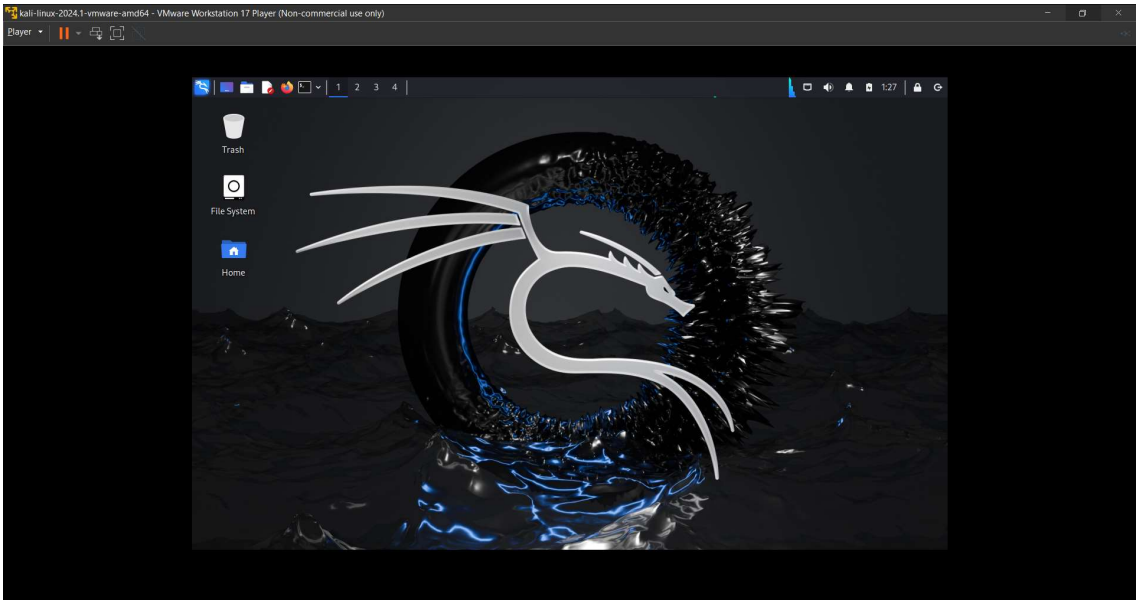


Figure 5.3.2.4: Setting Up Kali Linux VM 4

5.3.3 Setting Up Wireshark

1) Turn on wireshark and sniff the packet on the Ethernet port

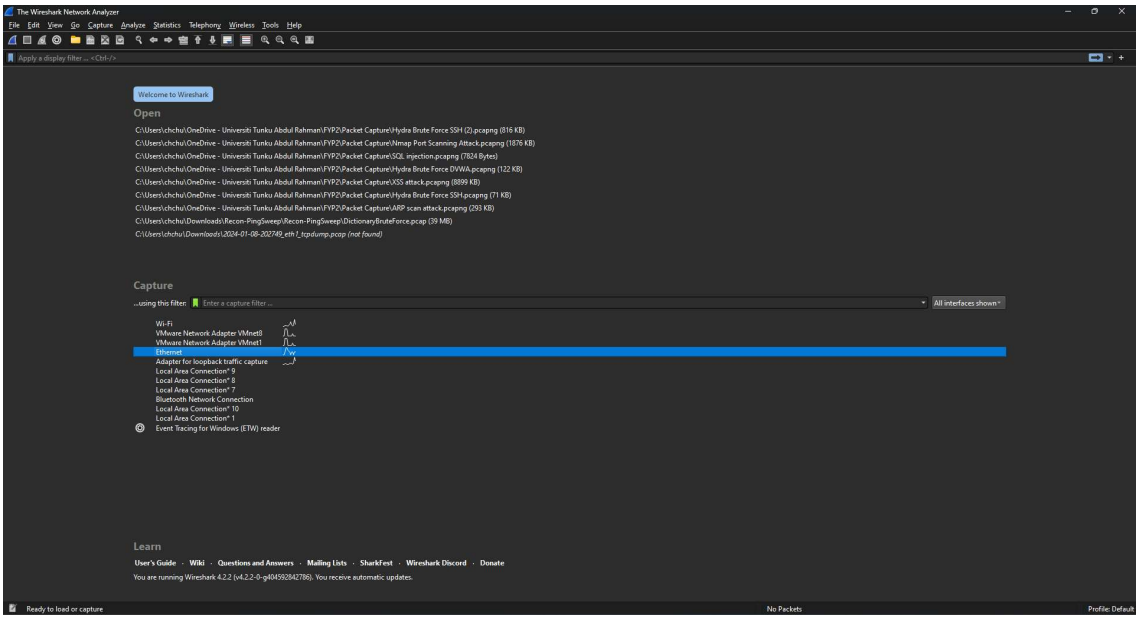


Figure 5.3.3.1: Setting Up Wireshark

5.4 System Operation

5.4.1 Performing Network Attacks & Capturing Network Packets

The Kali Linux virtual machine, equipped with a diverse array of hacking tools, serves as the platform for launching network attacks against the Raspberry Pi. Installed on a laptop, the Kali VM is hosted using VMWare Workstation 17 Player. A bash script has been developed to automate the steps involved in executing the hacking procedures efficiently.

Wireshark, a network protocol analyzer, is employed to capture the flow of network packets. Prior to launching an attack, Wireshark is activated to monitor the packet flow on the Ethernet port of the laptop.

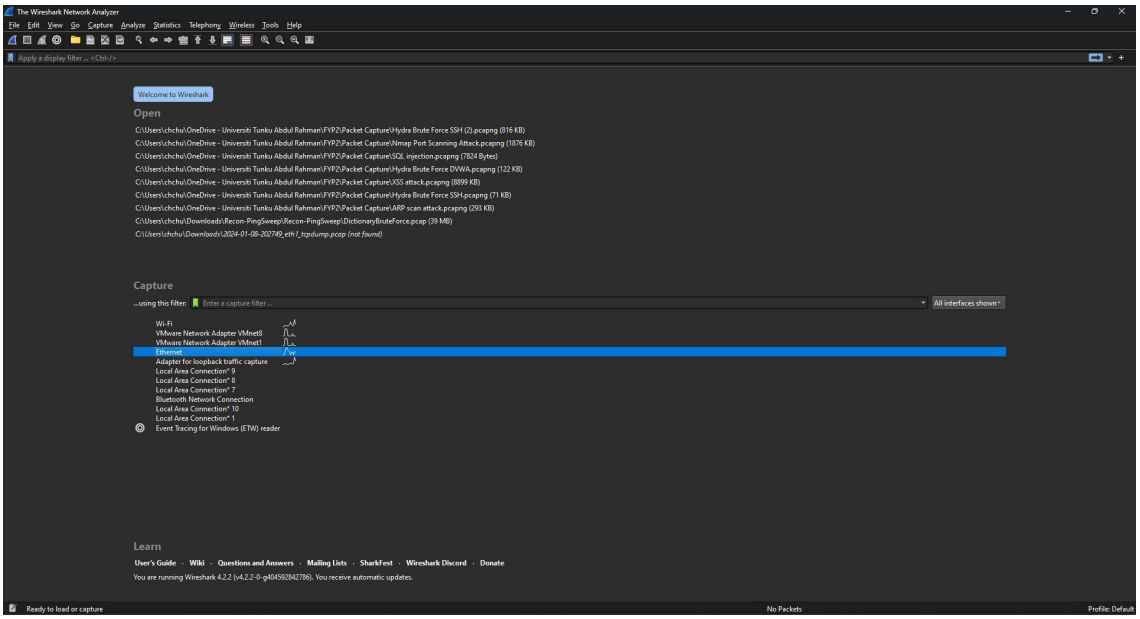


Figure 5.4.1.1: Capturing Attack Network Flow 1

Below consists of the screenshots of the network attacks that are launched to the victim:

1) Port Scanning

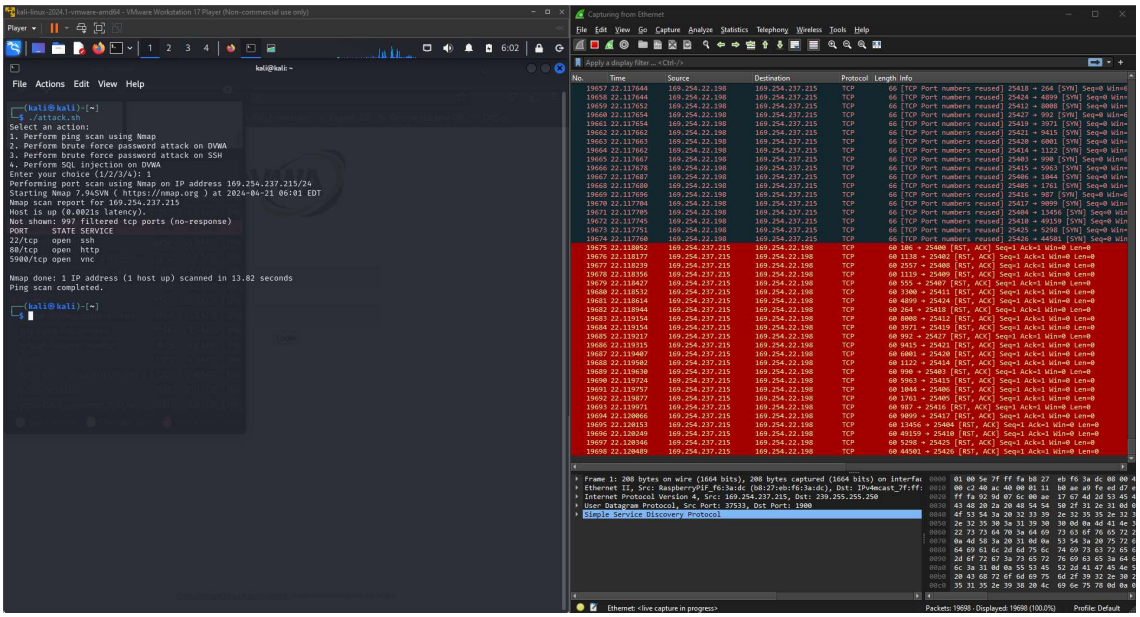


Figure 5.4.1.2: Capturing Attack Network Flow 2

There are 3 services that have its port open which is SSH on port 22 TCP, HTTP on port 80 TCP and VNC on port 5900 TCP.

2) SSH Brute Force

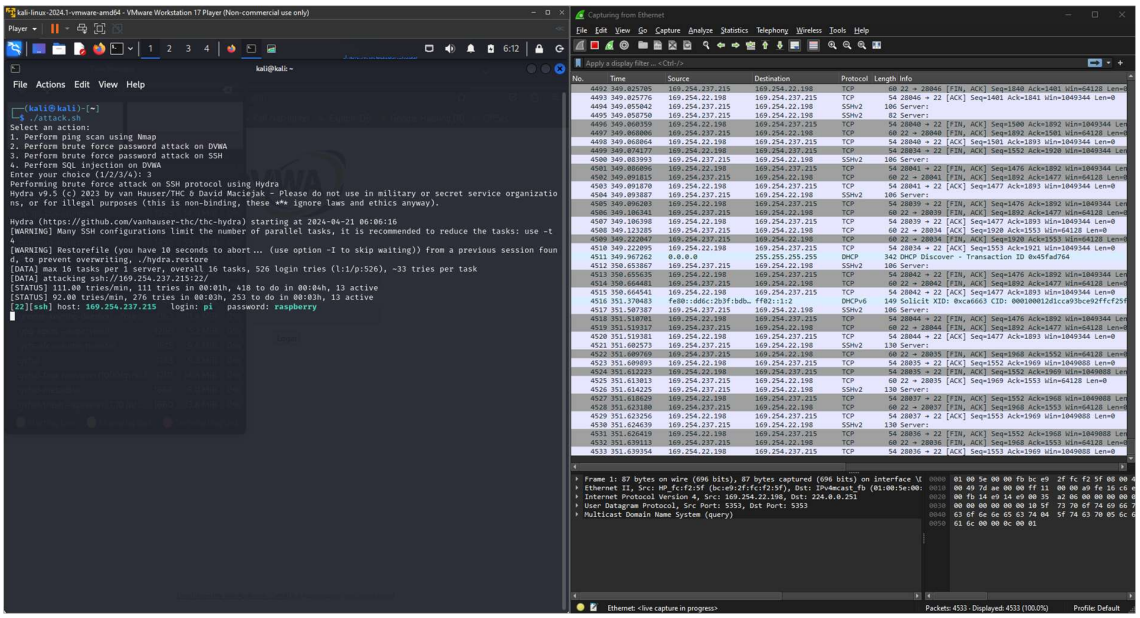


Figure 5.4.1.3: Capturing Attack Network Flow 3

The Hydra successfully brute force the SSH password.

3) Cross-site scripting

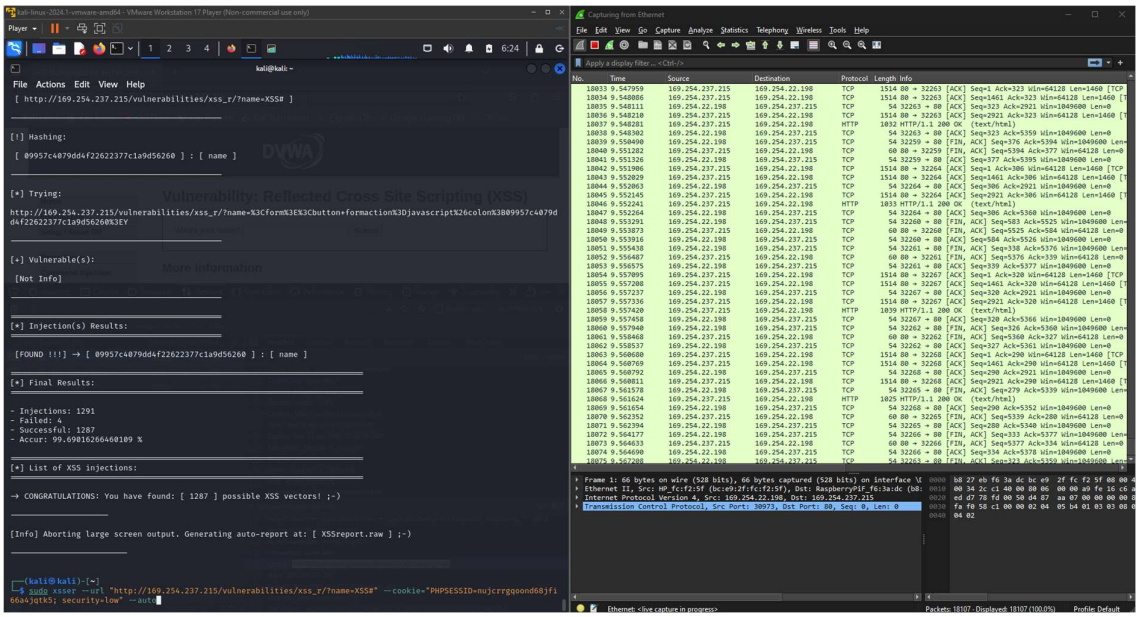


Figure 5.4.1.4: Capturing Attack Network Flow 4

5.4.2 Create Network Logs from Network Flow

Zeek will be used to analyze the attacking network flow and output their relevant network logs.

All the packet capture will be analyzed by Zeek to get the output logs.

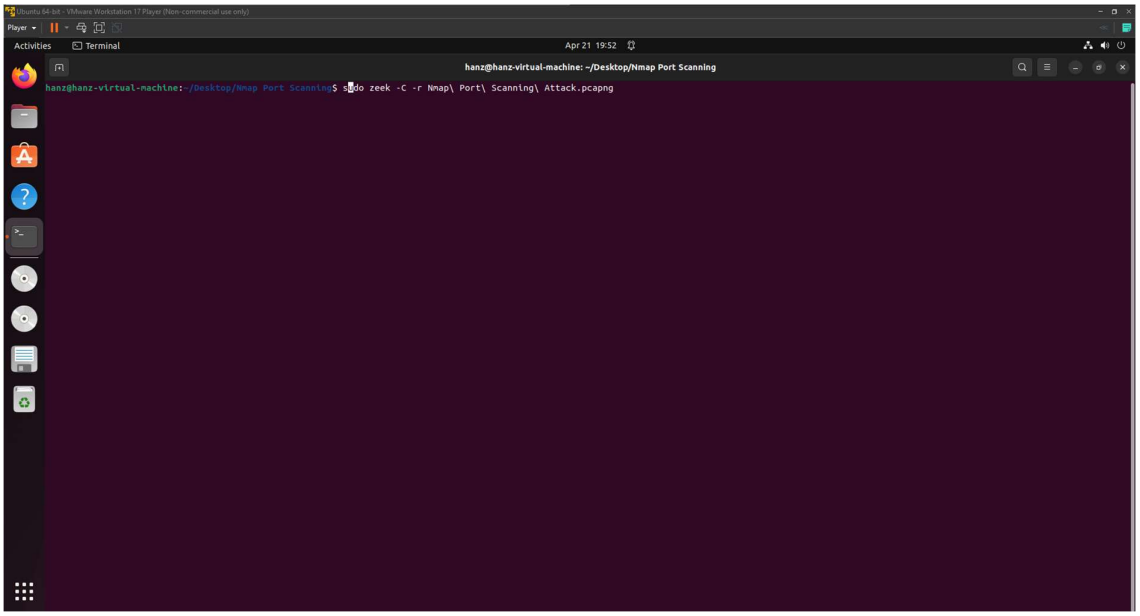


Figure 5.4.2.1: Generating Network Flow Logs 1

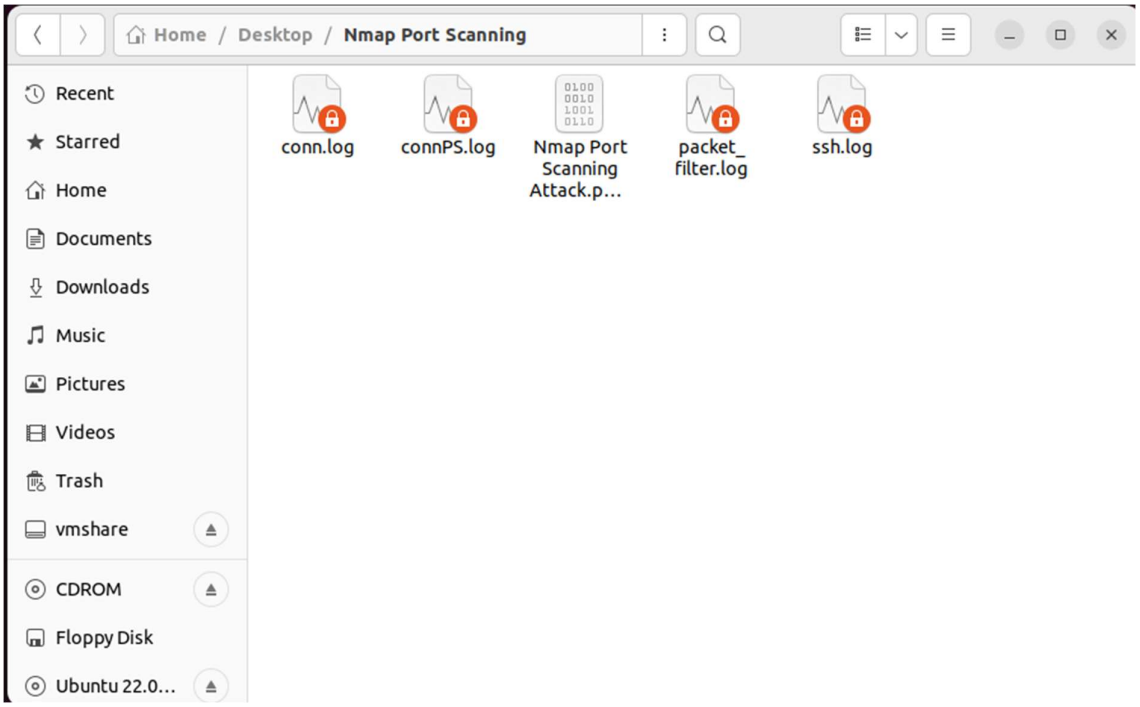


Figure 5.4.2.2: Generating Network Flow Logs 2

5.4.3 Preprocessing Network Logs into CSV

The conn.log generated from Zeek will be extracted out for model classification. The 3 conn.log files from the 3 attacks will be process and the connection features will be extracted

through a Python script. The logs features will then be converted to csv format and labelled with their respective attack types.

Load self generated dataset and convert to csv

```
In [2]: import utils.ConnLogToCSV as a

In [3]: a.connToCSV(file_path="connPS.log", out_path="connPS.csv", label="scanning")
        DataFrame exported to connPS.csv

In [6]: a.connToCSV(file_path="connSSHBF2.log", out_path="connSSHBF2.csv", label="password")
        DataFrame exported to connSSHBF2.csv

In [8]: a.connToCSV(file_path="connXSS.log", out_path="connXSS.csv", label="xss")
        DataFrame exported to connXSS.csv

In [1]: import pandas as pd

In [2]: ps = pd.read_csv('connPS.csv')
        ps
```

Out[2]:

	ts	src_ip	src_port	dst_ip	dst_port	proto	service	duration	src_bytes	dst_bytes	conn_st
0	1.713373e+09	169.254.22.198	2730	169.254.237.215	443	tcp	-	0.000567	0	0	F
1	1.713373e+09	169.254.22.198	2731	169.254.237.215	1723	tcp	-	0.000595	0	0	F
2	1.713373e+09	169.254.22.198	2733	169.254.237.215	110	tcp	-	0.000669	0	0	F
3	1.713373e+09	169.254.22.198	2734	169.254.237.215	23	tcp	-	0.000743	0	0	F
4	1.713373e+09	169.254.22.198	2732	169.254.237.215	443	tcp	-	0.000750	0	0	F
...	...	...	...	...	...	...	...	...	...	...	...
9970	1.713373e+09	169.254.22.198	3977	169.254.237.215	5822	tcp	-	0.005653	0	0	F
9971	1.713373e+09	169.254.22.198	3104	169.254.237.215	1721	tcp	-	0.003919	0	0	F
9972	1.713373e+09	169.254.22.198	4600	169.254.237.215	2000	tcp	-	0.004147	0	0	F
9973	1.713373e+09	169.254.22.198	3983	169.254.237.215	691	tcp	-	0.009832	0	0	F
9974	1.713373e+09	169.254.22.198	3827	169.254.237.215	1433	tcp	-	0.000676	0	0	F

9975 rows × 17 columns

Figure 5.4.3.1: Converting Logs to CSV Dataset 1

```
In [3]: sshbf2 = pd.read_csv('connSSHBF2.csv')
        sshbf2
```

Out[3]:

	ts	src_ip	src_port	dst_ip	dst_port	proto	service	duration	src_bytes	dst_bytes	conn_st
0	1.713370e+09	169.254.22.198	60555	169.254.237.215	22	tcp	ssh	0.146890	1147	1631	
1	1.713370e+09	169.254.22.198	60570	169.254.237.215	22	tcp	-	0.034374	23	0	RS
2	1.713370e+09	169.254.22.198	60556	169.254.237.215	22	tcp	ssh	21.546022	1551	1967	
3	1.713370e+09	169.254.22.198	60557	169.254.237.215	22	tcp	ssh	21.562857	1559	1967	
4	1.713370e+09	169.254.22.198	60558	169.254.237.215	22	tcp	ssh	21.580120	1551	1967	
...	...	...	...	...	...	...	...	...	...	...	
125	1.713370e+09	169.254.22.198	60699	169.254.237.215	22	tcp	ssh	8.131181	1323	1787	
126	1.713370e+09	169.254.22.198	60700	169.254.237.215	22	tcp	ssh	8.137298	1323	1787	
127	1.713370e+09	169.254.22.198	60707	169.254.237.215	22	tcp	ssh	2.809731	1171	1683	
128	1.713370e+09	169.254.22.198	60708	169.254.237.215	22	tcp	ssh	2.812163	1171	1683	
129	1.713370e+09	169.254.22.198	60709	169.254.237.215	22	tcp	ssh	2.762568	1171	1683	

130 rows × 17 columns

```
In [4]: xss = pd.read_csv('connXSS.csv')
        xss
```

Out[4]:

	ts	src_ip	src_port	dst_ip	dst_port	proto	service	duration	src_bytes	dst_bytes	conn_st
0	1.712656e+09	169.254.22.198	51343	169.254.237.215	80	tcp	http	0.313582	378	5335	
1	1.712656e+09	169.254.22.198	51345	169.254.237.215	80	tcp	http	0.313490	378	5335	
2	1.712656e+09	169.254.22.198	51342	169.254.237.215	80	tcp	http	0.317780	334	5309	
3	1.712656e+09	169.254.22.198	51344	169.254.237.215	80	tcp	http	0.316926	342	5315	
4	1.712656e+09	169.254.22.198	51346	169.254.237.215	80	tcp	http	0.315051	415	5346	
...	...	...	...	...	...	...	...	...	...	...	
1288	1.712656e+09	169.254.22.198	52305	169.254.237.215	80	tcp	http	0.072642	410	5359	
1289	1.712656e+09	169.254.22.198	52418	169.254.237.215	80	tcp	http	0.215784	438	5381	
1290	1.712656e+09	169.254.22.198	52520	169.254.237.215	80	tcp	http	0.220125	380	5337	
1291	1.712656e+09	169.254.22.198	52420	169.254.237.215	80	tcp	http	0.209538	465	5416	
1292	1.712656e+09	169.254.22.198	52425	169.254.237.215	80	tcp	http	0.189681	447	5402	

1293 rows × 17 columns

Figure 5.4.3.2: Converting Logs to CSV Dataset 2

5.4.4 Detecting and Classifying Threats

The generated CSV dataset will subsequently serve as input for the developed models for classification purposes. Following classification, the output can be effectively visualized using a pie chart. This visualization method offers users a straightforward means to interpret whether the network flow comprises any potential threats. By presenting the classification results in a pie chart format, users can swiftly discern the distribution of normal and threatening network activities, facilitating quick and informed decision-making regarding network security measures. This visual representation enhances the usability and accessibility of the classification results, empowering users to take proactive steps to mitigate identified threats.

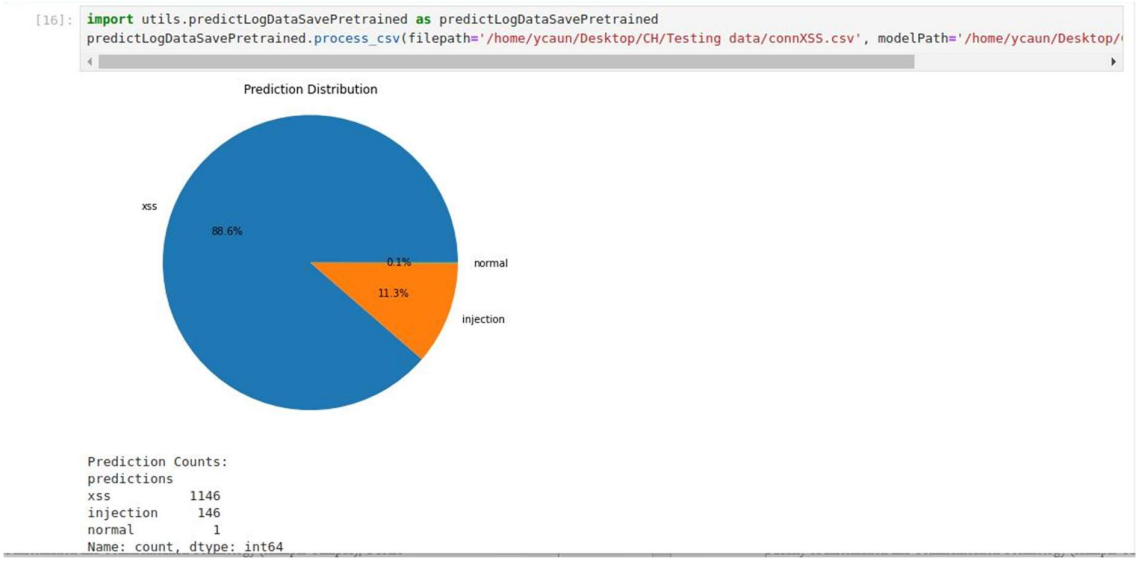


Figure 5.4.4.1: Generating Predictions with SetFit Classification Model 1

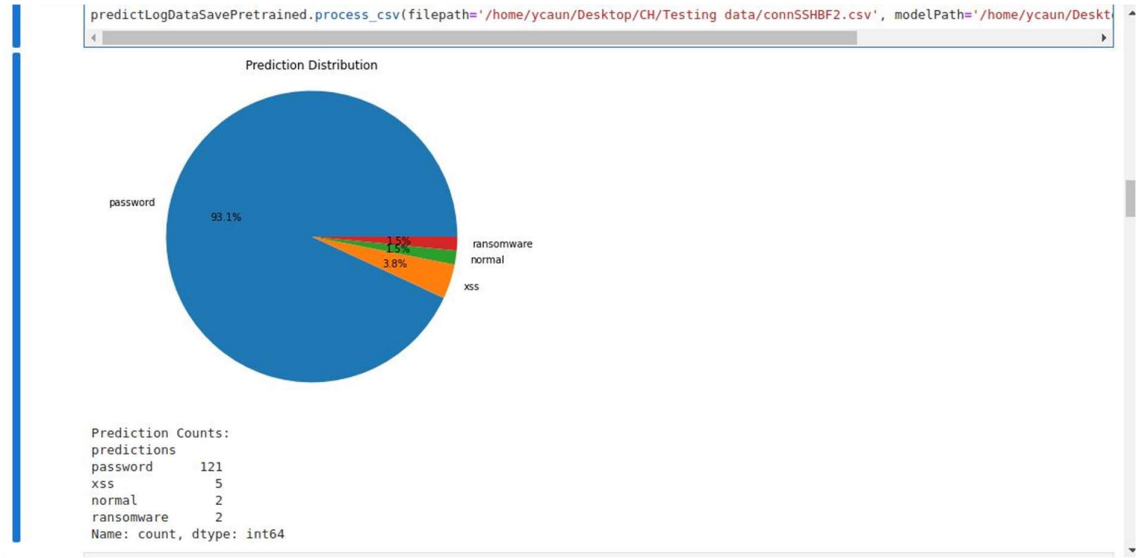


Figure 5.4.4.2: Generating Predictions with SetFit Classification Model 2

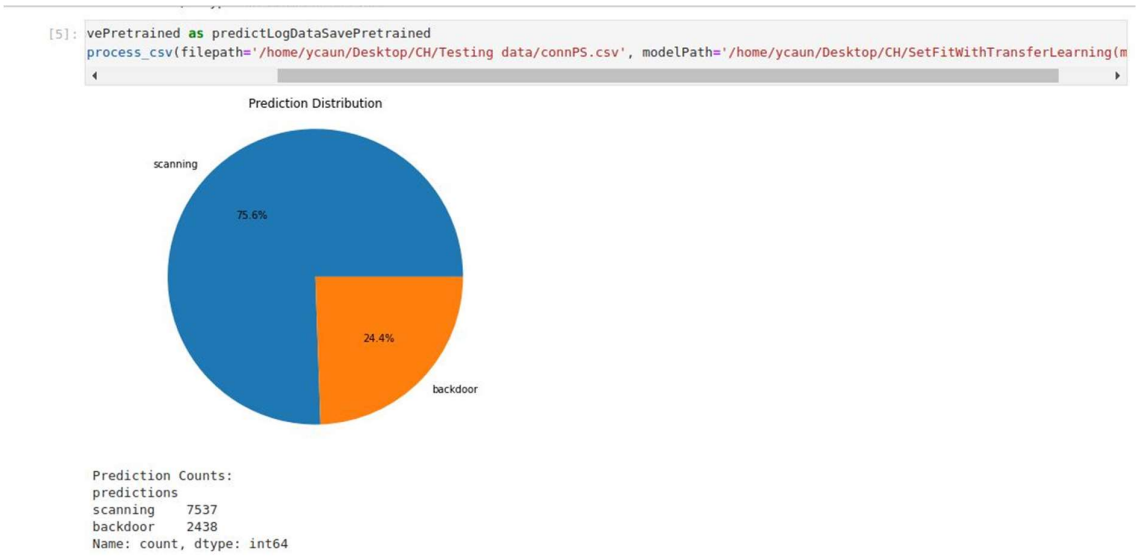


Figure 5.4.4.3: Generating Predictions with SetFit Classification Model 3

The diagrams provided depict pie charts illustrating the classification results of the SetFit (Transfer Learning) model when making inferences on CSV files containing their respective attack classes. It is evident that the majority of logs are accurately classified into their respective attack categories.

5.5 Implementation Issues and Challenges

Difficulties in reproducing the Network Logs preprocessing steps in ToN-IoT

Reproducing the Network Logs preprocessing steps outlined by ToN-IoT proved challenging due to the absence of comprehensive documentation detailing their process for aggregating and integrating features from various log files generated by Zeek. This presented a formidable obstacle to replicating their methodology accurately. To address this issue, we opted to focus solely on the primary log file, specifically the conn.log features. Despite this limitation, we were able to demonstrate that this streamlined approach did not result in significant discernible performance degradation.

Chapter 6 System Evaluation and Discussion

6.1 Models testing and Performance Metrics

6.1.1 Evaluation Cases

In the evaluation phase, the performance of the developed models—namely, the DistilBert-based model, SetFit model, and a KNN classifier—will be rigorously tested using various testing datasets. Initially, these models will undergo evaluation using the ToN-IoT test set, which has been partitioned out beforehand. Additionally, a self-generated dataset comprising unseen attack logs will be utilized to further assess the models' capabilities in identifying novel threats. Furthermore, a combined dataset incorporating both the ToN-IoT test set and the self-generated unseen dataset will be employed to provide a comprehensive evaluation scenario.

Moreover, the effectiveness of transfer learning will be investigated by applying the technique to the models using the unseen data. This analysis aims to ascertain whether transfer learning enhances the models' performance in detecting and mitigating emerging threats. Through these evaluations, we seek to gain insights into the strengths and limitations of each model and assess their suitability for practical deployment in real-world IoT security scenarios.

6.1.2 Evaluation Metrics

To evaluate the proposed method's recognition performance, several key evaluation metrics are utilized: Accuracy (ACC), Recall, Precision, and F1 Score. These metrics assess the model's performance based on four parameters: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

Accuracy (ACC) measures the proportion of correctly classified samples, both normal and abnormal. It is calculated by dividing the sum of TP and TN by the total number of samples (TP + FP + TN + FN).

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

Precision, also referred to as the accuracy rate, quantifies the proportion of correctly predicted normal data out of all predicted normal data. It is determined by dividing TP by the sum of TP and FP.

$$Precision = \frac{TP}{TP + FP}$$

Recall, alternatively known as the check-all rate, represents the proportion of correctly predicted normal data out of all actual normal data. It is calculated by dividing TP by the sum of TP and FN.

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

F1 Score serves as a balanced measure of precision and recall, providing a reconciled average of the two metrics. It is computed by taking the harmonic mean of precision and recall, thereby accounting for their contradictory nature.

$$F1\ Score = \frac{2 \times Precision\ \times Recall}{Precision\ \times Recall}$$

In summary, these evaluation metrics collectively provide insights into the model's ability to correctly classify normal and abnormal samples, ensuring a comprehensive assessment of its recognition performance.

Additionally, confusion matrices will be generated as part of the evaluation process to provide a detailed analysis of the models' performance. Confusion matrices offer a visual representation of the classification results, depicting the number of true positives, true negatives, false positives, and false negatives for each class or category in the dataset.

6.2 Testing Setup and Result

6.2.1 Hardware

The hardware involved in this IoT threat detection project is a computer. A computer is needed to test the models and conduct evaluation with the datasets.

Table 6.2.1.1: Specifications of laptop

Description	Specifications
Model	HP Pavilion Laptop 15-cs3137tx
Processor	Intel Core i7-1065G7
Operating System	Windows 11
Graphic	NVIDIA GeForce MX250 4GB DDR3
Memory	16GB DDR4 RAM
Storage	512 GB NVME SSD

6.2.2 Software

Anaconda is used as the package management system to set up the testing environment. Below includes the libraries that is needed for the testing environment.

1. PyTorch Library
2. Transformer Library
3. SetFit Library
4. Sentence Transformer Library
5. Pandas Library

Jupyter Notebook is used as the main tool for model evaluation with the dataset.

6.2.3 Result Analysis

Evaluation on ToN-IoT test set

Table 6.2.3.1: Accuracy Evaluation on ToN-IoT Testset

IoT Threat Classification Model	Best Accuracy
DistilBert-based (All features)	0.9998
DistilBert-based (Conn features)	0.9802
SetFit (All features)	0.8885
SetFit (Conn features)	0.8807
KNN Classifier	0.9793

Among these models, the DistilBert-based model using all features achieved the highest accuracy, indicating its effectiveness in accurately classifying IoT threats. Interestingly, even

when utilizing only connection features, the DistilBert-based model still performed exceptionally well, with a slightly lower accuracy score.

Additionally, the few-shot learning models, represented by the SetFit architecture, demonstrated notable performance, achieving an accuracy of 0.88. This result is particularly significant considering that the SetFit models were trained with a significantly smaller dataset of only 64 samples for each class of attack which is 640 altogether, compared to the DistilBert-based model, which utilized approximately 400,000 samples for training. This highlights the potential of few-shot learning techniques in achieving competitive performance with limited training data.

Overall, the SetFit models showcase promising performance, particularly considering their ability to achieve robust results with a smaller training dataset.

DistilBert-based (All features) classifier per-class evaluation (ToN-IoT test set)

Table 6.2.3.2: DistilBert-based (All features) classifier per-class evaluation on ToN-IoT Testset

Attack Type	Accuracy	Precision	Recall	F1-score
Backdoor	1.00	1.00	1.00	1.00
Ddos	1.00	1.00	1.00	1.00
Dos	1.00	1.00	1.00	1.00
Injection	1.00	1.00	1.00	1.00
mitm	1.00	1.00	1.00	1.00
normal	0.99	0.99	0.99	0.99
password	0.99	0.99	0.99	0.99
ransomware	0.99	1.00	0.99	0.99
scanning	1.00	1.00	1.00	1.00
xss	0.99	0.99	0.99	0.99

The DistilBert-based model with all features showcases outstanding performance in classifying various types of network threats, demonstrating its effectiveness in accurately identifying and categorizing different attack types with minimal misclassifications. The consistently high scores across all evaluation metrics indicate the model's robustness and reliability in threat detection across a range of attack scenarios.

DistilBert-based (Conn features) classifier per-class evaluation (ToN-IoT test set)

Table 6.2.3.3: DistilBert-based (Conn features) classifier per-class evaluation on ToN-IoT Testset

Attack Type	Accuracy	Precision	Recall	F1-score
Backdoor	1.00	1.00	1.00	1.00
Ddos	0.96	0.98	0.96	0.97
Dos	0.98	0.99	0.98	0.98
Injection	0.96	0.98	0.96	0.97
mitm	0.71	0.66	0.71	0.68
normal	0.99	0.99	0.99	0.99
password	0.98	0.99	0.98	0.98
ransomware	0.93	0.84	0.93	0.88
scanning	0.99	0.98	0.99	0.99
xss	0.91	0.93	0.91	0.92

While the DistilBert-based model using connection features demonstrated strong performance across various attack types, there were some challenges in accurately identifying certain types of threats such as MitM attacks, leading to lower performance metrics for the categories. The limitation of MitM which it has extremely less data samples compared to other classes and the reliance of the attack with dns features may be the cause of this issue. However, the model's overall performance remained robust, with high accuracy and precision in detecting most types of network threats.

SetFit (All features) classifier per-class evaluation (ToN-IoT test set)

Table 6.2.3.4: SetFit (All features) classifier per-class evaluation on ToN-IoT Testset

Attack Type	Accuracy	Precision	Recall	F1-score
Backdoor	0.98	0.97	0.98	0.98
Ddos	0.92	0.85	0.92	0.88
Dos	0.85	0.97	0.85	0.91
Injection	0.93	0.69	0.93	0.79
mitm	0.88	0.05	0.89	0.10
normal	0.87	1.00	0.87	0.93
password	0.97	0.74	0.97	0.84
ransomware	0.96	0.70	0.96	0.81
scanning	0.98	0.92	0.98	0.95

xss	0.82	0.71	0.82	0.76

SetFit (Conn features) classifier per-class evaluation (ToN-IoT test set)

Table 6.2.3.5: SetFit (Conn features) classifier per-class evaluation on ToN-IoT Testset

Attack Type	**Accuracy**	**Precision**	**Recall**	**F1-score**
Backdoor	0.98	1.00	0.98	0.99
Ddos	0.91	0.70	0.91	0.79
Dos	0.95	0.86	0.95	0.90
Injection	0.89	0.64	0.89	0.74
mitm	0.88	0.07	0.88	0.13
normal	0.85	1.00	0.85	0.92
password	0.97	0.72	0.97	0.82
ransomware	0.96	0.67	0.96	0.79
scanning	0.98	0.99	0.98	0.99
xss	0.85	0.78	0.85	0.81

In the SetFit models, there is not a significant difference in performance between the two models. However, it is noticeable that the "mitm" class exhibits an extremely low precision, indicating that many instances from other classes are misclassified as "mitm".

6.2.3.2 Evaluation on unseen dataset (Conn Features Only)

Table 6.2.3.6: Accuracy Evaluation on Unseen Dataset

IoT Threat Classification Model	**Best Accuracy**
DistilBert-based	0.0013
DistilBert-based (Transfer Learning)	0.7800
SetFit	0.0000
SetFit (Transfer Learning)	0.9000
KNN Classifier	0.0000 (unable to fit with unseen text data)

The multiclass IoT Threat Classification models struggle to accurately classify unseen logs into their respective classes. However, the SetFit model demonstrates superior adaptability by efficiently learning new network threats when retrained with only a few new threat samples. The DistilBert-based model also attained an accuracy of 0.78%, positioning it as the second-best performer following the implementation of model retraining. In contrast, the KNN

classifier faces limitations in predicting new protocols due to constraints with the label encoder's ability to accommodate new protocol values.

DistilBert-based (Transfer Learning) (Conn features) classifier per-class evaluation (unseen dataset)

Table 6.2.3.7: DistilBert-based (Transfer Learning) (Conn features) classifier per-class evaluation on Unseen Dataset

Attack Type	Accuracy	Precision	Recall	F1-score
password	0.80	1.00	0.94	0.97
scanning	0.22	1.00	0.52	0.68
xss	0.56	1.00	0.88	0.94

SetFit (Transfer Learning) (Conn features) classifier per-class evaluation (unseen dataset)

Table 6.2.3.8: SetFit (Transfer Learning) (Conn features) classifier per-class evaluation on Unseen Dataset

Attack Type	Accuracy	Precision	Recall	F1-score
password	0.98	1.00	0.98	0.99
scanning	0.76	1.00	0.76	0.86
xss	0.96	0.98	0.96	0.97

The assessment of unseen datasets reveals that the SetFit model demonstrates superior adaptability to new, previously unseen threats following the retraining process, substantially improving its accuracy from 0% to 90%. Conversely, the DistilBert-based model exhibits suboptimal performance post-retraining, notably with significantly reduced recall for the scanning class. More samples are needed to train the DistilBert-based model to fit with the new threats.

5.2.3.3 Evaluation on combined dataset (Conn Features Only)

Table 6.2.3.9: Accuracy Evaluation on Combined Dataset

IoT Threat Classification Model	Best Accuracy
DistilBert-based	0.9359
DistilBert-based (Transfer Learning)	0.9592

SetFit	0.8913
SetFit (Transfer Learning)	0.9223

Given that the unseen dataset constitutes only 3% of the overall dataset, the evaluation conducted on the combined dataset demonstrates the adaptability of both the SetFit and DistilBert-based models. Through retraining using transfer learning techniques, these models showcase their ability to effectively adapt to both new and existing threats, thereby improving their classification performance across the entire dataset.

DistilBert-based (Transfer Learning) (Conn features) classifier per-class evaluation (combined dataset)

Table 6.2.3.10: DistilBert-based (Transfer Learning) (Conn features) classifier per-class evaluation on Combined Dataset

Attack Type	Accuracy	Precision	Recall	F1-score
Backdoor	1.00	0.94	1.00	0.97
Ddos	0.94	1.00	0.94	0.97
Dos	1.00	0.98	1.00	0.99
Injection	1.00	0.96	1.00	0.98
mitm	0.96	0.98	0.96	0.97
normal	1.00	0.94	1.00	0.97
password	0.98	1.00	0.98	0.99
ransomware	0.94	0.89	0.94	0.91
scanning	0.95	1.00	0.95	0.97
xss	0.91	0.98	0.91	0.94

SetFit (Transfer Learning) (Conn features) classifier per-class evaluation (combined dataset)

Table 6.2.3.11: SetFit (Transfer Learning) (Conn features) classifier per-class evaluation on Combined Dataset

Attack Type	Accuracy	Precision	Recall	F1-score
Backdoor	1.00	0.96	1.00	0.98
Ddos	0.88	0.92	0.90	0.90
Dos	0.92	0.98	0.95	0.95

Injection	0.98	0.94	0.96	0.96
mitm	0.76	0.97	0.85	0.85
normal	0.90	0.82	0.86	0.86
password	0.96	0.98	0.97	0.97
ransomware	0.98	0.89	0.93	0.93
scanning	0.95	1.00	0.97	0.97
xss	0.89	0.80	0.84	0.84

According to the evaluation metrics delineated previously, the SetFit model, leveraging transfer learning, demonstrates commendable performance as the second-best performer, achieving an accuracy score of 0.9223, with only a marginal 4% difference compared to the DistilBert-based model, which achieved an accuracy score of 0.9592. It is noteworthy to emphasize that the SetFit model is specifically designed as a few-shot learning model, indicating its capacity to achieve competitive results despite its reduced data requirement.

6.3 Project Challenges

The ToN-IoT dataset lacks comprehensive documentation regarding the methodologies employed during data collection, particularly concerning the specific attacks executed. Consequently, reproducing these attacks verbatim is unfeasible. To address this limitation, we devised a bespoke attack methodology tailored to our IoT testbed environment. This approach enables us to simulate and execute attacks representative of those encountered in real-world scenarios, thereby compensating for the absence of detailed attack documentation in the ToN-IoT dataset.

Additionally, the ToN-IoT dataset lacks detailed specifications for all attack types. For instance, the "dos" class encompasses a broad spectrum of attacks, including UDP floods, SYN floods, NTP amplification, DNS amplification, SSDP amplification, IP fragmentation, SYN-ACK floods, Ping of Death, and TCP SYN flood. This lack of granularity impedes further analysis and extraction of specific attack samples from distinct classes for the purpose of few-shot learning.

The multifaceted nature of network threats poses a formidable challenge for current datasets to encompass the full spectrum of attack variations across all classes. This limitation significantly

impedes the ability of multiclass classification models to achieve comprehensive generalization across diverse network attack threats. Hence, deploying the multiclass IoT threat classification model in real-life scenarios presents a substantial challenge. As such, this challenge underscores the critical need for innovative methodologies that can effectively address the inherent complexities and variations inherent in IoT threat landscapes. Overcoming this obstacle is paramount to advancing the efficacy and reliability of threat detection and classification systems in safeguarding IoT networks against evolving security risks.

Time and processing power constraints have imposed limitations on our ability to conduct extensive model fine-tuning and experimentation, particularly considering the complexity of developing two distinct models. These constraints have hindered our capacity to explore various hyperparameters, optimization techniques, and model architectures comprehensively. As a result, our experimentation scope has been constrained, potentially impacting the depth of our model optimization efforts and the breadth of our findings. Despite the constraints, we have managed to achieve high-performance results for our models. This underscores the effectiveness of our approaches and highlights the robustness of the methodologies employed.

6.4 Objectives Evaluation

Objective 1 Evaluation:

The establishment of an IoT network testbed for generating and capturing unseen network threats represents a significant milestone in the research objectives of generating unseen network threats to evaluate models' performance in classifying realistic, unseen realistic network threats. By leveraging this testbed, the captured dataset was utilized to comprehensively evaluate the performance of the developed models. However, the evaluation results revealed that the multiclass IoT threats classification model exhibited limitations in accurately predicting unseen samples. This finding highlights the challenges inherent in addressing the complexity and variability of real-world network threat scenarios especially when we are performing multiclass classification for 10 different attack classes.

Objective 2 Evaluation:

The research successfully achieved the objective of developing innovative Transformer-based model tailored specifically for IoT threat detection, leveraging advanced Natural Language

Processing (NLP) techniques. Two Transformer-based language classification models were meticulously developed, trained, and rigorously tested to evaluate their performance. Both models demonstrated commendable performance in accurately classifying a wide range of IoT network threats, indicating the effectiveness of the novel methodology in enhancing threat detection capabilities in IoT environments.

Objective 3 Evaluation:

The successful development of the SetFit-based IoT threats classification model represents a significant achievement in the research objectives of developing the novel few-shot learning model for IoT threat classification. By leveraging few-shot learning techniques, the model has demonstrated its efficacy in detecting and classifying network threats in IoT environments. This accomplishment underscores the effectiveness of adopting innovative methodologies to enhance the capabilities of threat detection models, particularly in scenarios where limited labeled data is available.

Objective 4 Evaluation:

The implementation of model retraining using transfer learning approaches represents a pivotal step towards enhancing the prediction accuracy of the classification model for unseen attacks. Through this process, the model underwent adaptation to incorporate knowledge from pre-existing model, thereby enhancing its ability to classify novel threat vectors encountered in the network. The results of the retraining efforts demonstrate a notable improvement in the model's performance, reaffirming the efficacy of transfer learning as a valuable strategy for enabling the model to adapt and evolve in response to emerging threats. This successful implementation underscores the importance of ongoing refinement and optimization to ensure the model remains robust and effective in safeguarding IoT networks against evolving security challenges.

6.5 Concluding Remark

In conclusion, this research endeavor has led to significant advancements in the domain of IoT threat detection and classification. Through the development and evaluation of novel methodologies and models, we have made substantial strides towards enhancing the security posture of IoT networks. The successful implementation of Transformer-based language classification models, coupled with the innovative utilization of few-shot learning techniques,

underscores the effectiveness of advanced AI-driven approaches in mitigating emerging threats.

Furthermore, the establishment of an IoT network testbed for generating and evaluating unseen network threats has provided invaluable insights into the capabilities and limitations of our models. While challenges persist in accurately predicting unseen threats, the utilization of transfer learning approaches has demonstrated promising results in bolstering the models' adaptability and prediction accuracy.

This research stands as a pioneering endeavor in the field, marking the first instance of testing a multiclass threat classification model with a self-generated unseen dataset. By undertaking this approach, we have pushed the boundaries of current methodologies and expanded the scope of research in IoT threat detection and classification. This groundbreaking work not only fills a crucial gap in existing literature but also lays the foundation for future studies to build upon. The insights gained from this research are invaluable, providing a roadmap for further advancements in the realm of IoT security.

Looking ahead, continued research and development efforts are warranted to further refine and optimize our models, ensuring their efficacy in addressing the evolving threat landscape of IoT networks. By leveraging cutting-edge technologies and methodologies, we remain committed to advancing the state-of-the-art in IoT threat detection and classification, ultimately fostering a more secure and resilient IoT ecosystem.

Chapter 7 Conclusion and Recommendation

7.1 Conclusion

In conclusion, this research project has successfully devised and assessed an innovative methodology for detecting IoT threats utilizing limited labeled textual datasets leveraging Large Language Model (LLM). The SetFit architecture has demonstrated notable efficacy in accurately classifying threats, even when confronted with scant training data. This addresses the pressing need for adaptive security frameworks capable of promptly responding to emergent IoT vulnerabilities. Comparative analysis underscores the superiority of the proposed SetFit approach over conventional techniques, particularly in scenarios characterized by sparse data availability. The investigation underscores the potential of leveraging deep contextual representations from Sentence Transformer for facilitating few-shot learning, thereby enabling the construction of robust models on resource-constrained edge devices commonly encountered in IoT environments. Furthermore, the elucidation of the model adaptation process underscores the capability of the SetFit-based IoT Threat Classification model to undergo retraining with limited samples and adeptly accommodate new, previously unseen threats. Additionally, our research has showcased the efficacy of LLM models in detecting IoT threats, as evidenced by the outstanding performance of the DistilBert-based IoT threat classification model, achieving an accuracy of 99.98%.

7.2 Recommendation

Future endeavors could address the limitations of multiclass IoT threat classification models by exploring alternative approaches. Prior studies have demonstrated that binary classification, distinguishing between logs indicative of threats and those that are not, may exhibit better generalization capabilities in detecting unseen samples. Integrating a hybrid pipeline encompassing both binary and multiclass models could potentially offer a solution to this challenge.

Moreover, future endeavors could center on exploring enhanced network flow features capable of encapsulating diverse attack characteristics. The identification of representative features holds promise for advancing the efficacy of IoT threat classification models. By utilizing on

features that aptly capture nuanced attack information, the models' classification capabilities can be further refined, thereby augmenting their effectiveness in discerning and mitigating IoT-related security risks.

REFERENCES

[1] N. Moustafa, 'A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets', *Sustain Cities Soc*, vol. 72, Sep. 2021, doi: 10.1016/j.scs.2021.102994.

[2] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and Adna N Anwar, 'TON-IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems', *IEEE Access*, vol. 8, pp. 165130–165150, 2020, doi: 10.1109/ACCESS.2020.3022862.

[3] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. H. D. 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, doi: 10.1109/JIOT.2021.3085194.

[4] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, 'CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment', *Sensors*, vol. 23, no. 13, p. 5941, Jun. 2023, doi: 10.3390/s23135941.

[5] A. P. & M. J. E. Sebastian Garcia, 'IoT-23: A labeled dataset with malicious and benign IoT network traffic', Zenodo.

[6] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, 'Edge-IIoTset: A New Comprehensive Realistic Cyber Security Dataset of IoT and IIoT Applications for Centralized and Federated Learning', *IEEE Access*, vol. 10, pp. 40281–40306, 2022, doi: 10.1109/ACCESS.2022.3165809.

[7] G. Guo, X. Pan, H. Liu, F. Li, L. Pei, and K. Hu, 'An IoT Intrusion Detection System Based on TON IoT Network Dataset', in *2023 IEEE 13th Annual Computing and Communication Workshop and Conference, CCWC 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 333–338. doi: 10.1109/CCWC57344.2023.10099144.

[8] A. R. Gad, M. Haggag, A. A. Nashat, and T. M. Barakat, 'A Distributed Intrusion Detection System using Machine Learning for IoT based on ToN-IoT Dataset', *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 6, 2022, doi: 10.14569/IJACSA.2022.0130667.

[9] A. R. Gad, A. A. Nashat, and T. M. Barkat, 'Intrusion Detection System Using Machine Learning for Vehicular Ad Hoc Networks Based on ToN-IoT Dataset', *IEEE Access*, vol. 9, 2021, doi: 10.1109/ACCESS.2021.3120626.

[10] A. Sharma, H. Babbar, and A. Sharma, 'TON-IoT: Detection of Attacks on Internet of Things in Vehicular Networks', in *6th International Conference on Electronics, Communication and Aerospace Technology, ICECA 2022 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 539–545. doi: 10.1109/ICECA55336.2022.10009070.

[11] W. Ding, M. Abdel-Basset, and R. Mohamed, 'DeepAK-IoT: An effective deep learning model for cyberattack detection in IoT networks', *Inf Sci (N Y)*, vol. 634, pp. 157–171, Jul. 2023, doi: 10.1016/j.ins.2023.03.052.

[12] I. Tareq, B. M. Elbagoury, S. El-Regaily, and E. S. M. El-Horbaty, 'Analysis of ToN-IoT, UNW-NB15, and Edge-IIoT Datasets Using DL in Cybersecurity for IoT', *Applied Sciences (Switzerland)*, vol. 12, no. 19, Oct. 2022, doi: 10.3390/app12199572.

[13] I. Idrissi, M. Azizi, and O. Moussaoui, 'Accelerating the update of a DL-based IDS for IoT using deep transfer learning', *Indonesian Journal of Electrical Engineering and*

Computer Science, vol. 23, no. 2, pp. 1059–1067, Aug. 2021, doi: 10.11591/ijeecs.v23.i2.pp1059-1067.

[14] P. Wang, X. Wang, Y. Song, J. Huang, P. Ding, and Z. Yang, 'TransIDS: A Transformer-based approach for intrusion detection in Internet of Things using Label Smoothing', in *2023 4th International Conference on Computer Engineering and Application, ICCEA 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 216–222. doi: 10.1109/ICCEA58433.2023.10135426.

[15] A. Ghourabi, 'A Security Model Based on LightGBM and Transformer to Protect Healthcare Systems From Cyberattacks', *IEEE Access*, vol. 10, pp. 48890–48903, 2022, doi: 10.1109/ACCESS.2022.3172432.

[16] M. Wang, N. Yang, and N. Weng, 'Securing a Smart Home with a Transformer-Based IoT Intrusion Detection System', *Electronics (Switzerland)*, vol. 12, no. 9, May 2023, doi: 10.3390/electronics12092100.

[17] P. Anand, Y. Singh, H. Singh, M. D. Alshehri, and S. Tanwar, 'SALT: transfer learning-based threat model for attack detection in smart home', *Sci Rep*, vol. 12, no. 1, Dec. 2022, doi: 10.1038/s41598-022-16261-9.

[18] L. Tunstall *et al.*, 'Efficient Few-Shot Learning Without Prompts', Sep. 2022.

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3, 3	**Study week no.:2**
Student Name & ID: Chua Cheng Han 2001761	
Supervisor: Aun Yichiet	
Project Title: IoT Threats Detection using Few-Shots Learning	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Research on the methods on improving SetFit model.

2. WORK TO BE DONE

Discover the feasibility of information gain.

3. PROBLEMS ENCOUNTERED

The performance of SetFit model cannot be improved through extensive fine-tuning.

4. SELF EVALUATION OF THE PROGRESS

In the progress, looks ok.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3, 3	**Study week no.:4**
Student Name & ID: Chua Cheng Han 2001761	
Supervisor: Aun Yichiet	
Project Title: IoT Threats Detection using Few-Shots Learning	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Begin the planning to set up IoT testbed for model deployment.

2. WORK TO BE DONE

Get Raspberry Pi from UTAR.

3. PROBLEMS ENCOUNTERED

Information gain is not suitable to implement in SetFit.

4. SELF EVALUATION OF THE PROGRESS

Remained the SetFit architecture as it is already robust. Move towards stage 2 of the project.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3, 3	**Study week no.:6**
Student Name & ID: Chua Cheng Han 2001761	
Supervisor: Aun Yichiet	
Project Title: IoT Threats Detection using Few-Shots Learning	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Discovering the network testbed and start to perform network attacks to the environment.

2. WORK TO BE DONE

Finalize the attacks to be performed. Discovering the process of generating logs and covert it to dataset.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

Everything under the planning.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3, 3	**Study week no.:8**
Student Name & ID: Chua Cheng Han 2001761	
Supervisor: Aun Yichiet	
Project Title: IoT Threats Detection using Few-Shots Learning	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Collected all the attacks network flow.

2. WORK TO BE DONE

Converts all the attacks into dataset.

3. PROBLEMS ENCOUNTERED

Discovering the process of streamlining generated logs to Kafka failed. Change to manual method.

4. SELF EVALUATION OF THE PROGRESS

Need to be faster.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3, 3	**Study week no.:10**
Student Name & ID: Chua Cheng Han 2001761	
Supervisor: Aun Yichiet	
Project Title: IoT Threats Detection using Few-Shots Learning	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Test the models with the generated dataset and found that the model cannot accurately predict.

2. WORK TO BE DONE

Research on the way of model retraining to adapt new threats.

3. PROBLEMS ENCOUNTERED

Models are vulnerable to unseen threats.

4. SELF EVALUATION OF THE PROGRESS

Need to be faster.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: 3, 3	**Study week no.:12**
Student Name & ID: Chua Cheng Han 2001761	
Supervisor: Aun Yichiet	
Project Title: IoT Threats Detection using Few-Shots Learning	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Finalize the evaluation on the model after the model adaptation process. The result is acceptable.

2. WORK TO BE DONE

Continue to fine tune the model retraining methods.

3. PROBLEMS ENCOUNTERED

-

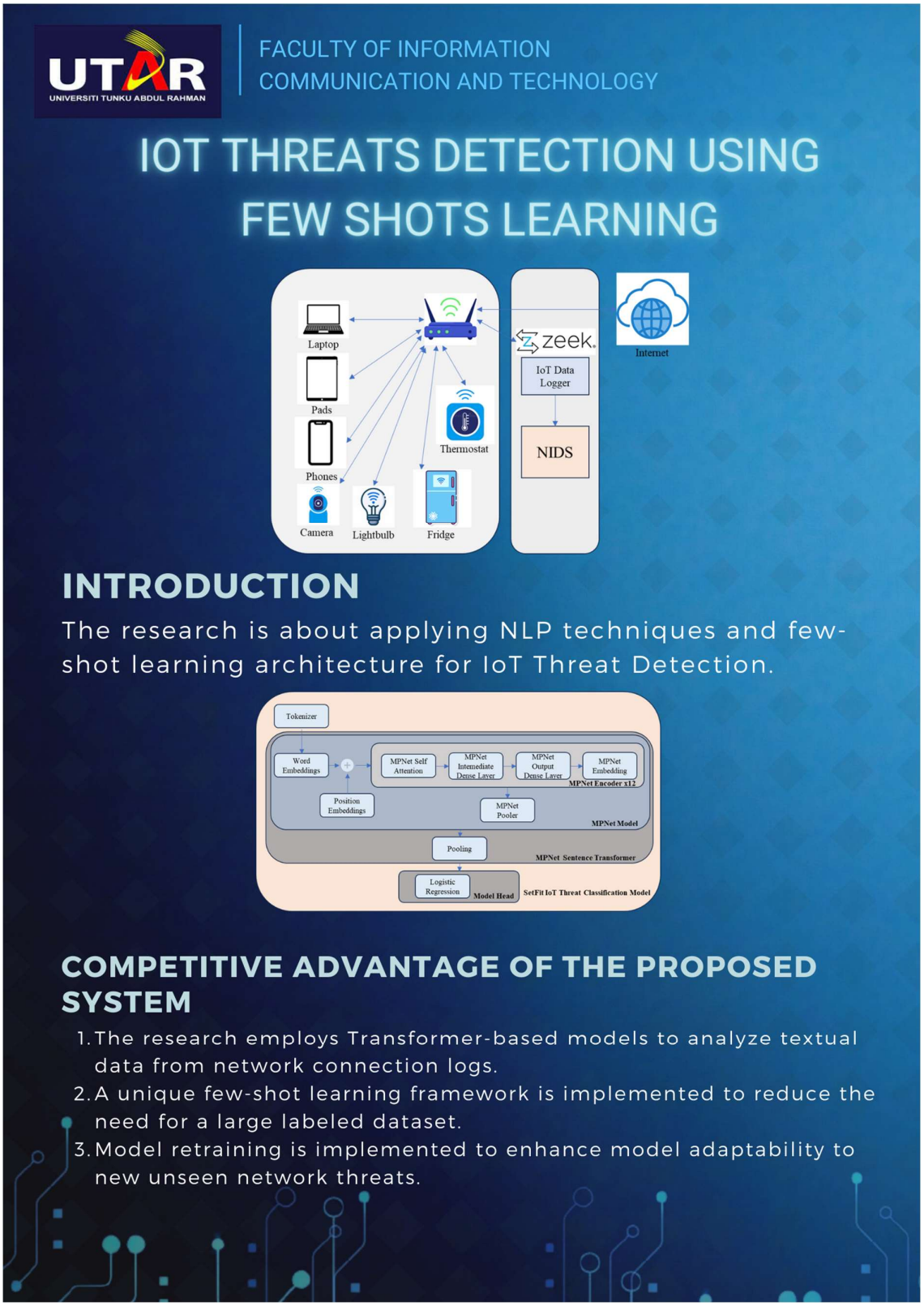
4. SELF EVALUATION OF THE PROGRESS

Almost in the process of finalizing FYP2.

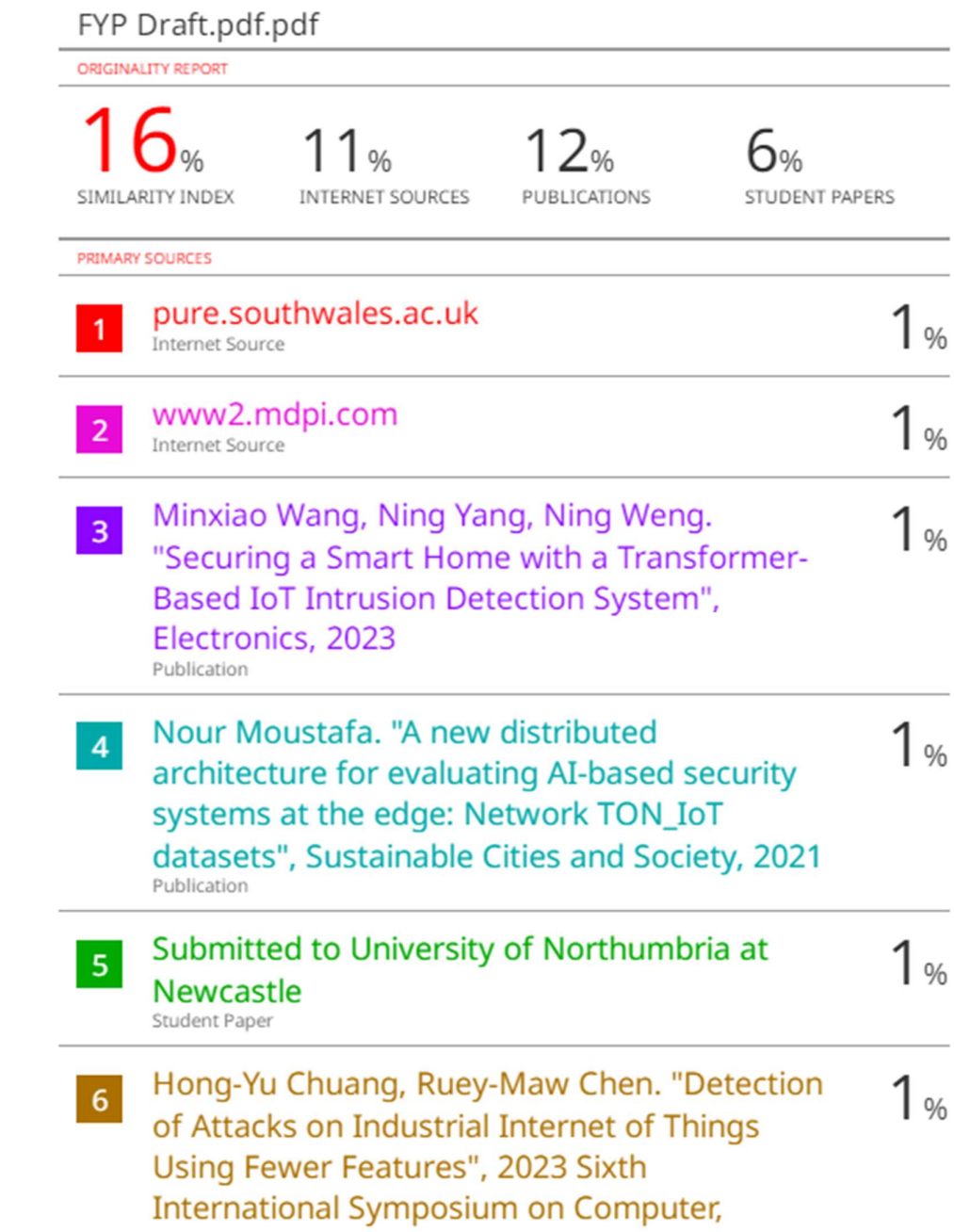
Supervisor's signature

Student's signature

POSTER


UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION
COMMUNICATION AND TECHNOLOGY
IOT THREATS DETECTION USING
FEW SHOTS LEARNING
Laptop
Pads
Phones
Camera
Lightbulb
Thermostat
Fridge
zeek
IoT Data Logger
NIDS
Internet
INTRODUCTION
The research is about applying NLP techniques and few-shot learning architecture for IoT Threat Detection.
Tokenizer
Word Embeddings
Position Embeddings
MPNet Self Attention
MPNet Intemediate Dense Layer
MPNet Output Dense Layer
MPNet Embedding
MPNet Encoder x12
MPNet Pooler
MPNet Model
Pooling
MPNet Sentence Transformer
Logistic Regression
Model Head
SetFit IoT Threat Classification Model
COMPETITIVE ADVANTAGE OF THE PROPOSED SYSTEM
1. The research employs Transformer-based models to analyze textual data from network connection logs.
2. A unique few-shot learning framework is implemented to reduce the need for a large labeled dataset.
3. Model retraining is implemented to enhance model adaptability to new unseen network threats.

PLAGIARISM CHECK RESULT



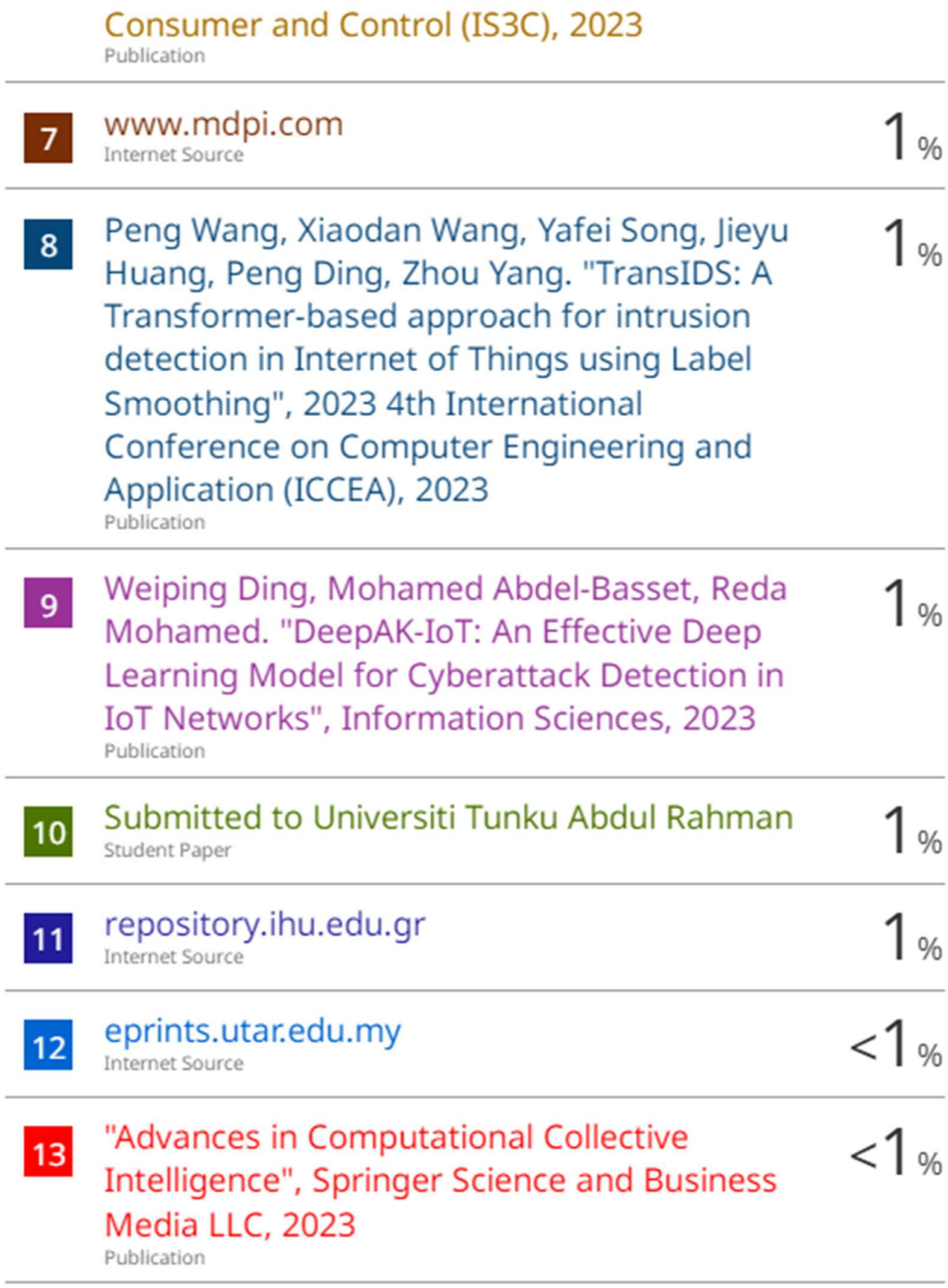
FYP Draft.pdf.pdf

ORIGINALITY REPORT

16%	11%	12%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	pure.southwales.ac.uk Internet Source	1%
2	www2.mdpi.com Internet Source	1%
3	Minxiao Wang, Ning Yang, Ning Weng. "Securing a Smart Home with a Transformer-Based IoT Intrusion Detection System", Electronics, 2023 Publication	1%
4	Nour Moustafa. "A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets", Sustainable Cities and Society, 2021 Publication	1%
5	Submitted to University of Northumbria at Newcastle Student Paper	1%
6	Hong-Yu Chuang, Ruey-Maw Chen. "Detection of Attacks on Industrial Internet of Things Using Fewer Features", 2023 Sixth International Symposium on Computer,	1%



	Consumer and Control (IS3C), 2023 Publication	
7	www.mdpi.com Internet Source	1 %
8	Peng Wang, Xiaodan Wang, Yafei Song, Jieyu Huang, Peng Ding, Zhou Yang. "TransIDS: A Transformer-based approach for intrusion detection in Internet of Things using Label Smoothing", 2023 4th International Conference on Computer Engineering and Application (ICCEA), 2023 Publication	1 %
9	Weiping Ding, Mohamed Abdel-Basset, Reda Mohamed. "DeepAK-IoT: An Effective Deep Learning Model for Cyberattack Detection in IoT Networks", Information Sciences, 2023 Publication	1 %
10	Submitted to Universiti Tunku Abdul Rahman Student Paper	1 %
11	repository.ihu.edu.gr Internet Source	1 %
12	eprints.utar.edu.my Internet Source	<1 %
13	"Advances in Computational Collective Intelligence", Springer Science and Business Media LLC, 2023 Publication	<1 %

PLAGIARISM CHECK RESULT

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

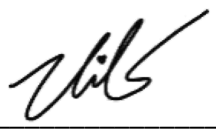
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Chua Cheng Han
ID Number(s)	20ACB01761
Programme / Course	CS
Title of Final Year Project	IoT Threats Detection using Few Shots Learning

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 16 % **Similarity by source** Internet Sources: 11 % Publications: 12 % Student Papers: 6 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.



Signature of Supervisor

Name: Aun Yichiet

Date: 26/04/2024

Signature of Co-Supervisor

Name: _____

Date: _____

Bachelor of Computer Science (Honours)
Faculty of Information and Communication Technology (Kampar Campus), UTAR

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB01761
Student Name	Chua Cheng Han
Supervisor Name	Aun YiChiet

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

_____
(Signature of Student)
Date: 25/4/2024