TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

PROJECT NO.: THA079MSISE018

TITLE: INTERACTIVE MALWARE ANALYSIS USING ROBERTA BASED MODEL

BY

UTKARSHA SHUKLA

A PROJECT

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN INFORMATICS AND INTELLIGENT SYSTEMS ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

KATHMANDU, NEPAL

AUGUST, 2024

Title: Interactive Malware Analysis using Roberta Based Model

by

Utkarsha Shukla

THA079MSISE018

Project Supervisor

Er. Om Prakash Mahato

A project submitted in partial fulfillment of the requirements for the degree of

Master of Science in Informatics and Intelligent Systems Engineering

Department of Electronics and Computer Engineering

Institute of Engineering, Thapathali Campus

Tribhuvan University

Kathmandu, Nepal

August, 2024

ACKNOWLEDGMENT

This project work would not have been achievable without the direction, support and the help of many individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First of all, I would like to express my sincere gratefulness to my supervisor, **Er. Om Prakash Mahato**, of **Nepal Telecommunications Authority** for providing invaluable guidance, insightful comments, thorough suggestions, and encouragement throughout the duration of this project work. I would also like to extend my heartfelt thanks to the M.Sc. coordinator, **Er. Dinesh Baniya Kshatri**, for managing the project work, offering insightful feedback, and demonstrating immeasurable patience.

I am also grateful to my classmates and friends for offering me advice and moral support. To my family, thank you for your unwavering achievement in all of my pursuits and inspiring me to follow my dreams. I am particularly appreciative to my parents, who supported me emotionally, believed in me and wanted the best for me.

Utkarsha Shukla
THA079MSISE018
August, 2024

ABSTRACT

Abstract: This study introduces a novel approach for malware analysis by creating a context from datasets with a transformer-encoder based model. The hypothesis is that the process will significantly enhance the accuracy and performance of malware detection. The research involves collecting relevant Android malware datasets, converting them into text if they are not in specific format and making them contextual to different android malware labels . The preprocessed data is then analyzed by SecureBERT model which is based on Roberta which is transformer encoder model, for precise malware classification which is then tested by giving text message to the prompt and the model classifies the message. The findings demonstrate the efficacy of this approach, showcasing its potential in bolstering cybersecurity measures and analyzing malware bases on real-time texts.This method offers a promising advancement in the field of malware detection and analysis.

Keywords: android, cybersecurity , SecureBERT, transformer

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Must be in Lexicographical Order

ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
DBN	Deep Belief Network
CNN	Convolutional Neural Networks
DPI	Deep Packet Inspection
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
LoRA	Low-Rank Adaptation
LSTM	Long Short Memory Network
MalBERT	Malware Bidirectional Encoder Representations from Transformers
ML	Machine Learning
ROBERTa	Robustly Optimized BERT Approach

1 INTRODUCTION

1.1 Background

Malware has become a common threat in the digital world, causing significant harm to individuals, organizations, and governments. The financial losses, data breaches, exposure of sensitive information resulting from malware attacks highlight the urgent need for robust detection and prevention mechanisms.Modern malware is increasingly sophisticated, it employs advanced techniques to evade detection, such as polymorphism, metamorphism, and advanced obfuscation methods. This evolving complexity makes traditional signature-based detection methods deficient in nature.Conventional antivirus solutions depend heavily on static signatures, which are specific patterns or sequences found in known malware. While effective against previously identified threats, these methods struggle to detect new, unknown malware.Some contemporary approaches use behavioral analysis, which is performend by monitoring system activities to identify suspicious behavior. However, these methods arelikely to produce high false-positive rates and they may not always detect sophisticated malware that mimics legitimate software behavior.Machine learning models can extract and learn from various features of malware, such as system calls, network traffic, and file characteristics and others. These models can generalize from the training data to detect new malware variants[1]. By incorporating machine learning models in the system, malware detection systems become more adaptive and resilient to the rapidly evolving threat landscape. They can detect new, previously unknown malware by identifying suspicious patterns and behaviors rather than relying on a predefined signature database. However, constant updation and retraining is required to remain effective against new types of malware. Transformer-based models have become an influential architecture in various natural language processing (NLP) applications, including malware detection and classification. Originating from the ”Attention is All You Need” paper by Vaswani et al. (2017), transformer models introduced a novel mechanism called self-attention, enabling them to capture complex dependencies between different parts of a sequence. This attention mechanism makes transformers particularly powerful for processing long sequences of text and understanding context on a global scale.

In malware detection, the transformation from traditional signature-based and heuristic

methods to machine learning and, more recently, transformer-based models represents a significant evolution. Traditional methods often struggled to adapt to the growing complexity and diversity of malware, especially with polymorphic and metamorphic malware types that continuously change their appearance to evade detection. Transformer models, specially BERT (Bidirectional Encoder Representations from Transformers) and its variants like Roberta, have proven to be effective for textual malware classification due to their ability to process large corpora of cybersecurity-related texts. SecureBERT is a variant of Roberta model which is pretrained on vast amounts of text data, including research papers, forums, and threat reports, which allows them to understand nuanced cybersecurity threats and behaviors. Fine-tuning these models for malware classification helps them specialize in identifying specific characteristics of various malware types, making them adept at classifying malware by analyzing contextual signals within the text. The key advantage of transformer models in malware analysis lies in their capability to understand and model relationships within complex textual descriptions of malware behaviors, attack vectors, and system vulnerabilities. By employing this contextual understanding, transformer-based models can more accurately detect and classify malware threats in diverse textual data sources, ranging from malware descriptions in reports to threat indicators in logs. This shift from traditional rule-based methods to sophisticated AI-driven models represents a notable advancement in the field of malware detection, offering greater adaptability, accuracy, and scalability in identifying modern threats.

1.2 Motivation

The motivation behind this project is rooted in the critical necessity to enhance the precision and efficiency of malware detection and analysis in the face of increasingly sophisticated cybersecurity threats. As the digital landscape evolves, traditional malware detection methods often struggle to keep pace with the complex and varied nature of modern malware. Leveraging advanced machine learning techniques, particularly transformer-based models, presents a promising solution to these challenges. We will be applying SecureBERT which is known for the exceptional capability to understand and process contextual information within data, the model highly effective in analyzing the intricate patterns and behaviors characteristic of malware. This project is looking to utilize these strengths to improve the accuracy of identifying and classifying malware, thereby fortifying cybersecurity defenses. This study introduces an innovation approach

for Android malware detection by integrating advanced data preprocessing techniques with a transformer-based language model. The hypothesis is that this combination will significantly enhance the accuracy and efficiency of malware detection. The research involves collecting relevant Android malware datasets, extracting key features, and applying thorough data preprocessing to effectively organize the data. The preprocessed data is then analyzed by SecureBERT, for precise malware identification and characterization. The findings demonstrate the effectiveness of this hybrid approach, showcasing its potential in bolstering cybersecurity measures and mitigating the impact of malicious threats in digital environments. This integrated method offers a promising advancement in the field of malware detection and analysis.

1.3 Problem Statement

The challenge involves in developing an accurate and efficient system to detect and analyze malware by applying transformer based models to address challenges such as identifying complex patterns and relationships in malware data, handling long sequential features crucial for detection, improving precision and efficiency in malware detection, adapting transformer models for cybersecurity tasks, and leveraging transfer learning for low-resource tasks. The goal is to create a robust system that enhances cybersecurity defenses against evolving threats by effectively detecting and analyzing malicious software. Challenges faced in traditional system could be as:

1. **Complexity and Variability of Malware Data:** Modern malware exhibits highly complex and diverse behaviors which ranges from polymorphic and metamorphic to fileless and stealthy techniques. These complexities make it extremely difficult for traditional signature-based detection methods to accurately identify and classify malicious activities. As a result, many sophisticated malware variants evade detection, leaving digital infrastructures vulnerable to exploitation and compromise.

2. **Limitations of Traditional Techniques:** Existing malware detection systems predominantly rely on signature-based approaches, which match files against known patterns of malicious code. However, this method is inherently reactive and fails to detect previously unseen or zero-day malware strains.Signature-based

detections are ineffective against polymorphic malware, since it continuously alters its code to evade detection. If we rely entirely on traditional techniques it will leave organizations vulnerable to emerging and unknown malware threats.

3. **High False Positive Rates:** Conventional malware detection methods often generates a significant number of false positives which results in identifying legitimate files or activities as malicious in an incorrect manner. These false alarms not only strain cybersecurity resources but also lead to alert fatigue among security personnel, diminishing the effectiveness of threat response efforts. Consequently, organizations face challenges in distinguishing genuine threats from benign anomalies, thereby impeding timely and accurate incident response.

From the challenges mentioned above it makes us clear that there exists an important need for more advanced and effective malware analysis techniques. By leveraging innovative approaches and cutting-edge technologies, such as machine learning and artificial intelligence, it is essential to develop robust solutions capable of accurately detecting, analyzing, and mitigating sophisticated malware threats. Addressing these challenges is crucial to fortifying cybersecurity defenses, protecting digital assets, and preserving the integrity of critical infrastructures in an increasingly hostile cyber landscape.[2]

1.4 Objective of Project

The main objectives of the project are:

- Developing and implementing SecureBERT model will utilize its advanced contextual understanding capabilities to effectively identify and classify various types of malware. By training these models on diverse and comprehensive datasets, the aim is to improve the accuracy and efficiency of malware detection compared to traditional methods.
- Develop and fine-tune SecureBERT to handle large-scale malware textual datasets, ensuring robust performance across various malware categories.

1.5 Scope of the Project

1.5.1 Capabilities

The project has utilized SecureBERT model for textual malware classification by analyzing and predicting malware types which are based on contextual understanding of combined features and characteristics of different Android malware labels. The model is specifically trained on cybersecurity-related texts,websites, research papers, through which it understand the nuances of cybersecurity threats and behaviors, provide more sophisticated analysis than traditional methods. By utilizing the advanced contextual understanding capabilities of these models, the project aims to upgrade the accuracy and performance of malware detection through extensive experimentation with diverse datasets for training and fine-tuning in a significant manner. Additionally, the project explores the integration of these models into existing cybersecurity frameworks to assess their real-world applicability and effectiveness for textual datasets which ultimately aims to enhance cybersecurity defenses by providing a more reliable and advanced solution for detecting and mitigating malware threats

1.5.2 Limitations

The project may face several critical limitations that must be addressed for effective implementation. Firstly, the scalability and generalizability of the model in detecting a diverse range of malware variants and adapting to evolving threats require rigorous validation across varied datasets and real-world scenarios. Additionally, the substantial computational resources needed for training and deploying both models, especially for large-scale malware detection tasks, pose practical challenges. Ensuring the model's robustness and adaptability to new malware behaviors and evasion techniques is essential, necessitating continuous refinement and validation in diverse cybersecurity environments. These considerations are pivotal for enhancing the model's effectiveness and applicability in bolstering cybersecurity defenses against modern threats. Malware authors frequently develop new evasion techniques, such as sandbox detection, anti-debugging measures, and the use of advanced encryption and packing methods to avoid analysis. The robustness of a detection system is measured by its ability to withstand these techniques and still identify malicious behavior. Unable to account for these methods can render even the most sophisticated models ineffective, as malware becomes better at mimicking

legitimate software.

1.6 Potential Applications

Cybersecurity Operations Transformer-based models like SecureBERT can be utilized for text analysis by enhacing real-time threat detection and response, through which the accuracy of malware identification through text-based indicators will be improved. This will further strengthen the ability to protect sensitive data and digital infrastructure by mitigating identified threats in a swift manner.

1.6.1 Threat Intelligence

Continuous analyzing and classifying text data from new malware samples, will provide critical insights of the same, through which security professionals will be guided to counter emerging threats.

1.6.2 Forensic Analysis

The techniques developed can be crucial in forensic analysis of malware incidents,which focuses on text-based artifacts such as command and control messages, malware documentation, or malicious scripts. forensic investigators can gather evidence, reconstruct attack vectors, and aid in the legal prosecution of cybercriminals, by gathering textual elements.

1.6.3 Incident Response

During malware incidents, the project's text analysis capabilities can be rapidly deployed to assess and respond to threats. The detailed understanding of malware through text data can help security teams determine the nature of the malware, its potential impact, and the necessary steps for effective mitigation of the same.

1.6.4 Malware Research

Through text-based features, researchers in malware analysis can grip the outcomes of this project to advance their understanding of malware behaviors and evolution. The improved detection and analytical capabilities shall allow the discovery of new malware trends and the development of more robust defenses against future threats.

1.7 Originality of Project

We will be applying novel approach of SecureBERT model for malware classification based on context and understanding of the data. The model will be able to analyze and predict the type of malware by understanding the context through the text.The text comprises of combined features and characteristics of different android malware labels.SecureBERT model are specifically trained on cybsersecurity related texts, due to which the nuances related to cybersecurity threats and behaviour are understood by the model.

1.8 Organisation of Project Report

The material in this project report is organised into seven chapters. After this introductory chapter introduces the problem topic this project tries to address, chapter 2 contains the literature review of important and relevant publications, pointing toward a notable research gap. Chapter 3 tells us about the methodology for the implementation of this project. Chapter 4 provides an overview of what has been accomplished or obtained after the implementation. Chapter 5 contains some important discussions and analysis performed on the used model and methods during implementation. Chapter 6 covers conclusion of the project, which covers the requirements met for the project.Chapter 7 contains future enhancement which covers the potential requirement or methods the project has the capability to cover.

2 LITERATURE REVIEW

2.1 MeMalDet: A memory analysis-based malware detection framework using deep autoencoders and stacked ensemble under temporal evaluations

2.1.1 Author's work

Maniriho et al. (2024) developed MeMalDet, a memory analysis-based malware detection framework using deep autoencoders and a stacked ensemble approach. The model was evaluated under temporal settings, it demonstrated superior accuracy, reduced false positives, and improved ROC curve performance compared to other memory forensics methods. The framework excels in detecting ransomware but lacks extensive real-world qualitative analysis.[3]The study showed notable improvements in malware detection accuracy through memory analysis but could benefit from exploring scalability and adaptability for various malware types. Mohammad's work on machine learning for memory forensics suggests potential advancements, though more details on its quantitative and qualitative results would help evaluate its overall effectiveness.

Method/Architecture:

The authors implemented MeMalDet using Python with TensorFlow, Keras, and Scikit-learn, and used Jupyter Notebook for development which focuses on detecting malware through memory analysis by capturing behavioral characteristics during execution.

Quantitative and Qualitative Results Obtained: MeMalDet achieved an average accuracy of 99.78%, peaking at 99.85% with a 90/10 training/testing split. Precision, recall, and F1-score were high, with precision reaching 0.9993 for benign and 0.9976 for malware samples. The F1-score was 0.9985. The ROC and precision-recall curves showed near-perfect separability between malware and benign applications..

Strengths:

High Performance: The proposed technique showed high accuracy, precision, recall, and F1-scores, indicating robust performance in detecting malware.

Effective Memory Analysis: MeMalDet effectively utilized memory analysis to capture behavioral characteristics, which is crucial for detecting obfuscated and evolving malware.

Weakness:

Temporal Bias: The performance slightly gets degraded when tested with temporal splits.

Consumption: The study also evaluated the computational resources required, such as memory and running time which indicated the technique's resource demands during training and testing phases.

2.1.2 Research gap:

Existing frameworks struggle to adapt to the evolving malware landscape, with current research lacking real-time evaluation methods. Future studies should focus on enhancing temporal evaluation to ensure resilient and flexible malware detection which is crucial for advancing memory-based malware detection and cybersecurity.

2.2 A Transformer-Based Framework for Payload Malware Detection and Classification

2.2.1 Author's work

The paper presents a DPI algorithm using transformers to detect malicious traffic by analyzing raw payload bytes. Evaluated on the UNSW-NB15 and CIC-IOT23 datasets, the model achieved 79% accuracy in binary classification and 72% in multi-classification. The self-attention mechanism of transformers effectively learns complex patterns in packet content, showcasing their potential in intrusion detection systems.

Method/Architecture:

The authors introduced a transformer-based model for deep packet inspection that captures contextual patterns in network packets. The architecture features an embedding layer, transformer blocks with multi-head self-attention, and a classification output layer, leveraging self-attention to assess the importance of different packet sequence parts.

Quantitative and Qualitative Results Obtained:

For the UNSW-NB15 dataset, the transformer-based method achieved 79.57% accuracy and F1-Score, outperforming 1D-CNN (74.98%), 2D-CNN (75.56%), and LSTM (71.65%). On the CIC-IOT23 dataset, it attained 79.07% accuracy and 81.25% F1-Score, surpassing 1D-CNN (75.30%) and LSTM (71.60%).

Strengths:

The transformer-based model excels in capturing long-range dependencies and con-

textual information in network packets, resulting in high performance for intrusion detection.

Weakness:

The model's recall is slightly lower than 2D-CNN, which shows there is potential issues with false negatives that need addressing

2.2.2 Research gap

While the study highlights the potential of transformer-based models for detecting payload malware, further research is needed to develop interpretable models that balance accuracy and complexity.

2.3 MADRAS-NET: A deep learning approach for detecting and classifying android malware using Linknet

2.3.1 Author's work

The model MADRAS-NET advances Android malware classification using a deep learning approach with Linknet architecture and novel feature extraction. It effectively detects and classifies malware by analyzing app features from a large dataset.[4] MADRAS-NET excels in Android malware classification through effective semantic segmentation using Linknet, capturing detailed spatial relationships and subtle patterns in malware features. It performs robustly across diverse malware samples, enhancing detection accuracy. However, its complexity limits interpretability, which can affect adoption in contexts requiring transparent decision-making. This work contributes to ongoing research in using deep learning for malware detection.

Method/Architecture:

The paper proposes the **MADRAS-NET** methodology for Android malware detection. The **MADRAS-NET** architecture utilizes the **AndMal-2020** dataset for training and evaluation. It incorporates both **static features** (like Android malware flows, permissions, and intents) and **dynamic features** (like API calls and log files).

Quantitative and Qualitative Results Obtained:

The model was tested on 400,000 Android apps (200,000 benign and 200,000 malicious), achieved 99.59% accuracy and a superior AUC of 0.997 compared to LSTM, GAN, and DBN. It effectively restores spatial information lost during encoding through the

LinkNET architecture, using fewer parameters, making it suitable for real-time applications.

Strengths:

- High accuracy (99.59%) in malware detection.
- Better performance compared to traditional methods like **LSTM**, **GAN**, and **DBN**.
- High AUC of 0.997 indicating excellent classification performance.

Weakness:

- The paper primarily focuses on the **AndMal-2020** dataset, which may limit generalization to other datasets.
- There is an emphasis on the need for further improvement in real-time attack detection in SDN networks.

2.3.2 Research gap

The existing work has made significant strides in Android malware detection, there still remains a need for more interpretable and lightweight models which could balance accuracy with resource in an efficient manner. Additionally, exploring adversarial robustness and transferability across platforms could be a promising avenue for future research.

2.4 Automated malware detection using machine learning and deep learning approaches for android applications

2.4.1 Author's work

R. Mahalakshmi's MAD-NET for Android malware detection uses Deep Belief Networks (DBN) to achieve 99.83% accuracy, outperforming techniques like ANN, GAN, and LSTM. Its strengths are high accuracy and effective DBN use, but it faces challenges with interpretability and overfitting. S. Poornima's study reviews Android malware detection methods using machine learning, providing useful insights but lacking empirical results. Its strength is the comprehensive overview, but it is limited by the absence of quantitative data.

Method/Architecture:

The authors proposed a MAD-NET technique for detecting and classifying malware in Android devices. The process involves:

Feature Extraction Data from CICAndMal2017 datasets are classified into signature-based and behavior-based data.

Classification Phase:

- Features are converted to sequence data and fed into the classification phase using the Deep Belief Network (DBN) technique.
- Malware is classified as either benign or malicious, and detection reports are generated.

Quantitative and Qualitative Results Obtained:

Quantitative Results

The performance of various machine learning algorithms was evaluated using metrics such as Accuracy, Precision, Recall, F1 Score, and AUC (Area Under the Curve). The study included:

The dataset used in the study comprised 5000 markers, including 426 malicious and 5065 benign samples. It included a total of 2126 samples and recorded 2,583,878 network hits. Each sample in the dataset was characterized by 84 network flow features, providing a comprehensive basis for analyzing the behavior and classification of malicious versus benign activities.

Performance Metrics: Evaluated using K-Nearest Neighbor (KNN), Decision Tree, Random Forest, Naive Bayes, SVM, and CNN. Validation metrics for malware classification included:

- **KNN:** Accuracy 95.59%, Precision 94.59%, Recall 96.89%, F1 Score 95.87%
- **Decision Tree:** Accuracy 97.78%, Precision 96.65%, Recall 96.34%, F1 Score 97.59%
- **Random Forest:** Accuracy 95.56%, Precision 97.05%, Recall 97.87%, F1 Score 98.45%

- **Naive Bayes:** Accuracy 98.00%, Precision 98.90%, Recall 97.56%, F1 Score 97.54%
- **SVM:** Accuracy 97.65%, Precision 98.99%, Recall 99.45%, F1 Score 98.87%
- **CNN:** Accuracy 98.99%, Precision 99.37%, Recall 98.69%, F1 Score 99.12%

Qualitative Results

- The study highlighted the effectiveness of feature extraction from APK files, permissions, intentions, suspicious API calls, and dynamic analysis for detecting malware.
- Techniques like signature-based detection were noted to have limitations in handling dynamic malware.

Strengths:

The study's strengths lie in its comprehensive feature extraction approach by utilizing both static and dynamic analysis to capture a broad spectrum of relevant data. Additionally, the evaluation was conducted using a wide range of machine learning algorithms, producing robust performance metrics that enhance the reliability and validity of the classification results.

Weakness:

A major weakness of the study is its dependency on the quality and quantity of the dataset, which could limit the generalizability and accuracy of the results.The reliance on signature-based detection may not be effective against new or evolving malware, as these methods often struggle to identify threats that have not been previously encountered or are relatively new.

2.4.2 Researchgap

The field of Android malware detection has made progress, but there's a demand for interpretable and resource-efficient models. Investigating adversarial robustness and cross-platform transferability could be valuable for future research.

2.5 MalBERTv2: Code Aware BERT-Based Model for Malware Identification

2.5.1 Author's work

Rahali and Akhloufi (2023) introduced MalBERTv2, an advanced version of the MalBERT model for malware detection. It features a comprehensive pipeline for code-aware feature extraction and is pre-trained on a large dataset of Android apps, followed by fine-tuning for malware detection. MalBERTv2, with its pre-tokenization and feature extraction modules, showed superior performance in identifying malware, outperforming several leading detection systems. Its success is attributed to its robust feature extraction and pre-tokenization processes, validated by extensive evaluations on diverse datasets.

Method/Architecture:

The authors proposed MalBERTv2 which is an enhanced version of MalBERT for malware classification and is based on the BERT architecture. The model is trained from scratch on a dataset of malware and goodware samples, it features a multilevel classifier with distinct training and testing phases. The model involves fine-tuning BERT on the dataset and adding a classification layer on top.

Quantitative Results:

The quantitative results show that MalBERTv2 significantly outperforms other baseline models in terms of accuracy, F1-score, MCC (Matthews correlation coefficient), precision, recall, and AUC (Area Under the Curve) across multiple datasets. The key results from the test set are:

- **TFIDF + SVM**:
  - Accuracy: 0.9696 (MixG-Androzoo), 0.8588 (MixG-VirusShare)
  - F1-score: 0.9698 (MixG-Androzoo), 0.8571 (MixG-VirusShare)
- **Fasttext + CNN**:
  - Accuracy: 0.9535 (MixG-Androzoo), 0.8016 (MixG-VirusShare)
  - F1-score: 0.9554 (MixG-Androzoo), 0.7633 (MixG-VirusShare)
- **MalBERT**:
  - Accuracy: 0.9757 (MixG-Androzoo), 0.9240 (MixG-VirusShare)

  - F1-score: 0.9762 (MixG-Androzoo), 0.9247 (MixG-VirusShare)

- **MalBERTv2** (not explicitly listed but implied to be higher than MalBERT):
  - Improves accuracy and weighted F-measure compared to other baselines, with accuracy ranging between 0.8224 and 0.9376 for different datasets.

Qualitative Results:

Qualitative analysis highlights that MalBERTv2's preprocessing technique enhances feature representation, helping the model to focus on relevant patterns. The study also discusses the quality of the datasets used, ensuring that only reliable datasets were selected and analyzed. A knowledge graph was applied to the output text generated by the feature generator to assess dataset quality and identify potential issues or biases.

Strengths and Weaknesses:

Strengths:

- **Performance**: MalBERTv2 achieves higher accuracy and better overall performance compared to other models like TFIDF + SVM, Fasttext + CNN, and MalBERT.
- **Feature Representation**: The use of a new pre-tokenization technique and improved preprocessing enhances the model's ability to represent and focus on the most relevant features.

Weaknesses:

The model's training and preprocessing techniques are complex and might require significant computational resources and expertise to implement and fine-tune effectively.

2.5.2 Researchgap

The research gap highlights the need for improvements in generalization, interpretability, and adaptability in deep learning-based malware detection. While MalBERTv2 effectively identifies malware and outperforms existing models, forthcoming work should focus on creating more robust models that handle obfuscated malware, enhance interpretability, and adapt to evolving threats to maintain high accuracy and efficiency.

3 METHODOLOGY

3.1 Theoretical Formulations

BERT language model is an open source machine learning framework used for natural language processing (NLP). It is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context between words. The framework was pretrained using text from Wikipedia and can be fine-tuned with question-and-answer data sets. BERT, stands for Bidirectional Encoder Representations from Transformers, it is based on transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection.[5]

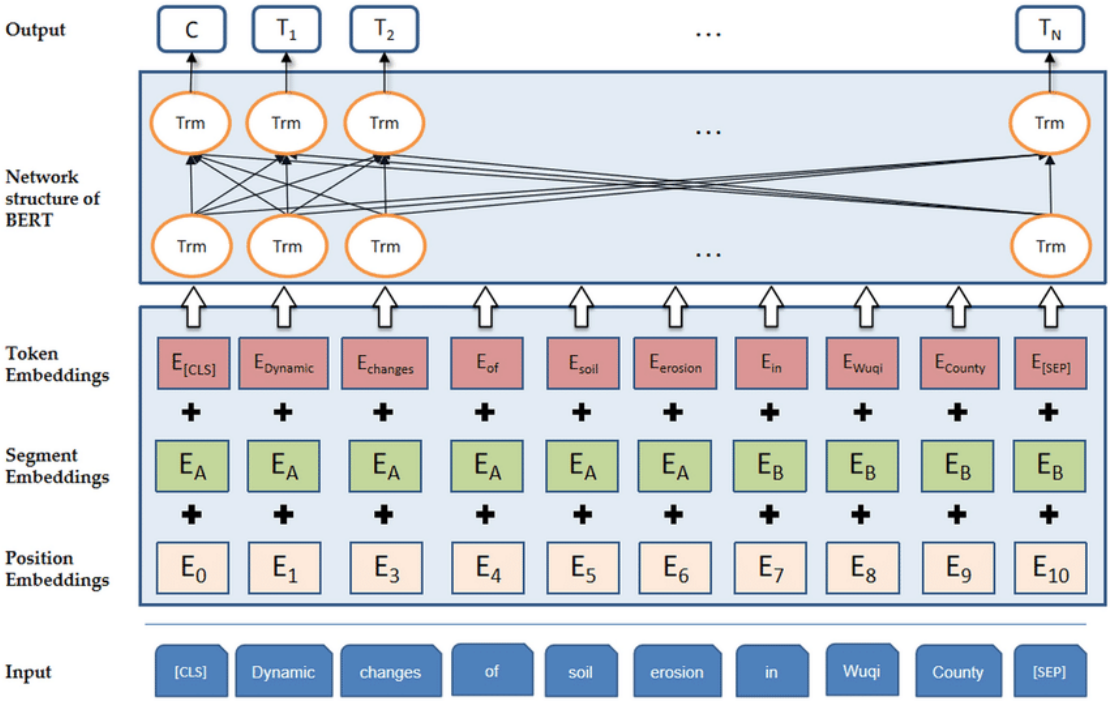


Figure 3.1: Overall Structure of BERT Model
,Source: Adapted from [6]

BERT (Bidirectional Encoder Representations from Transformers)

- **Developed by:** Google
- **Published in:** 2018
- **Architecture:** Transformer-based model using the encoder mechanism

- **Training Objective:**
  - *Masked Language Modeling (MLM):* Randomly masks some tokens in the input, then trains the model to predict the masked tokens.
  - *Next Sentence Prediction (NSP):* Trains the model to understand the relationship between two sentences by predicting whether a given sentence B follows sentence A in the text.
  - *Optimization:* BERT uses smaller training batch and fewer training steps for optimization.

RoBERTa (A Robustly Optimized BERT Pretraining Approach)

- **Developed by:** Facebook AI
- **Published in:** 2019
- **Architecture:** Transformer-based model, essentially a modified BERT
- **Training Objective:** Focuses solely on the MLM task with several optimizations.
  - *No NSP Task:* Removes the NSP objective from BERT's training procedure, arguing it doesn't contribute significantly to performance.
  - *Dynamic Masking:* Applies masking patterns to the input dynamically during training rather than using a fixed masking pattern.
  - *Training Data and Duration:* Trained on much more data (160GB compared to BERT's 16GB) and for a longer time, with larger batch sizes and learning rates.
  - *Hyperparameter Tweaks:* Implements several training tweaks, including longer training periods, larger batch sizes, and varying the learning rate schedule.
  - *Optimization:* The model use mini-batches and more training steps, with modifications in the learning rate schedule and the optimizer's parameters during optimization of model.[7]

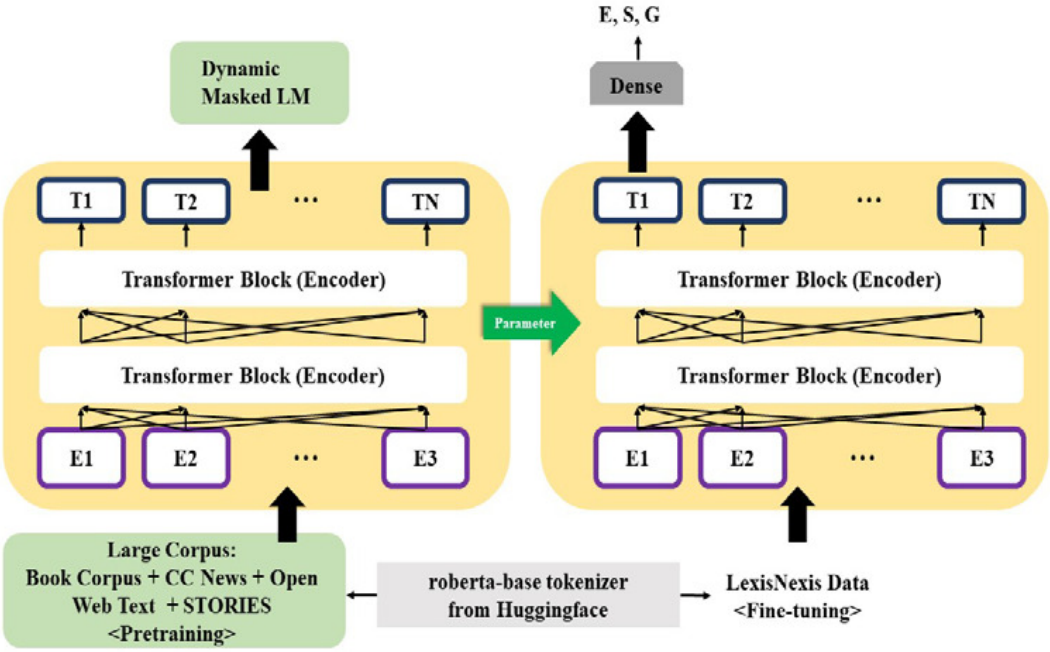


Figure 3.2: Roberta Model
, Source Adapted from [8]

SecureBERT

SecureBERT, is a variant of the BERT (Bidirectional Encoder Representations from Transformers) model, which is specifically designed for security-related tasks. BERT itself is a transformer-based model that has been pre-trained on a vast amount of text data, enabling it to comprehend the context of words within sentences. Its bidirectional approach allows it to capture information from both preceding and following words, enhancing its understanding of language semantics. In the world of Android malware analysis, SecureBERT utilizes BERT's inherent strengths in natural language understanding and tailors them for security applications. This involves fine-tuning SecureBERT on a dataset comprising malware-related texts specific to Android environments. Through this process, SecureBERT learns to identify patterns and characteristics indicative of malware, thereby aiding in the detection and classification of malicious applications and behaviors targeting Android devices.

3.1.1 Architecture Details

Base Model

SecureBERT builds upon the RoBERTa-base architecture, which consists of:

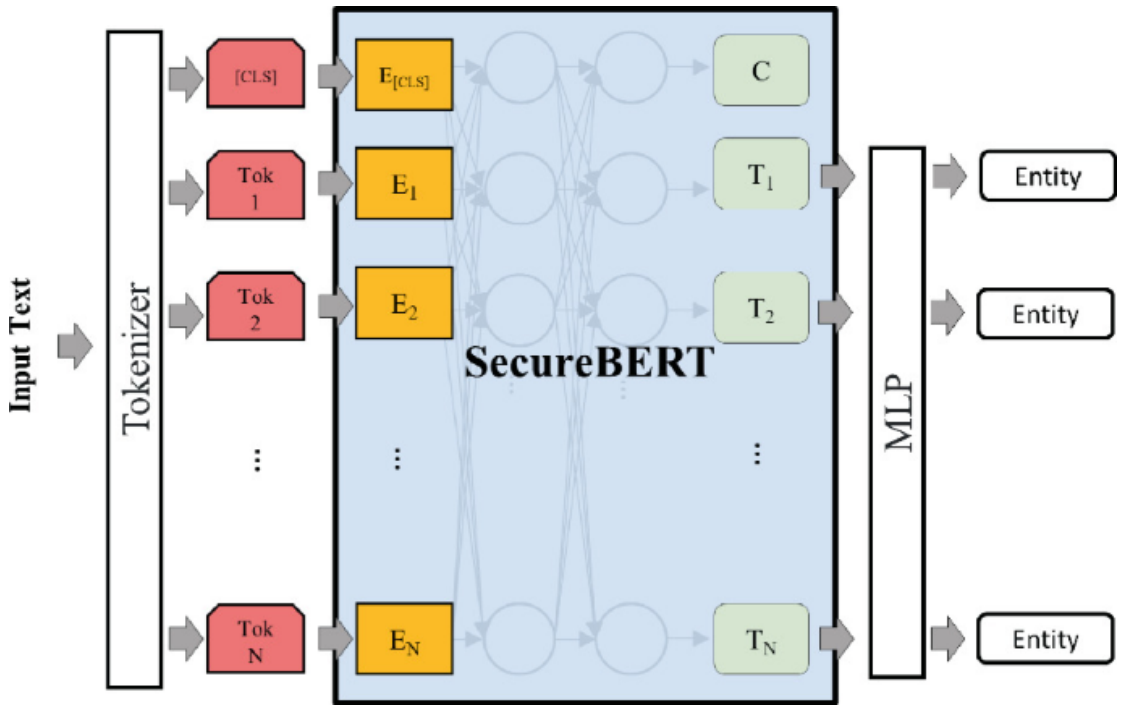


Figure 3.3: SecureBERT Architecture
, Source: Adapted From [9]

- **Input Layer:** This layer processes tokenized input data.
- **Transformer Layers:** SecureBERT features 12 hidden transformer layers, which utilize self-attention mechanisms to capture contextual relationships within the text.

Custom Tokenizer

A specialized tokenizer is employed, tailored for cybersecurity terminology. This tokenizer expands the vocabulary to approximately 50,265 tokens, allowing SecureBERT to effectively handle domain-specific language and jargon.

Training Data

The model is trained on a substantial dataset comprising 98,411 cybersecurity-related textual elements, totaling around 1 billion tokens. This extensive training enables SecureBERT to understand and generate relevant responses in the cybersecurity domain.

Masked Language Modeling (MLM)

SecureBERT employs MLM techniques during training, which have been shown to outperform other models like RoBERTa and SciBERT in predicting masked words within cybersecurity texts. This capability is crucial for tasks such as threat detection

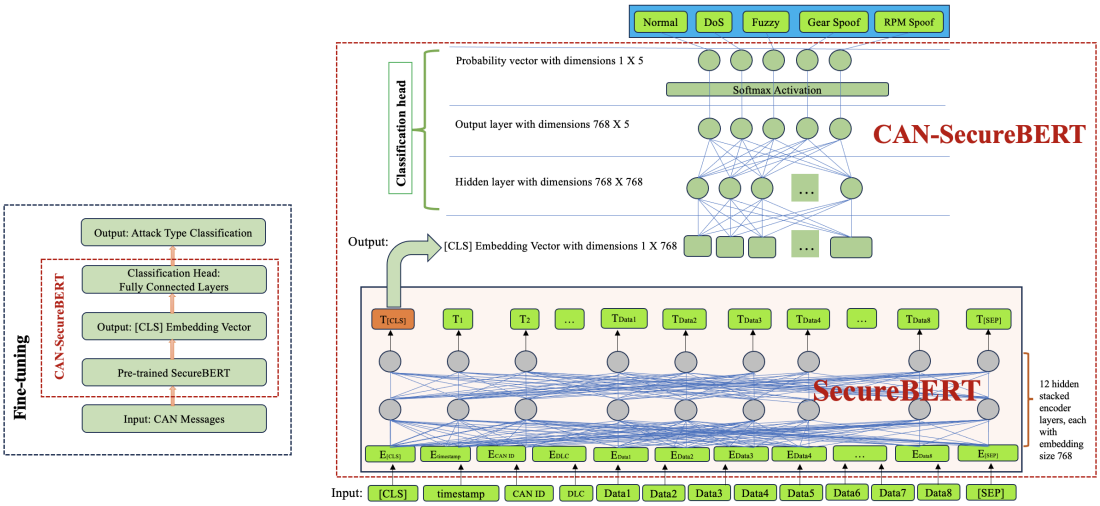


Figure 3.4: SecureBERT Architecture Details
Source: Adapted from [9]

Table 2.1: The statistics of collected cybersecurity corpora for training the SecureBERT.

Type	No. Documents
Articles	8,955
Books	180
Survey Papers	515
Blogs/News	85,953
Wikipedia (cybersecurity)	2,156
Security Reports	518
Videos (subtitles)	134
Total	**98,411**

Vocabulary size	1,674,434 words
Corpus size	1,072,798,637 words
Document size	2,174,621 documents (paragraphs)

Table 2.2: The resources collected for cybersecurity textual data.

Websites
Trendmicro, NakedSecurity, NIST, GovernmentCIO Media, CShub, Threatpost, Techopedia, Portswigger, Security Magazine, Sophos, Reddit, FireEye, SANS, Drizgroup, NETSCOUT, Imperva, DANIEL MIESSLER, Symantec, Kaspersky, PacketStorm, Microsoft, RedHat, Tripwire, Krebs on Security, SecurityFocus, CSO Online, InfoSec Institute, Enisa, MITRE
Security Reports and Whitepapers
APT Notes, VNote, CERT, Cisco Security Reports , Symantec Security Reports
Books, Articles, and Surveys
Tags: cybersecurity, vulnerability, cyber attack, hack
ACM CCS: 2014-2020 , IEEE NDSS (2016-2020), IEEE Oakland (1980-2020) ACM Security and Privacy (1980-2020), Arxiv , Cybersecurity and Hacking books
Videos (YouTube)
Cybersecurity courses, tutorial, and conference presentations

Figure 3.5: SecureBERT Datasets
, Source: Adapted from [10]

and risk assessment.

Parameter Count

SecureBERT contains approximately 123 million parameters, making it a robust model for processing complex cybersecurity information. **Noise Injection**

During training, noise is introduced into the token weights of the vocabulary to enhance the model's robustness and adaptability to varied cybersecurity contexts.

3.1.2 Supporting Pre-processing Steps

The dataset, comprisses of text messages and their labels, which was loaded and assigned appropriate column names. Labels were cleaned by removing whitespace and converting them to lowercase to prevent mismatches. Text labels were then converted to integer labels using a predefined mapping. Any rows with unmapped or negative labels were removed. The data was split into training, validation, and test sets. Texts were tokenized

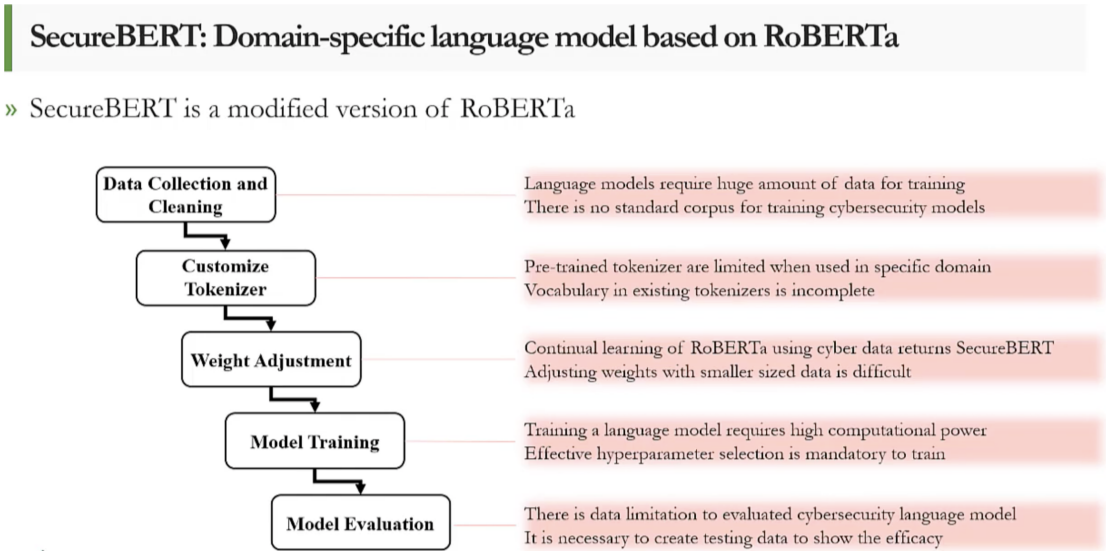


Figure 3.6: SecureBERT-Roberta

into IDs, with padding or truncation to a fixed length as needed. Finally, PyTorch datasets were created for training, validation, and testing by combining the tokenized texts with their labels.

3.1.3 Post-processing Steps

In post processing the model is evaluated on the test set to obtain predictions. Key evaluation metrics, including accuracy, precision, recall, and F1 score, are computed. A confusion matrix is generated to visualize model performance across different classes, and a detailed classification report provides precision, recall, and F1 scores for each class. Additionally, an interactive prompt is available for users to input text messages and receive predictions from the trained model, continuing until the user chooses to exit.

3.1.4 Major Benefits of the Chosen Technique

- **Contextual Understanding:**SecureBERT is a transformer encoder based models which understand the context of words in a sentence, leading to more accurate classification of malware-related text.
- **Finetuning Capabilities:** The model can be finetuned on specific datasets which makes the model adaptable to various cybersecurity tasks.
- **High Performance:** BERT models have demonstrated state-of-the-art perfor-

mance in various NLP related tasks which includes text classification.

- **Robustness:** The model architecture handles noisy and imbalanced data in an effective manner as compared to traditional methods.

3.1.5 Assumptions Taken into Account

- **Quality of data:** The dataset used for training, testing and validation is represented as real world data ,the model can enter.
- **Label Accuracy:** The labels in the dataset are accurate and they correctly represent the categories.
- **Tokenization Efficiency:** The tokenizer converts text into token IDs without significant loss of information or data loss.
- **Model Generalization:** The model seems to generalize well to unseen data after being fine-tuned on the training dataset.
- **Hardware Availability:** Computational Resources are available to train model efficiently in a reasonable timeframe.
- **Consistent Data Preprocessing:** The preprocessing steps consistently cleans and prepares the data.

3.2 Mathematical Modelling

Pre-processing Equations The pre-processing steps involve converting textual data into vector embeddings using techniques such as Word2Vec. The mathematical equation for Word2Vec can be represented as follows: Word2Vec is a widely used method in natural language processing (NLP) that allows words to be represented as vectors in a continuous vector space. Word2Vec is an effort to map words to high-dimensional vectors to capture the semantic relationships between words, developed by researchers at Google. Words with similar meanings should have similar vector representations, according to the main principle of Word2Vec. Word2Vec utilizes two architectures: **CBOW (Continuous Bag of Words):** The CBOW model predicts the current word given context words within a specific window. The input layer contains the context

words and the output layer contains the current word. The hidden layer contains the dimensions we want to represent the current word present at the output layer .

$$\text{Word2Vec}(w_i) = \sum_{\substack{j=1 \\ j \neq i}}^{n} P(w_j|w_i) \tag{3.1}$$

Where:

- w_i represents the input word.
- $P(w_j|w_i)$ represents the conditional probability of word w_j given word w_i[11].

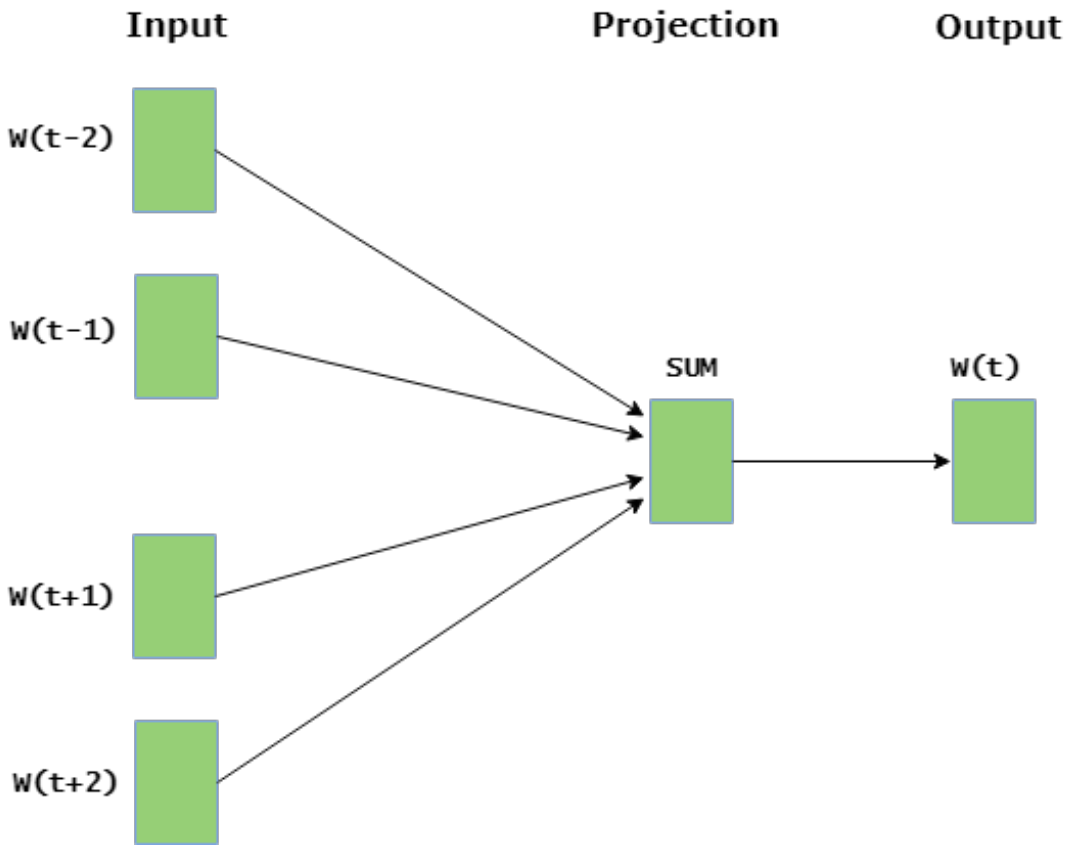


Figure 3.7: Word to Vector form

Label Encoding: Given a categorical feature x with m categories, each category c_i is assigned a unique integer label l_i, where i ranges from 1 to m. The label encoding for category c_i is represented as:

$$[l_i = i] \tag{3.2}$$

where i represents the index of the category.[12] Remove Unmapped Labels
To remove entries with missing labels:

$$\mathbf{D} = \mathbf{D} \setminus \{(t_i, l_i) \mid l_i = \text{NaN}\} \tag{3.3}$$

Check Negative Labels

To ensure that all labels are non-negative:

$$\text{assert}\,(\forall l_i \in \mathbf{D},\ l_i \geq 0) \tag{3.4}$$

Mathematical Foundation of SecureBERT SecureBERT are variants of the Roberta model adapted for cybsersecurity domain tasks, utilizes several key mathematical components: **Token Embeddings**

Attention Mechanism The self-attention mechanism calculates attention scores as follows:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.5}$$

where Q is the query matrix, K is the key matrix, V is the value matrix, and d_k is the dimension of the keys. SecureBERT begins by embedding input tokens t_i into dense vectors $\mathbf{e}_i$ using an embedding matrix $\mathbf{E}$:

$$\mathbf{E} = \text{Embedding}(T) \tag{3.6}$$

Transformer Encoder

SecureBERT employs multiple transformer encoder layers to process embeddings from previous layers into new representations $\mathbf{H}^{(l)}$:

$$\mathbf{H}^{(l)} = \text{Transformer}^{(l)}(\mathbf{H}^{(l-1)}) \tag{3.7}$$

where $\mathbf{H}^{(0)} = \mathbf{E}$ and $\text{Transformer}^{(l)}$ denotes the l-th transformer layer. **Contextualized Representations**

After L layers of transformers, the final output $\mathbf{C}$ represents contextualized embeddings capturing semantic meanings:

$$\mathbf{C} = \mathbf{H}^{(L)} \tag{3.8}$$

Classification Head

Both models utilize classification head to extract information from the special [CLS] token representation $\mathbf{z}$:

$$\mathbf{z} = \text{CLS}(\mathbf{C}) \tag{3.9}$$

Logits Calculation

The classification head outputs logits $\mathbf{o}$ using a linear transformation $\mathbf{W}$ and bias $\mathbf{b}$:

$$\mathbf{o} = \mathbf{W}\mathbf{z} + \mathbf{b} \tag{3.10}$$

Postprocessing Equations:

Softmax Activation:

Finally, logits are passed through a softmax function to obtain class probabilities $\hat{y}$:

$$\hat{y} = \text{softmax}(\mathbf{o}) \tag{3.11}$$

Prediction:

The class with the highest probability is selected. The equation for the predicted class is:

$$\text{Predicted Class} = \arg\max(p_i) \tag{3.12}$$

where p_i is the probability of class i. The models are rained by optimizing parameters $\mathbf{E}, \mathbf{W}$, and $\mathbf{b}$ using backpropagation and gradient descent, adapting the pre-trained BERT model effectively for security-related tasks. **Mathematical Foundation of LoRA (Low-Rank Adaptation) Introduction** Low-Rank Adaptation (LoRA) is a technique designed to efficiently fine-tune large pre-trained models by introducing a small number of trainable parameters into the model while keeping the majority of the original model weights frozen. **Model Parameterization** Consider a pre-trained model with weight matrix $W \in \mathbb{R}^{d \times k}$, where d is the input dimension and k is the output dimension. In standard fine-tuning, all elements of W would be updated. However, in LoRA, we decompose W into:

$$W = W_0 + \Delta W$$

where:

- W_0 is the original pre-trained weight matrix, kept fixed during fine-tuning.
- ΔW is the low-rank update matrix.[13]

Low-Rank Decomposition The low-rank update matrix ΔW is decomposed into the

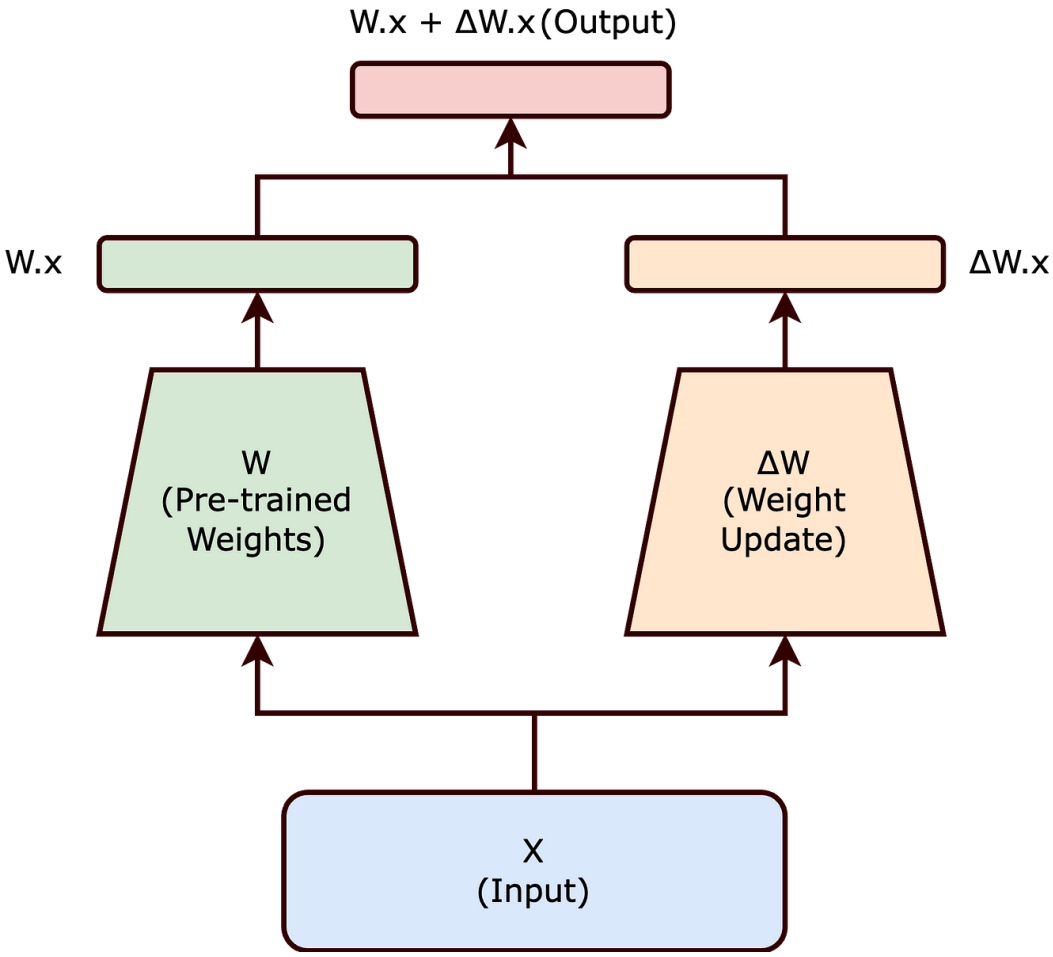


Figure 3.8: LoRA Process
Source: Adapted from [14]

product of two smaller matrices:

$$\Delta W = AB$$

where:

- $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$ are the low-rank matrices.
- r is the rank of the approximation, with $r \ll \min(d,k)$.

This decomposition significantly reduces the number of trainable parameters from $d \times k$ to $r \times (d + k)$. **Training Objective** The training objective in LoRA is to optimize the low-rank matrices A and B while keeping W_0 fixed. The loss function $\mathscr{L}$ for training can be expressed as:

$$\mathscr{L} = \mathscr{L}_{\text{task}}(X, Y; W_0 + AB) + \lambda (\|A\|_F^2 + \|B\|_F^2)$$

where:

- $\mathscr{L}_{\text{task}}$ is the task-specific loss function (e.g., cross-entropy loss for classification).
- X and Y represent the input data and target labels, respectively.
- $\|\cdot\|_F$ denotes the Frobenius norm.
- λ is a regularization parameter that controls the contribution of the low-rank matrices' norms to the loss function.

Computational Efficiency LoRA reduces the number of additional parameters and computational cost as follows:

$$\text{Additional Parameters} = r \times (d + k)$$

This is significantly smaller compared to the full parameter count $d \times k$, leading to faster and more memory-efficient training. **Gradient Computation** During training, the gradients of the loss function with respect to A and B are computed using backpropagation.[15]

3.3 System Block Diagram

Block Explanations

Explanation of each phases are given below.

1. Data Preprocessing

- **Textual Datasets**: The datasets are in textual format which are mixed from various sources of android malware dataset..
- **Data Cleaning**: Noise and irrelevant information from the raw data are removed to improve the quality of data..

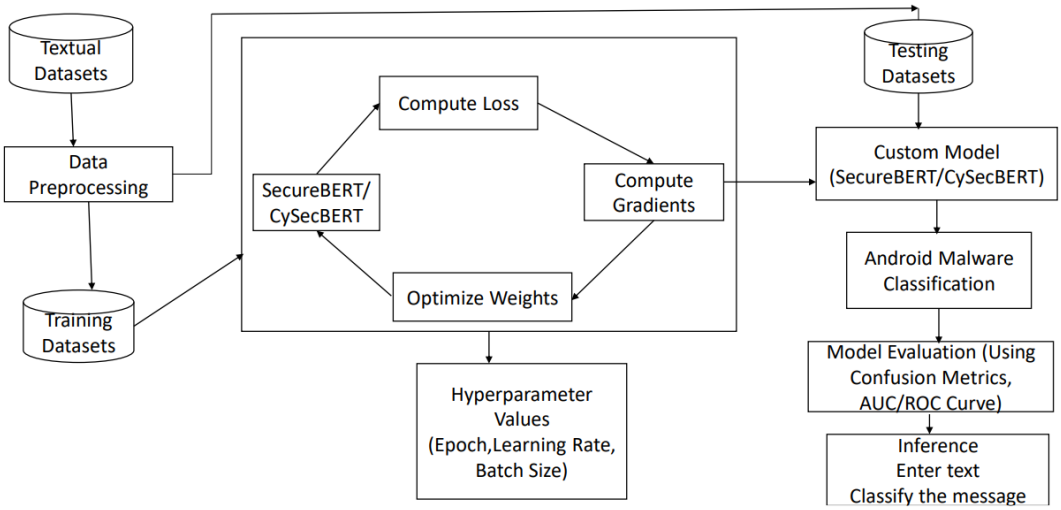


Figure 3.9: Block Diagram for Android Malware Analysis using SecureBERT

- **Data Normalization**: The format of the data and features are standardised and normalized to ensure consistency.
- **Label Encoding**: categorical labels are converted into numerical values for compatibility with machine learning models.
- **Splitting Datasets**: The dataset is divided into training, validation, and test sets to ensure the evaluation takes place in a robust manner.
- **Create Structured Data**: The data is formatted into a structure that can be used by the model by Pytorch.

2. Model Training and finetuning

- **Load Pretrained Model**: SecureBERT, which is a pretrained model is loaded.
- **Finetune Model**: The pretrained model is finetuned and adjusted in order for the specific task using techniques like LoRA (Low-Rank Adaptation).
- **Model Training**: The model is trained on the prepared dataset which involves:
  - **Hyperparameter Grid**: Model has been experimented with different hyperparameters (e.g., learning rate, batch size) to find the best configuration.

- **Epoch**: The number of passes the model makes through the entire training dataset is defined.
- **Learning Rate**: Step size for each iteration is defined to minimize the loss function.
- **Batch Size**: The number of samples processed before updating the model's parameters is determined.

3. Model Evaluation and Inference

- **Model Prediction**: The trained model is leveraged to make predictions on unseen data (test set).
- **Metrics Calculation**: The model's performance is evaluated using various metrics like accuracy, precision, recall, and F1-score.
- **Confusion Matrix**: The performance of the classification model is visualized by showing true positives, false positives, true negatives, and false negatives.
- **ROC-AUC Curve**: The Receiver Operating Characteristic (ROC) curve and The Area Under the Curve (AUC) is calculated to assess the model's ability to distinguish between classes.
- **Enter Text Message**: Text message is input into the model for prediction.
- **Label Prediction**: The model outputs a prediction label, Which classifies the text message according to the labels defined.

3.4 Instrumentation Requirements

3.4.1 Hardware & Software Tools:

- **Hardware**: The project will require a device with specifications like 16 GB RAM DD4 and a 12th Gen Intel(R) Core(TM) i5-1235U processor running at 1.30 GHz with graphics Intel® Iris Xe Graphics.
- **Essential Software Tools**: Essential software tools will include:

- **Programming Environments:**
  - **Python**: The primary programming language used for developing and implementing the model.
- **Machine Learning Frameworks:**
  - **TensorFlow,PyTorch**: These frameworks will be utilized for building and training the transformer-based model.
- **Feature Extraction and Data Cleaning/Organizing Libraries:**
  - **Scikit-learn**: Used for feature extraction, data preprocessing, and applying machine learning algorithms.
  - **Pandas**: Utilized for data manipulation and analysis.
- **Transformer-Based Libraries:**
  - Libraries specifically designed for transformers, such as **Hugging Face's Transformers**, will be employed for implementing and fine-tuning SecureBERT model.
- **Development Environment:**
  - **Google Colab**: A cloud-based environment that will be used for coding, executing, finetuning, quantization (if required) and experimenting with the model, providing access to powerful computational resources.
  - **Kaggle**: Kaggle provides a cloud-based computational environment called Kaggle Notebooks which enables reproducible and collaborative analysis using pre-configured Jupyter Notebooks with popular data science libraries.It offer a convenient cloud-based setup, so that users can also develop locally using virtual environment tools like Conda, Virtualenv, and Poetry. It allows users to leverage cloud-based CPU and GPU resources to train and evaluate machine learning models, enabling computations beyond local hardware limitations.

3.4.2 Purpose of the Device:

The device will serve as the computational platform for running complex computations involved in training and testing machine learning models for the project.

3.4.3 Tentative Device Version/Type:

The specified device with 16 GB RAM and a 12th Gen Intel(R) Core(TM) i5-1235U processor aligns well with the computational requirements for training and testing machine learning transformer based models for malware analysis.

3.4.4 Access to the Device:

Access to the device will be obtained through personal ownership. By going through the specified device with suitable hardware specifications and essential software tools.

3.5 Dataset Explanation

3.5.1 Malware Definitions:

Ransomware

Ransomware is a type of malware that restricts access to a victim's data, it demands payment to restore that access. In the context of Android malware, ransomware manifests in two primary forms: **Lock-Screen Ransomware**

This variant blocks access to the device by displaying a message that claims the user has violated laws, it often demands payment to unlock the device. This type of ransomware may not encrypt files but instead prevents the user from using their device altogether. **Crypto-Ransomware**

It is more sophisticated than lock-screen variants, it encrypts the user's files, making them inaccessible without a decryption key. Victims are then instructed to pay a ransom which is usually in cryptocurrency, to regain access to their data.

Scareware

Its aim is to manipulate and scare users into believing their system is infected with a virus compromised. It generates fake-alerts, popus or warnings. **Trojan**

Trojans are malicious codes that can perform harful activities such as stealing data credentials. It disguises itself as legitimate software in order to trick user for installing or using it .

Worm

Worm can spread itself automatically from one device to another without requiring much of user interaction. worms do not need to attach themselves to other

programs to propagate it. It can spread independently over a network or through messaging platforms like WhatsApp.

Keylogger

keylogger is a type of spyware which is designed to monitor and record every keystroke made on an Android device.

SMSMalware

SMS Malware is a type of malware which exploits vulnerabilities in SMS systems on mobile devices.

Polymorphic

Polymorphic is a type of virus which constantly modifies its code but retains the same functionality which makes antivirus difficult to detect .

Downloader

Downloader refers to a type of malicious software that is designed to download while leads to installation of additional malware onto a device after it has been initially installed.

Fake App

A fake app is an unauthorized application that mimics the appearance and functionality of a legitimate app, the intention of the creation is malicious. These apps are designed to deceive users into believing they are authentic, allowing cybercriminals to carry out various malicious activities. **Cryptojacker**

A cryptojacker hijacks the computing resources of a mobile device to mine cryptocurrency without the permission of a user.

Adware

Adware displays unwanted advertisements and pop-ups on infected devices, the user's knowledge or permission. Revenue is generated for its creators through these intrusive marketing campaigns.

Spyware

Spyware is a type of malicious software designed to monitor and collect sensitive information from a user's device without their knowledge or consent. It operates covertly, often tracking user activities and exfiltrating personal data such as login credentials, financial information, messages, and browsing history.

Datasets description: The chosen datasets, Androzoo, Drebin,CicMaldroid2017,trojan,

ransomware, Tuandromd, are highly relevant datasets which are used in the project as they collectively provide a comprehensive collection of Android malware samples and benign applications. These datasets are widely used in the research community for malware analysis and have been extensively validated in various studies. By using these datasets, the project ensures a robust and diverse representation of malware behaviors and characteristics, enabling the development of an effective malware detection model.

3.5.2 MalwareTextDB

MalwareTextDB is a comprehensive database which is specifically designed to provide annotated malware articles. It is an essential resource for researchers and practitioners in the field of cybersecurity, particularly those working on natural language processing (NLP) and text mining related to malware analysis and detection.

Key Features of MalwareTextDB

- **Extensive Collection**: MalwareTextDB contains a vast collection of articles related to malware, gathered from various reputable sources. These articles cover a wide range of topics, including malware types, attack vectors, mitigation strategies, and real-world case studies.
- **Annotated Data**: One of the standout features of MalwareTextDB is its annotated nature. Each article is meticulously annotated with relevant metadata, such as the type of malware discussed, attack techniques, affected systems, and more.
- **High-Quality Data**: The articles included in MalwareTextDB are curated to ensure high quality and relevance. They are sourced from trusted publications, academic journals, cybersecurity blogs, and official reports from cybersecurity organizations.
- **Diverse Malware Types**: The database covers a diverse range of malware types, including but not limited to viruses, worms, trojans, ransomware, spyware, adware, and advanced persistent threats (APTs). This diversity is crucial for training robust machine learning models.

- **Application in NLP and Machine Learning**: MalwareTextDB is particularly valuable for training NLP models to understand and detect malware-related content. Researchers can use this database to develop and fine-tune models for various applications, such as malware classification, threat intelligence extraction, and automated report generation.
- **Open Access and Community Contribution**: MalwareTextDB is designed to be an open-access resource, encouraging contributions from the cybersecurity community.
- **Structured Format**: The data in MalwareTextDB is provided in a structured format, making it easy to integrate with existing data pipelines and machine learning frameworks. This structure includes fields for article titles, publication dates, authors, abstracts, full texts, and annotations.
- **Real-World Applicability**: The database supports real-world applicability by offering insights into how malware is described and discussed in various contexts.[16]

AndroZoo Description: AndroZoo is one of the largest datasets of Android applications (APK files) collected from various sources, including the Google Play Store, third-party markets, research projects, and various internet repositories.

- **Size and Diversity:** AndroZoo contains millions of APK files, making it one of the most extensive datasets available for Android application analysis. This large size and diversity allow researchers to study a wide range of applications and malware variants.
- **Metadata:** Each APK file in the AndroZoo dataset is accompanied by metadata, including information such as package name, version code, version name, developer name, permissions requested, and other details extracted from the AndroidManifest.xml file.
- **SHA-256 Hashes:** Each APK file is uniquely identified by its SHA-256 hash value, providing a reliable means of referencing and retrieving specific applications from the dataset.

- **Source Information:** AndroZoo records the source from which each APK file was obtained, such as the Google Play Store, third-party markets, research projects, or internet repositories. This information helps researchers understand the provenance of the applications and assess their trustworthiness.
- **Malware Labels:** Some versions of AndroZoo include labels indicating whether an APK file is classified as malware or benign. These labels are often derived from antivirus scans, sandboxes, or manual analysis conducted by researchers.
- **File Analysis:** AndroZoo allows researchers to analyze the contents of APK files, including the resources, assets, native libraries, and other components that make up the application package.
- **API Access:** AndroZoo provides an API that allows researchers to programmatically query and retrieve APK files based on various criteria, such as package name, SHA-256 hash, permissions requested, or source information.
- **Temporal Data:** Some versions of AndroZoo include temporal data, indicating when each APK file was first observed or added to the dataset. This temporal information enables researchers to analyze trends, evolution, and changes in the Android application landscape over time[17].

Drebin Dataset Description: The Drebin dataset provides a rich set of features extracted from Android applications, enabling detailed analysis and machine learning model training.

[18] **Malware Classification:** The malware samples in the CICMalDroid 2017 dataset are classified into four categories, with samples coming from 42 unique malware families.

Feature Extraction: For feature extraction and selection, network traffic features in the form of .pcap files were captured, and over 80 features were extracted using CICFlowMeter-V3 during different states, including installation, before restart, and after restart.

Data Capturing States: The dataset defined specific scenarios for each

malware category and three states of data capturing to overcome the stealthiness of advanced malware, ensuring a comprehensive view of the malware samples.

Public Availability: The CICMalDroid 2017 dataset is publicly available.

Classification Categories: The dataset classifies the Android apps into five distinct categories:

- Adware.
- Scareware,
- SMS Malware
- Riskware
- Benign: Non-malicious apps.

[19].

3.5.3 Preparation of Dataset

In Android datasets, the files are in numerical or binary format, the model SecureBERT takes input as textual data.

3.5.4 Ransomware Dataset

The datasets consists of ransomware features.[20]

3.5.5 Trojan Detection

Trojan horse can penetrate the system or network when they are invited by the users unknowingly through the visiting of malicious and unknown websites. The datasets consists of benign and malicious network packets to detect Trojan[21].

3.5.6 TUANDROMD

Tuandromd dataset consists of 4465 instances and 241 attributes. Here, the attributes are classified as malware or goodware.[22]

3.5.7 Challenges and Considerations of Synthetic Dataset:

- **Realism**: The synthetic data must closely mimic real-world data to be useful. This includes not only the appearance and structure of the data but also the behavioral patterns of malware.

- **Bias and Overfitting**: If the synthetic data does not accurately represent the variations and complexities of real-world malware, models trained on this data may perform poorly in real-world scenarios. Care must be taken to avoid biases in the synthetic data.

- **Validation**: It's essential to validate models trained on synthetic data with real-world data to ensure they perform well in actual use cases. Synthetic data should complement rather than replace real-world data entirely.

- **Ethical Considerations**: Generating synthetic data should be done ethically, ensuring that it does not inadvertently create security risks or misuse scenarios.
  [23]

3.5.8 APT Notes

It is a collection of documents and notes related to Advanced Persistent Threat. [24]

3.5.9 Final Dataset

The final dataset is obtained after combining all the datasets mentioned above and using Faker and random package for synthetic data.

The figures of dataset are displayed in last page of methodology section.

3.6 Preprocessing and Post Processing Algorithm for Model

Algorithm 1 Preprocessing Algorithm

Input: dataset_file dataset ← LoadCSV(dataset_file) **Step 1: Assign Column Names** dataset_with_col_names ← AssignColumnNames(dataset) **Step 2: Clean Labels** cleaned_dataset ← CleanLabels(dataset_with_col_names) **Step 3: Label Encoding** encoded_dataset ← EncodeLabels(cleaned_dataset) **Step 4: Handle Missing and Invalid Labels** filtered_dataset ← RemoveInvalidLabels(encoded_dataset) **Step 5: Dataset Splitting** train_set, validation_set, test_set ← SplitDataset(filtered_dataset) **Step 6: Tokenize Texts** tokenized_train_set ← TokenizeTexts(train_set) tokenized_validation_set ← TokenizeTexts(validation_set) tokenized_test_set ← TokenizeTexts(test_set) **Step 7: Create Datasets** train_dataset ← CreatePyTorchDataset(tokenized_train_set) validation_dataset ← CreatePyTorchDataset(tokenized_validation_set) test_dataset ← CreatePyTorchDataset(tokenized_test_set)

Preprocessing Steps

1. **Load Dataset:** Load the initial dataset containing cybersecurity-related texts.
2. **Data Cleaning:** Clean the dataset by removing any irrelevant or noisy data.
3. **Data Normalization:** Normalize the data to ensure consistency in formatting (e.g., lowercasing, removing special characters.
4. **Label Encoding:** Encode categorical labels into numerical format for model training.
5. **Handle Missing and Negative Labels:** Rows with unmapped labels or negative labels (if any) were dropped.
6. **Dataset Splitting:** Split the cleaned and encoded data into training, validation, and test datasets.
7. **Tokenize Texts:** Tokenize the text data into a format suitable for input into the SecureBERT model.

8. **Create PyTorch Datasets:** Convert the tokenized data into PyTorch datasets for model training and evaluation.

Algorithm 2 Postprocessing Algorithm

Step 1: Evaluate Model
Input: test_set: The dataset used for evaluation
Output: predictions: Model predictions on the test set
predictions ← EvaluateModel(test_set)

Step 2: Compute Metrics
Input: predictions, test_set: The model predictions and true labels
Output: accuracy, precision, recall, f1_score: Evaluation metrics
accuracy, precision, recall, f1_score ← ComputeMetrics(predictions, test_set)

Step 3: Generate Confusion Matrix
Input: predictions, test_set: The model predictions and true labels
Output: confusion_matrix: A matrix visualizing model performance
confusion_matrix ← GenerateConfusionMatrix(predictions, test_set)

Step 4: Classification Report
Input: precision, recall, f1_score: Metrics computed in Step 2
Output: classification_report: Detailed report with precision, recall, and f1 score
classification_report ← GenerateClassificationReport(precision, recall, f1_score)

Post-processing Steps

1. **Model Prediction:** Generate predictions for the test dataset using the trained SecureBERT model.
2. **Compute Metrics:** Calculate relevant performance metrics (e.g., accuracy, precision, recall) based on the model's predictions and the ground truth labels.
3. **Confusion Matrix:** Create a confusion matrix to visualize the performance of the model across different classes.
4. **ROC-AUC Curve:** Generate the Receiver Operating Characteristic - Area Under Curve (ROC-AUC) to evaluate the model's ability to discriminate between classes..
5. **Interactive Classification:** An interface is provided for interactive classification which allows users to input new data and receive predictions from the model.

3.7 Elaboration of Working Principle

To illuminate the working principle, let's break down the process into three key stages: preprocessing, model training, and post-processing. **Working Principle: Manipulation of ML Ready data through Model Stages** After the data is preprocessed, it is fed into model for training,validation, and testing. We shall illustrate how the ML ready data is manipulated as it passes through different stages of the chosen model:

1. **Model Initialization**: The model is initialized with specific parameters and configurations from the Hugging Face transformers library and is used for text classification.
2. **Training the Model**: The training dataset is used to train the model. In this phase, the model learns to map input texts to their corresponding labels by minimizing the loss function.
3. **Validation**: The validation dataset is used to evaluate the model during training which helps in monitoring the model's performance and adjusting hyperparameters if required.
4. **Testing the model**: After training, the model is tested on the test dataset to evaluate its performance on unseen data. The model predictions are compared with the actual labels to compute various evaluation metrics.
5. **Postprocessing of Predictions**: The raw output predictions from the model are post-processed to generate meaningful results which includes converting the predicted labels from numerical to their corresponding textual form.

During the fine-tuning process, the model's internal representations are manipulated and adapted to the specific characteristics of the malware data, enabling it to learn the patterns and features relevant to malware analysis. Let's discuss about finetuning the model in detail

3.7.1 Finetuning SecureBERT with LoRA

LoRA (Low-Rank Adaptation)

LoRA is a parameter-efficient finetuning method that injects a small number

of trainable adapter parameters into the model, while keeping the original model weights frozen. It allows for faster and more memory-efficient finetuning compared to full model finetuning. The training process of the SecureBERT model for malware classification starts with inputting preprocessed text data and labels into the model, which then tokenizes the text into numerical representations. The model predicts malware classes and calculates the loss by comparing predictions to actual labels using a cross-entropy loss function. This loss indicates prediction accuracy and guides the model's learning. The optimization phase employs LoRA (Low-Rank Adaptation) to update the model's weights efficiently, using backpropagation to compute gradients for necessary adjustments. This iterative process, repeated across multiple epochs, fine-tunes the model to enhance its accuracy in classifying different types of malware, ensuring effective performance in real-world scenarios.

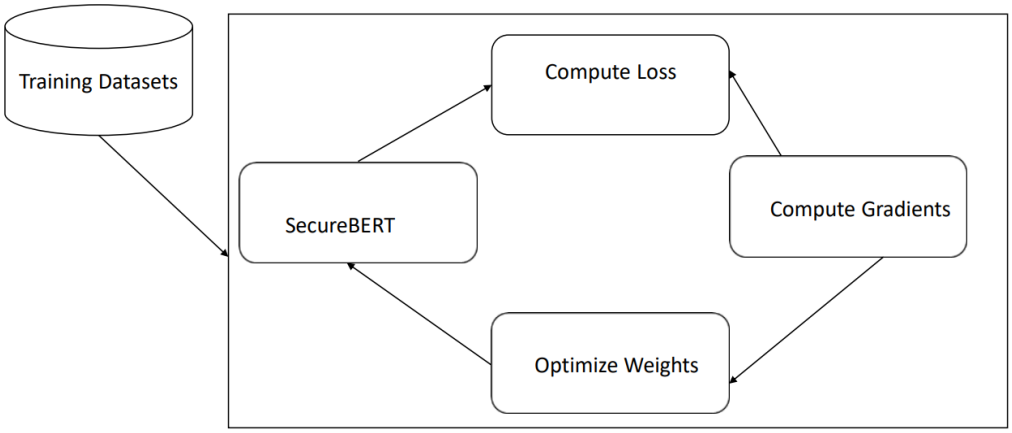


Figure 3.28: Finetuning the model

Hyperparameter Tuning

- **Description**: Hyperparameters such as learning rate, batch size, epoch and others can be adjusted to optimize the model's performance on the target task without changing the model architecture.
- **How It Works**: Hyperparameters are tuned based on validation performance to achieve better results with minimal changes to the model.

Post-Processing of model output:

After the model is trained and fine-tuned, output needs to be post-processed to make the model interpretable and useful. This involves several steps:

3.7.2 Sample Calculations

Logits and Softmax Suppose the model outputs the following logits for a two-class classification problem:

$$\text{Logits for class } 1 = 2.0 \tag{3.13}$$

$$\text{Logits for class } 2 = -1.0 \tag{3.14}$$

To calculate the softmax probabilities, we use the following formula:

$$P_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3.15}$$

where P_i is the probability of class i, and z_i is the logit for class i.

3.7.3 Step-by-Step Calculation

1. **Calculate the exponentials of the logits:**

$$e^{z_1} = e^{2.0} \approx 7.389 \tag{3.16}$$

$$e^{z_2} = e^{-1.0} \approx 0.3679 \tag{3.17}$$

2. **Calculate the sum of exponentials:**

$$\text{Sum} = e^{2.0} + e^{-1.0} \approx 7.389 + 0.3679 = 7.7569 \tag{3.18}$$

3. **Calculate the softmax probabilities:**

$$P_1 = \frac{e^{2.0}}{\text{Sum}} = \frac{7.389}{7.7569} \approx 0.951 \tag{3.19}$$

$$P_2 = \frac{e^{-1.0}}{\text{Sum}} = \frac{0.3679}{7.7569} \approx 0.049 \tag{3.20}$$

So the softmax probabilities are approximately:

$$\text{Probability of class } 1 \approx 0.951 \tag{3.21}$$

$$\text{Probability of class } 2 \approx 0.049 \tag{3.22}$$

3.7.4 Example Calculation for Logits

For a hypothetical simple model, suppose the logits are computed using a linear transformation:

$$\text{Logits} = W \cdot X + b \tag{3.23}$$

where W is the weight matrix, X is the input features, and b is the bias. Assume:

$$W = [0.5, -0.2] \tag{3.24}$$

$$b = [1.0, 0.5] \tag{3.25}$$

$$X = [2.0, 3.0] \tag{3.26}$$

Compute logits:

$$\text{Logits} = [0.5 \cdot 2.0 + (-0.2) \cdot 3.0 + 1.0, 0.5 \cdot 2.0 + (-0.2) \cdot 3.0 + 0.5] \tag{3.27}$$

$$\text{Logits} = [1.0 - 0.6 + 1.0, 1.0 - 0.6 + 0.5] \tag{3.28}$$

$$\text{Logits} = [1.4, 0.9] \tag{3.29}$$

Accuracy

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Precision

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

Recall

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

F1 Score

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

3.8 Verification and Validation Procedures

Relevance of Chosen Metrics:

The chosen confusion metrics are crucial for evaluating the performance of the project.They are important for measuring the effectiveness and reliability of the malware analysis system. **Basic Definitions and Formulas of Confusion Metrics:**

The chosen confusion metrics are crucial for evaluating the performance of the project. These metrics provide insights into the model's ability to correctly classify malware samples, assess its accuracy, and identify any misclassifications.

1. **True Positive (TP)**:
   - **Definition**: The number of correctly predicted positive instances (malware samples).
   - **Formula**: $TP =$ Number of malware samples correctly classified as malware.
2. **True Negative (TN)**:
   - **Definition**: The number of correctly predicted negative instances (non-malware samples).
   - **Formula**: $TN =$ Number of non-malware samples correctly classified as non-malware.
3. **False Positive (FP)**:
   - **Definition**: The number of non-malware samples incorrectly classified as malware.
   - **Formula**: $FP =$ Number of non-malware samples incorrectly classified as malware.
4. **False Negative (FN)**:
   - **Definition**: The number of malware samples incorrectly classified as non-malware.
   - **Formula**: $FN =$ Number of malware samples incorrectly classified as non-malware.
5. **Accuracy**:

- **Definition**: The proportion of correctly classified instances over the total number of instances.
- **Formula**:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.30}$$

6. **Precision**:
   - **Definition**: The proportion of correctly predicted positive instances among all instances predicted as positive.
   - **Formula**:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.31}$$

7. **Recall (Sensitivity)**:
   - **Definition**: The proportion of correctly predicted positive instances among all actual positive instances.
   - **Formula**:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.32}$$

8. **F1 Score**:
   - **Definition**: The harmonic mean of precision and recall, providing a balance between the two metrics.[25]
   - **Formula**:

$$\text{F1 Score} = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)} \tag{3.33}$$

ROC Curve: A Receiver Operating Characteristic (ROC) curve is a graphical representation that demonstrates the performance of a binary classifier as its threshold for discrimination changes. This curve is generated by plotting the true positive rate (TPR) against the false positive rate (FPR) at different threshold levels. The ROC curve highlights the balance between sensitivity

(TPR) and specificity (1 - FPR). Classifiers that produce curves nearing the top-left corner exhibit superior performance. A random classifier typically results in points along the diagonal where FPR equals TPR. The closer the ROC curve is to this 45-degree diagonal, the less accurate the classifier is.[26][27]

Equations for ROC-AUC Calculation: 1. **True Positive Rate (TPR)**:

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.34}$$

TPR is also known as sensitivity or recall. 2. **False Positive Rate (FPR)**:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{3.35}$$

FPR represents the proportion of negatives that were incorrectly classified as positives. 3. **Area Under the Curve (AUC)**: The ROC-AUC score is the area under the ROC curve and is calculated as:

$$\text{AUC} = \int_0^1 \text{TPR}\, d(\text{FPR}) \tag{3.36}$$

Alternatively, AUC can be approximated using numerical methods such as the trapezoidal rule:

$$\text{AUC} \approx \sum_{i=1}^{n} (\text{FPR}_i - \text{FPR}_{i-1}) \cdot \frac{\text{TPR}_i + \text{TPR}_{i-1}}{2} \tag{3.37}$$

where n is the number of points in the ROC curve. A higher AUC value indicates better classifier performance, with 1.0 representing a perfect classifier and 0.5 indicating no discriminative power.

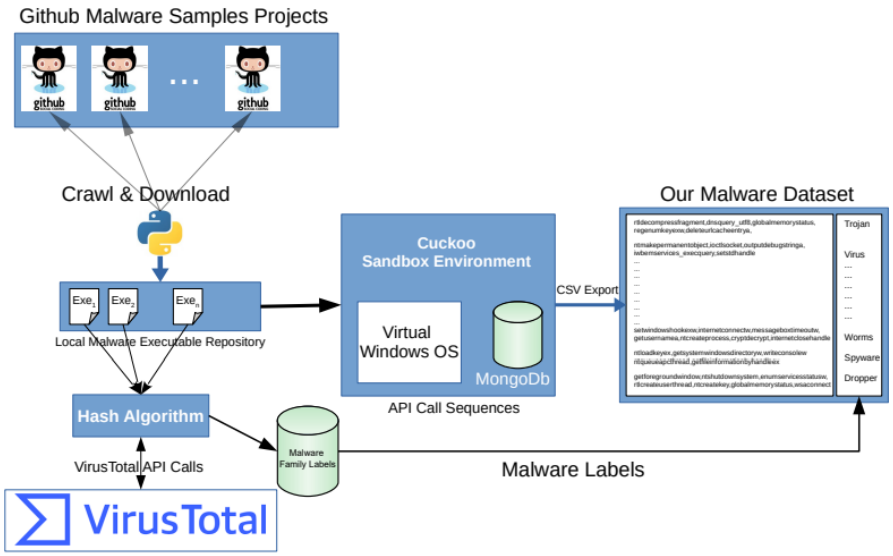


Figure 3.10: Malware DB Dataset Preparation
Source, adapted from [16]

Operation Poisoned Handover: Unveiling Ties Between APT Activity
in Hong Kong's Pro-Democracy Movement
As the pro-democracy movement in Hong Kong has continued, we've been watching for indications of confrontation taking place in cyberspace. Protests began in September and have continued to escalate.

In recent weeks, attackers have launched a series of Distributed Denial of Service attacks (DDoS) against websites promoting democracy in Hong Kong. According to the Wall Street Journal, websites belonging to Next Media's Apple Daily publication have suffered from an ongoing DDoS attack that "brought down its email system for hours". According to other reports, Next Media's network has suffered a "total failure" as a result of these attacks. Additionally, at least one member of the popular online forum HKGolden was arrested for posting messages encouraging support for the OccupyCentral Pro Democracy movement.

The use of DDoS attacks as a political tool during times of conflict is not new; patriotic hacktivist groups frequently use them as a means to stifle political activity of which they disapprove. The question of state sponsorship (or at least tacit approval) in online crackdowns is often up for debate and ambiguous from a technical evidence and tradecraft perspective.

In this case, however, we've discovered an overlap in the tools and infrastructure used by China-based advanced persistent threat (APT) actors and the DDoS attack activity. We believe that these DDoS attacks are linked to previously observed APT activity, including Operation Poisoned Hurricane. This correlation sheds light on the potential relationships, symbiosis and tool sharing between patriotic hacker activities designed to disrupt anti-government activists in China, and the APT activity we consistently see that is more IP theft and espionage-focused.

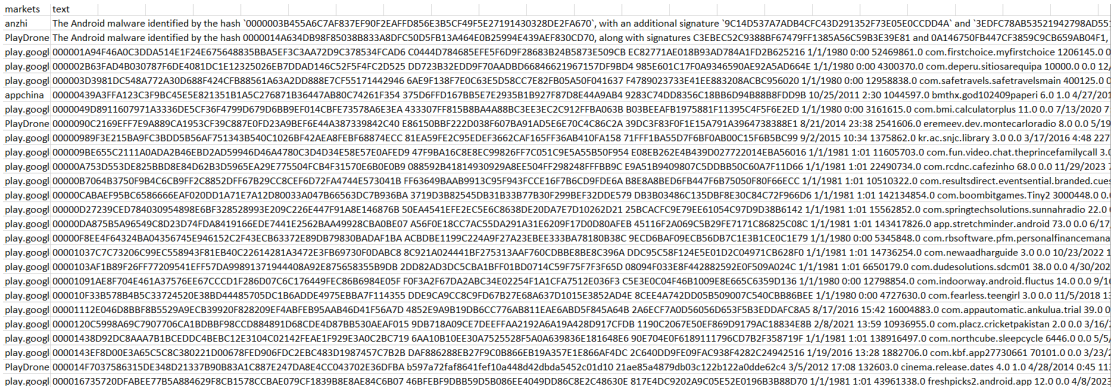
Ongoing DDoS Attacks Target the Pro-Democracy Movement

FireEye has identified a number of binaries coded to receive instructions from a set of command and control (C2) servers instructing participating bots to attack Next Media-owned websites and the HKGolden forum. Next Media is a large media company in Hong Kong and the HkGolden forum has been used as a platform to organize pro-democracy protests. Each sample we identified is signed with digital certificates that have also been used by APT actors to sign binaries in previous intrusion operations:

These binaries are W32 Cabinet self-extracting files that drop a variant of an older DDoS tool known as KernelBot . All of the samples we identified have the "NewVersion" value of 20140926. Structurally, all of these samples are similar in that they drop three files:

- ctfmon.exe-a legitimate, signed copy of the Pidgin IM client (md5 hash = 1685f978149d7ba8e039af9a4d5803c7)
- libssp-0.dll–malware DLL which is side-loaded by ctfmon.exe to decode and launch KernelBot. Most versions of this dll are also

Figure 3.11: Textual Description of one of Malware DB dataset



markets	text
anzhi	The Android malware identified by the hash `0000003B455A6C7AF837EF90F2EAFFD856E3B5CF49F5E27191430328DE2FA670`, with an additional signature `9C14D537A7ADB4CFC43D291352F73E05E0CCDD4A` and `3EDFC78AB53521942798AD55`
PlayDrone	The Android malware identified by the hash 00000014A634DB98F85038B833A8DFC50D5FB13A464E0B25994E439AEF830CD70, along with signatures C3EBEC52C9388BF67479FF138SA56C59B3E39E81 and 0A146750FB447CF3859C9CB659AB04F1,
play.googl	000001A94F46A0C3DDA514E1F24E675648835BBA5EF3C3AA72D9C378534FCAD6 C0444D784685EFE5F6D9F28683B24B5873E509CB EC82771AE018B93AD784A1FD2B625216 1/1/1980 0:00 52469861.0 com.firstchoice.myfirstchoice 1206145.0 0
play.googl	000002B63FAD4B030787F6DE4081DC1E12325026EB7DDAD146C52F5F4FC2D525 DD723B32EDD9F70AADBD66846621967157DF9BD4 985E601C17F0A9346590AE92A5AD664E 1/1/1980 0:00 4300370.0 com.deperu.sitiosarequipa 10000.0 0.0 12/
play.googl	000003D3981DC548A772A30D688F424CFB88561A63A2DD888E7CF55171442946 6AE9F138F7E0C63E5D58CC7E82FB05A50F041637 F4789023733E41EE883208ACBC956020 1/1/1980 0:00 12958838.0 com.safetravels.safetravelsmain 400125.0 0
appchina	00000439A3FFA123C3F9BC45E5E821351B1A5C276871B36447AB80C74261F354 375D6FFD167BB5E7E2935B1B927F87D8E44A9AB4 9283C74DD8356C18BB6D94B888B8FDD9B 10/25/2011 2:30 1044597.0 bmthx.god102409paperi 6.0 1.0 4/27/201
play.googl	0000049D891160797 1A3336DE5CF36F4799D679D6B89EF014CBFE73578A6E3EA 433307FF815BBBA4A88BC3EE3EC2C912FFBAD63B B03BEEAFB1975881F11395C4F5F6E2ED 1/1/1980 0:00 3161615.0 com.bmi.calculatorplus 11.0 0.0 7/13/2020 7
PlayDrone	00000090C2169EFF7E9A889CA1953CF39C887E0FD23A9BEF6E44A387339842C40 E861508BF222D038F607BA91AD5E6E70C4C86C2A 39DC3F83F0F1E15A791A3964738388E1 8/21/2014 23:38 2541606.0 eremeev.dev.montecarloradio 8.0 0.0 5/19
play.googl	00000989F3E215BA9FC3BDD5B56AF751343B540C1026BF42AEA8FEBF68874ECC 81EA59FE2C95EDEF3662CAF165FF36AB410FA158 71FFF1BA55D7F6BF0AB00C15F6B5BC99 9/2/2015 10:34 1375862.0 kr.ac.snjc.library 3.0 0.0 3/17/2016 4:48 227
play.googl	000009BE655C2111A0ADA2B46EBD2AD55946D46A4780C3D4D34E58E57E0AFED9 47F9BA16C8E8EC99826FF7C051C9E5A55B50F954 E08EB262E4B439D027722014EBA56016 1/1/1981 1:01 11605703.0 com.fun.video.chat.theprincefamilycall 3.0
play.googl	00000A753D553DE825BBD8E84D62B3D5965EA29E775504FCB4F31570E6B0E0B9 088592B41814930929A8EE504FF29824BFFFBB9C E9A51B9409807C5DDBB50C60A7F11D66 1/1/1981 1:01 22490734.0 com.rcdnc.cafezinho 68.0 0.0 11/29/2023 1
play.googl	00000B7064B3750F9B4C6CB9FF2C8852DFF67B29CC8CEF6D72FA4744E573041B FF63649BAAB9913C95F943FCCE16F7B6CD9FDE6A B8E8A8BED6FB447F6B75050F80F66ECC 1/1/1981 1:01 10510322.0 com.resultsdirect.eventsential.branded.cues
play.googl	00000CABAEF95BC6586666EAF020DD1A71E7A12D80033A047B66563DC7B936BA 3719D3B82545DB31B33B77B30F299BEF32DDE579 DB3B03486C135DBF8E30C84C72F966D6 1/1/1981 1:01 142134854.0 com.boombitgames.Tiny2 3000448.0 0.0
play.googl	00000D27239CED784030954898E6BF32852899E209C226E447F91A8E146876B 50EA4541EFE2EC5E6C8638DE20DA7E7D10262D21 25BCACFC9E79EE61054C97D9D38B6142 1/1/1981 1:01 13562852.0 com.springtechsolutions.sunnahradio 22.0
play.googl	00000DA875B5A96549C8D23D74FDA8419166EDE7441E2562BAA49928CBA0BE07 A56F0E18CC7AC55DA291A31E6209F17D0D80AFEB 45116F2A069C5B29FE7171C86825C08C 1/1/1981 1:01 143417826.0 app.stretchminder.android 73.0 0.0 6/17
play.googl	00000F8EE4F64324BA04356745E946152C2F43ECB63372E89DB79830BADAF1BA ACBDBE1199C224A9F27A23EBEE333BA78180B38C 9ECD6BAF09ECB56DB7C1E3B1CE0C1E79 1/1/1980 0:00 5345848.0 com.rbsoftware.pfm.personalfinancemana
play.googl	00001037C7C73206C99EC558943F81EB40C22614281A3472E3FB69730F0DABC8 8C921A024441BF275313AAF760CDBBE8BE8C396A DDC95C58F124E5E01D2C04971CB628F0 1/1/1981 1:01 14736254.0 com.newaadharguide 3.0 0.0 10/23/2022 1
play.googl	0000103AF1B89F26FF77209541EFF57DA99891371944408A92E875658355B9DB 2DD82AD3DC5CBA1BFF01BD0714C59F75F7F3F65D 08094F033E8F442882592E0F509A024C 1/1/1981 1:01 6650179.0 com.dudesolutions.sdcm01 38.0 0.0 4/30/202
play.googl	00001091AE8F704E461A37576EE67CCCD1F286D07C6C176449FEC86B6984E05F F0F3A2F67DA2ABC34E02254F1A1CFA7512E036F3 C5E3E0C04F46B1009E8E665C6359D136 1/1/1980 0:00 12798854.0 com.indoorway.android.fluctus 14.0 0.0 9/1
play.googl	000010F33B578B4B5C33724520E38BD44485705DC1B6ADDE4975EBBA7F114355 DDE9CA9CC8C9FD67B27E68A637D1015E3852AD4E 8CEE4A742DD05B509007C540CBB86BEE 1/1/1980 0:00 4727630.0 com.fearless.teengirl 3.0 0.0 11/5/2018 1
play.googl	00001112E046D8BBF8B5529A9ECB39920F828209EF4ABFEB95AAB46D41F56A7D 4852E9A9B19DB6CC776AB811EAE6ABD5F845A64B 2A6ECF7A0D56056D653F5B3EDDAFC8A5 8/17/2016 15:42 16004883.0 com.appautomatic.ankulua.trial 39.0 0
play.googl	0000120C5998A69C7907706CA1BDBBF98CCD884891D68CDE4D87BB530AEAF015 9DB718A09CE7DEEFFAA2192A6A19A428D917CFDB 1190C2067E50EF869D9179AC18834E8B 2/8/2021 13:59 10936955.0 com.placz.cricketpakistan 2.0 0.0 3/16/
play.googl	00001438D92DC8AAA7B1BCEDDC4BEBC12E3104C02142FEAE1F929E3A0C2BC719 6AA10B10EE30A7525528F5A0A639836E181648E6 90E704E0F6189111796CD7B2F358719F 1/1/1981 1:01 138916497.0 com.northcube.sleepcycle 6446.0 0.0 5/5/
play.googl	0000143EF8D00E3A65C5C8C380221D00678FED906FDC2EBC483D1987457C7B2B DAF886288EB27F9C0B866EB19A357E1E866AF4DC 2C640DD9FE09FAC938F4282C24942516 1/19/2016 13:28 1882706.0 com.kbf.app27730661 70101.0 0.0 3/23/
PlayDrone	00001 4F7037586315DE348D21337B90B83A1C887E247DA8E4CC043702E36DFBA b597a72faf8641fef10a448d42dbda5452c01d10 21ae85a4879db03c122b122a0dde62c4 3/5/2012 17:08 132603.0 cinema.release.dates 4.0 1.0 4/28/2014 0:45 11
play.googl	0000167352 0DFABEE77B5A884629F8CB1578CCBAE079CF1839B8E8AE84C6B07 46BFEBF9DBB59D5B086EE4049DD86C8E2C48630E 817E4DC9202A9C05E52E0196B3B88D70 1/1/1981 1:01 43961338.0 freshpicks2.android.app 12.0 0.0 4/8/2023

Figure 3.12: Androzoo Dataset Features

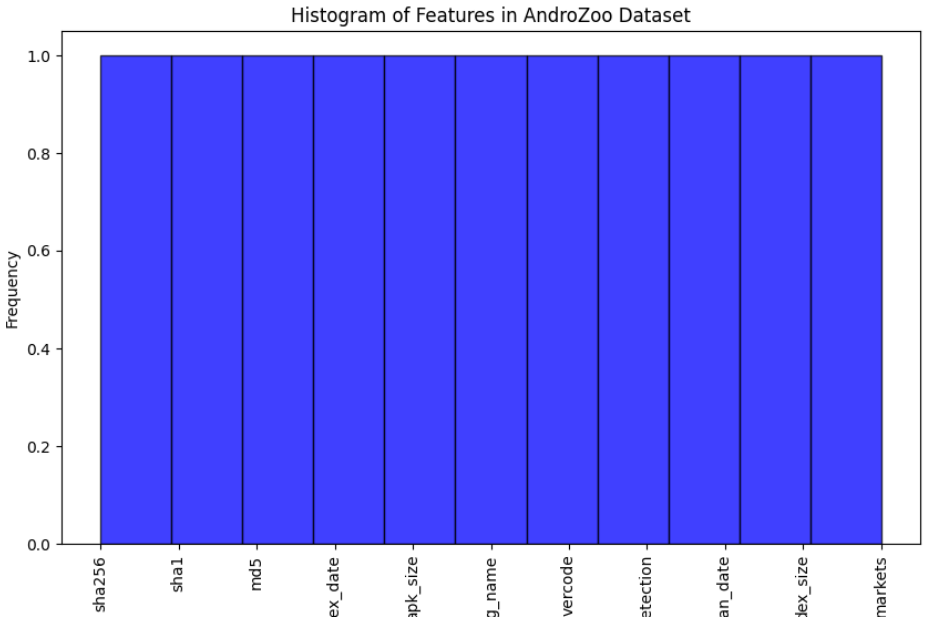


Figure 3.13: Histogram of Androzoo Dataset Features

text	class
	S
In our Android application, we utilize the `SmsManager` class to send SMS messages with	S
In our Android application, we send SMS messages using the `SmsManager` class with the	S
Ljava.lang.Class.cast Ljava.net.URLDecoder READ_PHONE_STATE Landroid.content.Cont	S
Ljava.net.URLDecoder READ_PHONE_STATE ClassLoader Ljava.lang.Class.getDeclaredFie	S
READ_PHONE_STATE READ_SMS Ljavax.crypto.spec.SecretKeySpec android.intent.actior	S
onServiceConnected bindService ServiceConnection android.os.Binder READ_PHONE_ST/	S
Ljava.lang.Class.cast Ljava.net.URLDecoder READ_PHONE_STATE Landroid.content.Cont	S
SEND_SMS android.telephony.SmsManager READ_PHONE_STATE RECEIVE_SMS READ_S	S
SEND_SMS READ_PHONE_STATE READ_SMS android.intent.action.BOOT_COMPLETED a	S
transact attachInterface android.os.Binder SEND_SMS Ljava.lang.Class.getCanonicalNam	S
SEND_SMS android.telephony.SmsManager READ_PHONE_STATE ClassLoader Ljava.lang	S
READ_PHONE_STATE android.intent.action.BOOT_COMPLETED KeySpec HttpGet.init Sec	S
transact onServiceConnected bindService attachInterface ServiceConnection android.os.	S
READ_PHONE_STATE ClassLoader Ljava.lang.Class.getDeclaredField TelephonyManager.(	S
SEND_SMS READ_PHONE_STATE TelephonyManager.getLine1Number android.telephony	S
SEND_SMS android.telephony.SmsManager READ_PHONE_STATE Landroid.content.Cont	S
SEND_SMS Ljava.lang.Class.getCanonicalName android.telephony.SmsManager READ_PI	S
SEND_SMS READ_PHONE_STATE Ljava.lang.Class.getField RECEIVE_SMS READ_SMS andr	S
READ_PHONE_STATE RECEIVE_SMS READ_SMS Ljavax.crypto.spec.SecretKeySpec androi	S
transact onServiceConnected bindService attachInterface ServiceConnection android.os.	S
SEND_SMS android.telephony.SmsManager READ_SMS INTERNET TelephonyManager.ge	S
SEND_SMS android.telephony.SmsManager READ_PHONE_STATE RECEIVE_SMS Telepho	S
SEND_SMS android.content.pm.Signature READ_PHONE_STATE ClassLoader RECEIVE_SM	S
transact onServiceConnected bindService attachInterface ServiceConnection android.os.	S
SEND_SMS android.telephony.SmsManager READ_PHONE_STATE Landroid.content.Cont	S

Figure 3.14: Drebin Dataset Features

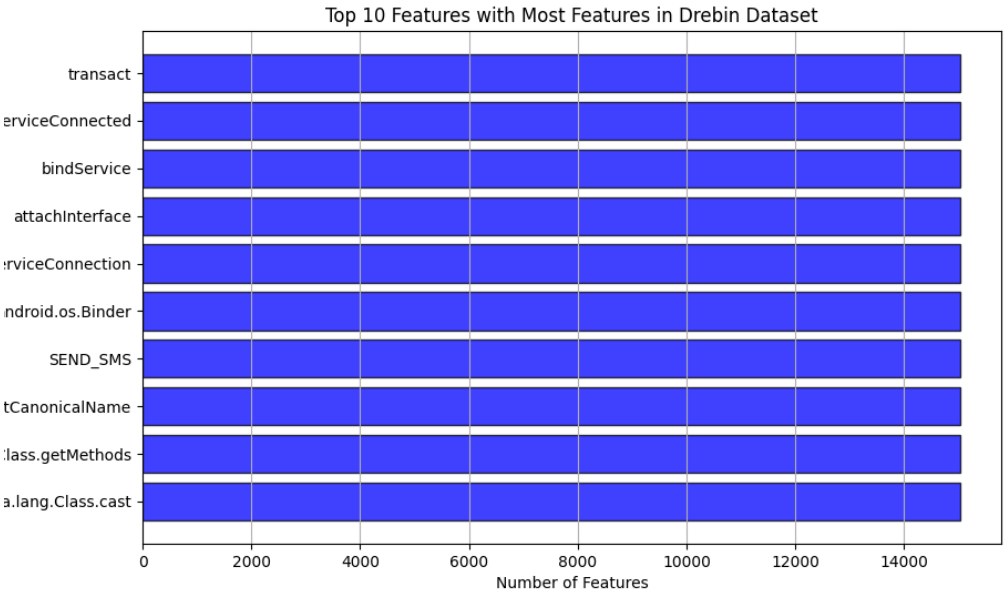


Figure 3.15: Histogram of Drebin Dataset Features

Flow ID	Source IP	Source Po	Destinatic	Destinatic	Protocol	Timestamp	Flow Dura	Total Fwd	Total Bacl	Total Leng	Total Leng	Fwd Packe	Fwd Packe	Fwd Packe	Fwd Packe	Bwd Packe	Bwd Packe	Bwd Packe	Bwd Packe	Flow Bytes	Flow P
172.217.2.	10.42.0.15	36635	172.217.2.	443	6	14/06/2017 01:54:51	22287	1	1	0	0	0	0	0	0	0	0	0	0	0	89.738
172.217.0.	10.42.0.15	57267	172.217.0.	443	6	14/06/2017 01:54:51	23295	1	1	0	0	0	0	0	0	0	0	0	0	0	85.855
172.217.0.	10.42.0.15	48347	172.217.0.	443	6	14/06/2017 01:54:51	23323	1	1	0	0	0	0	0	0	0	0	0	0	0	85.753
172.217.0.	10.42.0.15	56457	172.217.0.	443	6	14/06/2017 01:54:51	22184	1	1	0	0	0	0	0	0	0	0	0	0	0	90.155
172.217.2.	10.42.0.15	48575	172.217.2.	443	6	14/06/2017 01:54:51	22319	1	1	0	0	0	0	0	0	0	0	0	0	0	89.609
172.217.0.	10.42.0.15	45288	172.217.0.	443	6	14/06/2017 01:54:51	22347	1	1	0	0	0	0	0	0	0	0	0	0	0	89.497
172.217.0.	10.42.0.15	58629	172.217.0.	443	6	14/06/2017 01:54:51	22351	1	1	0	0	0	0	0	0	0	0	0	0	0	89.481
172.217.0.	10.42.0.15	35535	172.217.0.	443	6	14/06/2017 01:54:51	23394	1	1	0	0	0	0	0	0	0	0	0	0	0	85.492
172.217.0.	10.42.0.15	59218	172.217.0.	443	6	14/06/2017 01:54:51	23659	1	1	0	0	0	0	0	0	0	0	0	0	0	84.534
172.217.0.	10.42.0.15	47211	172.217.0.	443	6	14/06/2017 01:54:57	100	2	0	31	0	31	0	15.5	21.92031	0	0	0	0	310000	200
172.217.0.	172.217.0.	443	10.42.0.15	47211	6	14/06/2017 01:54:57	37	2	0	0	0	0	0	0	0	0	0	0	0	0	54054.
172.217.0.	10.42.0.15	54440	172.217.0.	443	6	14/06/2017 01:55:00	11	2	0	23	0	23	0	11.5	16.26346	0	0	0	0	2090909	18181
172.217.0.	172.217.0.	443	10.42.0.15	54440	6	14/06/2017 01:55:00	479	2	0	0	0	0	0	0	0	0	0	0	0	0	4175.3

Figure 3.16: CicAndMal2017 Dataset Features

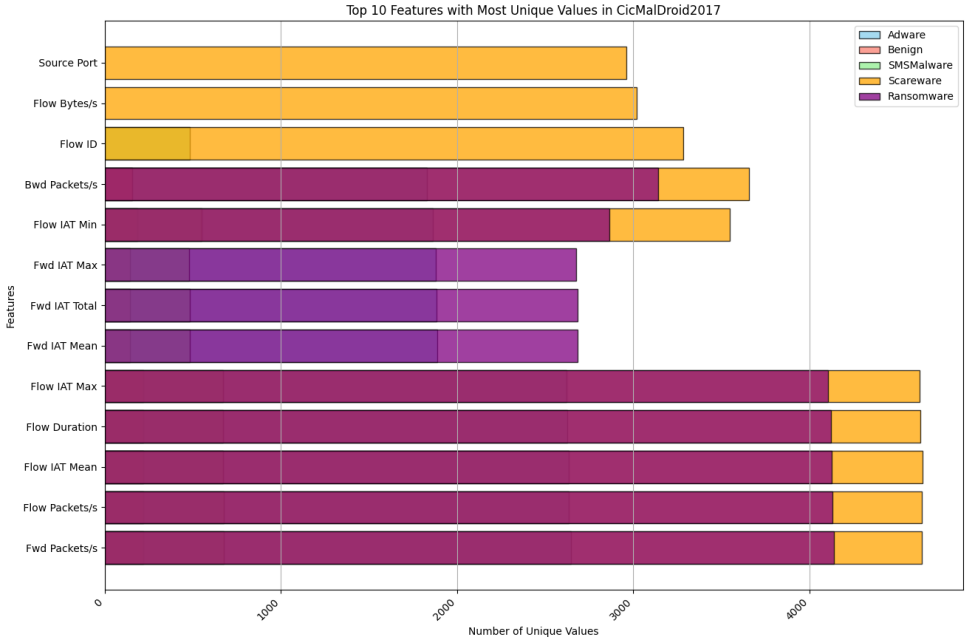


Figure 3.17: Histogram of CicAndMal2017 Dataset Features

,text
0,Source Port:443 Destination Port:40952 Protocol:6 Flow Duration:242471 Total Fwd Packets:2 Total Backward Packets:0 Total Length of Fwd Packets:0 Total Length of Bwd Packets:0 Fwd Packet Length Max:0 Fwd Packet Length Min:0 Fwd Packet Length Mean:0.0 Fwd Packet Length Std:0.0 Bwd Packet Length Max:0 Bwd Packet Length Min:0 Bwd Packet Length Mean:0.0 Bwd Packet Length Std:0.0 Flow Bytes/s:0.0 Flow Packets/s:8.248409088 Flow IAT Mean:242471.0 Flow IAT Std:0.0 Flow IAT Max:242471.0 Flow IAT Min:242471.0 Fwd IAT Total:242471.0 Fwd IAT Mean:242471.0 Fwd IAT Std:0.0 Fwd IAT Max:242471.0 Fwd IAT Min:242471.0 Bwd IAT Total:0.0 Bwd IAT Mean:0.0 Bwd IAT Std:0.0 Bwd IAT Max:0.0 Bwd IAT Min:0.0 Fwd PSH Flags:0 Bwd PSH Flags:0 Fwd URG Flags:0 Bwd URG Flags:0 Fwd Header Length:64 Bwd Header Length:0 Fwd Packets/s:8.248409088 Bwd Packets/s:0.0 Min Packet Length:0 Max Packet Length:0 Packet Length Mean:0.0 Packet Length Std:0.0 Packet Length Variance:0.0 FIN Flag Count:0 SYN Flag Count:0 RST Flag Count:0 PSH Flag Count:0 ACK Flag Count:1 URG Flag Count:1 CWE Flag Count:0 ECE Flag Count:0 Down/Up Ratio:0 Average Packet Size:0.0 Avg Fwd Segment Size:0.0 Avg Bwd Segment Size:0.0 Fwd Avg Bytes/Bulk:0 Fwd Avg Packets/Bulk:0 Fwd Avg Bulk Rate:0 Bwd Avg Bytes/Bulk:0 Bwd Avg Packets/Bulk:0 Bwd Avg Bulk Rate:0 Subflow Fwd Packets:2 Subflow Fwd Bytes:0 Subflow Bwd Packets:0 Subflow Bwd Bytes:0 Init_Win_bytes_forward:179 Init_Win_bytes_backward:-1 act_data_pkt_fwd:0 min_seg_size_forward:32 Active Mean:0.0 Active Std:0.0 Active Max:0 Active Min:0 Idle Mean:0.0 Idle Std:0.0 Idle Max:0.0 Idle Min:0.0 Label:SMSMALWARE_ZSONE
1,Source Port:36819 Destination Port:80 Protocol:6 Flow Duration:974659 Total Fwd Packets:5 Total Backward Packets:20 Total Length of Fwd Packets:422 Total Length of Bwd Packets:23963 Fwd Packet Length Max:422 Fwd Packet Length Min:0 Fwd Packet Length Mean:84.4 Fwd Packet Length Std:188.7241373 Bwd Packet Length Max:1460 Bwd Packet Length Min:0 Bwd Packet Length Mean:1198.15 Bwd Packet Length Std:545.107452 Flow Bytes/s:25019.00665 Flow Packets/s:25.64999656 Flow IAT Mean:40610.79167 Flow IAT Std:82661.30527 Flow IAT Max:269825.0 Flow IAT Min:3.0 Fwd IAT Total:743217.0 Fwd IAT Mean:185804.25 Fwd IAT Std:230168.4361 Fwd IAT Max:489693.0 Fwd IAT Min:13.0 Bwd IAT Total:744218.0 Bwd IAT Mean:39169.36842 Bwd IAT Std:95781.43164 Bwd IAT Max:316981.0 Bwd IAT Min:3.0 Fwd PSH Flags:0 Bwd PSH Flags:0 Fwd URG Flags:0 Bwd URG Flags:0 Fwd Header Length:120 Bwd Header Length:412 Fwd Packets/s:5.129999313 Bwd Packets/s:20.51999725 Min Packet Length:0 Max Packet Length:1460 Packet Length Mean:937.8846154 Packet Length Std:683.0735437 Packet Length Variance:466589.4662 FIN Flag Count:0 SYN Flag Count:0 RST Flag Count:0 PSH Flag Count:1 ACK Flag Count:0 URG Flag Count:0 CWE Flag Count:0 ECE Flag Count:0 Down/Up Ratio:4 Average Packet Size:975.4 Avg Fwd Segment Size:84.4 Avg Bwd Segment Size:1198.15 Fwd Avg Bytes/Bulk:0 Fwd Avg Packets/Bulk:0 Fwd Avg Bulk Rate:0 Bwd Avg Bytes/Bulk:0 Bwd Avg Packets/Bulk:0 Bwd Avg Bulk Rate:0 Subflow Fwd Packets:5 Subflow Fwd Bytes:422 Subflow Bwd Packets:20 Subflow Bwd Bytes:23963 Init_Win_bytes_forward:65535 Init_Win_bytes_backward:123 act_data_pkt_fwd:1 min_seg_size_forward:20 Active Mean:0.0 Active Std:0.0 Active Max:0 Active Min:0 Idle Mean:0.0 Idle Std:0.0 Idle Max:0.0 Idle Min:0.0 Label:SMSMALWARE_ZSONE
2,Source Port:36819 Destination Port:80 Protocol:6 Flow Duration:19169 Total Fwd Packets:5 Total Backward Packets:0 Total Length of Fwd Packets:0 Total Length of Bwd Packets:0 Fwd Packet Length Max:0 Fwd Packet Length Min:0 Fwd Packet Length Mean:0.0 Fwd Packet Length Std:0.0 Bwd Packet Length Max:0 Bwd Packet Length Min:0 Bwd Packet Length Mean:0.0 Bwd Packet Length Std:0.0 Flow Bytes/s:0.0 Flow Packets/s:260.837811 Flow IAT Mean:4792.25 Flow IAT Std:9551.178614 Flow IAT Max:19119.0 Flow IAT Min:5.0 Fwd IAT Total:19169.0 Fwd IAT Mean:4792.25 Fwd IAT Std:9551.178614 Fwd IAT Max:19119.0 Fwd IAT Min:5.0 Bwd IAT Total:0.0 Bwd IAT Mean:0.0 Bwd IAT Std:0.0 Bwd IAT Max:0.0 Bwd IAT Min:0.0 Fwd PSH Flags:0 Bwd PSH Flags:0 Fwd URG Flags:0 Bwd URG Flags:0 Fwd Header Length:100 Bwd Header Length:0 Fwd Packets/s:260.837811 Bwd Packets/s:0.0 Min Packet Length:0 Max Packet Length:0 Packet Length Mean:0.0 Packet Length Std:0.0 Packet Length Variance:0.0 FIN Flag Count:0 SYN Flag Count:0 RST Flag Count:0 PSH Flag Count:0 ACK Flag Count:1 URG Flag Count:0 CWE Flag Count:0 ECE Flag Count:0 Down/Up Ratio:0 Average Packet Size:0.0 Avg Fwd Segment Size:0.0 Avg Bwd Segment Size:0.0 Fwd Avg Bytes/Bulk:0 Fwd Avg Packets/Bulk:0 Fwd Avg Bulk Rate:0 Bwd Avg Bytes/Bulk:0 Bwd Avg Packets/Bulk:0 Bwd Avg Bulk Rate:0 Subflow Fwd Packets:5 Subflow Fwd Bytes:0 Subflow Bwd Packets:0 Subflow Bwd Bytes:0 Init_Win_bytes_forward:1871 Init_Win_bytes_backward:-1 act_data_pkt_fwd:0 min_seg_size_forward:20 Active Mean:0.0 Active Std:0.0 Active Max:0 Active Min:0 Idle Mean:0.0 Idle Std:0.0 Idle Max:0.0 Idle Min:0.0 Label:SMSMALWARE_ZSONE
3,Source Port:56512 Destination Port:80 Protocol:6 Flow Duration:715219 Total Fwd Packets:3 Total Backward Packets:3 Total Length of Fwd Packets:497 Total Length of Bwd Packets:900 Fwd Packet Length Max:497 Fwd Packet Length Min:0 Fwd Packet Length Mean:165.6666667 Fwd Packet Length Std:286.9430838 Bwd Packet Length Max:900 Bwd Packet Length Min:0 Bwd Packet Length Mean:300.0 Bwd Packet Length Std:519.6152423 Flow Bytes/s:1953.247886 Flow Packets/s:8.389038882 Flow IAT Mean:143043.8 Flow IAT Std:119611.8792 Flow IAT Max:270840.0 Flow IAT Min:48.0 Fwd IAT Total:408813.0 Fwd IAT Mean:204406.5 Fwd IAT Std:289006.5623 Fwd IAT Max:408765.0 Fwd IAT Min:48.0 Bwd IAT Total:484645.0 Bwd IAT Mean:242322.5 Bwd IAT Std:292397.8464 Bwd IAT Max:449079.0 Bwd IAT Min:35566.0 Fwd PSH Flags:0 Bwd PSH Flags:0 Fwd URG Flags:0 Bwd URG Flags:0 Fwd Header Length:80 Bwd Header Length:72 Fwd Packets/s:4.194519441 Bwd Packets/s:4.194519441 Min Packet Length:0 Max Packet Length:900 Packet Length Mean:199.5714286 Packet Length Std:360.1406471 Packet Length Variance:129701.2857 FIN Flag Count:0 SYN Flag Count:0 RST Flag Count:0 PSH Flag Count:1 ACK Flag Count:0 URG Flag Count:0 CWE Flag Count:0 ECE Flag Count:0 Down/Up Ratio:1 Average Packet Size:232.8333333 Avg Fwd Segment Size:165.6666667 Avg Bwd Segment Size:300.0 Fwd Avg Bytes/Bulk:0 Fwd Avg Packets/Bulk:0 Fwd Avg Bulk Rate:0 Bwd Avg Bytes/Bulk:0 Bwd Avg Packets/Bulk:0 Bwd Avg Bulk Rate:0 Subflow Fwd Packets:3 Subflow Fwd Bytes:497 Subflow Bwd Packets:3 Subflow Bwd Bytes:900 Init_Win_bytes_forward:65535 Init_Win_bytes_backward:123 act_data_pkt_fwd:1 min_seg_size_forward:20 Active Mean:0.0 Active Std:0.0 Active Max:0 Active Min:0 Idle Mean:0.0 Idle Std:0.0 Idle Max:0.0 Idle Min:0.0 Label:SMSMALWARE_ZSONE
4,Source Port:56512 Destination Port:80 Protocol:6 Flow Duration:49 Total Fwd Packets:2 Total Backward Packets:0 Total Length of Fwd Packets:0 Total Length of Bwd Packets:0 Fwd Packet

Figure 3.18: Textual format of CicMaldroid 2017 after combining all malware families

Label	Message
ADWARE_SELFMITE	Flow ID: 172.217.2.106-10.42.0.151-443-36635-6, Source: 10.42.0.151:36635, Destination: 172.217.2.106:443, Protocol: 6.0, Timestamp: 14/06/2017 01:54:51, Duration
RANSOMWARE_SIMPLOCKER	Flow ID: 172.217.1.162-10.42.0.211-443-40670-6, Source: 10.42.0.211:40670, Destination: 172.217.1.162:443, Protocol: 6.0, Timestamp: 16/06/2017 03:55:43, Duration
ADWARE_SELFMITE	Flow ID: 172.217.1.174-10.42.0.151-443-57273-6, Source: 10.42.0.151:57273, Destination: 172.217.1.174:443, Protocol: 6.0, Timestamp: 24/08/2017 01:10:10, Duration
SMSMALWARE_ZSONE	Flow ID: 216.58.219.234-10.42.0.151-443-38357-6, Source: 10.42.0.151:38357, Destination: 216.58.219.234:443, Protocol: 6.0, Timestamp: 24/08/2017 01:10:26, Durati
SMSMALWARE_ZSONE	Flow ID: 172.217.10.138-10.42.0.42-443-58647-6, Source: 10.42.0.42:58647, Destination: 172.217.10.138:443, Protocol: 6.0, Timestamp: 16/08/2017 04:29:13, Duration
SCAREWARE_VIRUSSHIELD.	Flow ID: 180.149.134.142-10.42.0.211-80-59193-6, Source: 10.42.0.211:59193, Destination: 180.149.134.142:80, Protocol: 6.0, Timestamp: 28/08/2017 05:17:14, Durati
RANSOMWARE_SIMPLOCKER	Flow ID: 10.42.0.211-103.7.30.118-35524-80-6, Source: 10.42.0.211:35524, Destination: 103.7.30.118:80, Protocol: 6.0, Timestamp: 27/06/2017 03:44:37, Duration: 304
BENIGN	Flow ID: 192.168.1.100-10.42.0.42-8004-59388-6, Source: 10.42.0.42:59388, Destination: 192.168.1.100:8004, Protocol: 6.0, Timestamp: 16/08/2017 04:07:02, Duration
RANSOMWARE_SIMPLOCKER	Flow ID: 172.217.2.106-10.42.0.151-443-48575-6, Source: 10.42.0.151:48575, Destination: 172.217.2.106:443, Protocol: 6.0, Timestamp: 14/06/2017 01:54:51, Duration
BENIGN	Flow ID: 180.149.136.194-10.42.0.151-80-36214-6, Source: 10.42.0.151:36214, Destination: 180.149.136.194:80, Protocol: 6.0, Timestamp: 24/08/2017 01:46:30, Durati

Figure 3.19: Textual features of dataset

BENIGN	According to Gran , the company has no plans to move all production to Russia , although that is where the company is growing .
BENIGN	Technopolis plans to develop in stages an area of no less than 100,000 square meters in order to host companies working in computer technologies and telecommunications , the statement said .
SMISHING	The international electronic industry company Elcoteq has laid off tens of employees from its Tallinn facility ; contrary to earlier layoffs the company contracted the ranks of its office workers , the daily Postimees reported .
SPAMMING	With the new production plant the company would increase its capacity to meet the expected increase in demand and would improve the use of raw materials and therefore increase the production profitability .
SPAMMING	According to the company 's updated strategy for the years 2009-2012 , Basware targets a long-term net sales growth in the range of 20 % -40 % with an operating profit margin of 10 % -20 % of net sales .
SPAMMING	FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is aggressively pursuing its growth strategy by increasingly focusing on technologically more demanding HDI printed circuit boards PCBs .
SPAMMING	For the last quarter of 2010 , Componenta 's net sales doubled to EUR131m from EUR76m for the same period a year earlier , while it moved to a zero pre-tax profit from a pre-tax loss of EUR7m .
SPAMMING	In the third quarter of 2010 , net sales increased by 5.2 % to EUR 205.5 mn , and operating profit by 34.9 % to EUR 23.5 mn .
SPAMMING	Operating profit rose to EUR 13.1 mn from EUR 8.7 mn in the corresponding period in 2007 representing 7.7 % of net sales .
SPAMMING	Operating profit totalled EUR 21.1 mn , up from EUR 18.6 mn in 2007 , representing 9.7 % of net sales .
SPAMMING	TeliaSonera TLSN said the offer is in line with its strategy to increase its ownership in core business holdings and would strengthen Eesti Telekom 's offering to its customers .
SPAMMING	STORA ENSO , NORSKE SKOG , M-REAL , UPM-KYMMENE Credit Suisse First Boston (CFSB) raised the fair value for shares in four of the largest Nordic forestry groups .
SPAMMING	A purchase agreement for 7,200 tons of gasoline with delivery at the Hamina terminal , Finland , was signed with Neste Oil OYj at the average Platts index for this September plus eight US dollars per month .
SPAMMING	Finnish Talentum reports its operating profit increased to EUR 20.5 mn in 2005 from EUR 9.3 mn in 2004 , and net sales totaled EUR 103.3 mn , up from EUR 96.4 mn .
SPAMMING	Clothing retail chain Sepp+Æl+Æ 's sales increased by 8 % to EUR 155.2 mn , and operating profit rose to EUR 31.1 mn from EUR 17.1 mn in 2004 .
SPAMMING	Consolidated net sales increased 16 % to reach EUR74 .8 m , while operating profit amounted to EUR0 .9 m compared to a loss of EUR0 .7 m in the prior year period .
SPAMMING	Foundries division reports its sales increased by 9.7 % to EUR 63.1 mn from EUR 57.5 mn in the corresponding period in 2006 , and sales of the Machine Shop division increased by 16.4 % to EUR 41.2 mn from EUR 35.4 mn in th
SPAMMING	HELSINKI (AFX) - Shares closed higher , led by Nokia after it announced plans to team up with Sanyo to manufacture 3G handsets , and by Nokian Tyres after its fourth-quarter earnings report beat analysts ' expectations , deale
SPAMMING	Incap Contract Manufacturing Services Pvt Ltd , a subsidiary of Incap Corporation of Finland , plans to double its revenues by 2007-2008 .
SPAMMING	Its board of directors will propose a dividend of EUR0 .12 per share for 2010 , up from the EUR0 .08 per share paid in 2009 .
SPAMMING	Lifetree was founded in 2000 , and its revenues have risen on an average by 40 % with margins in late 30s .
SPAMMING	(Filippova) A trilateral agreement on investment in the construction of a technology park in St Petersburg was to have been signed in the course of the forum , Days of the Russian Economy , that opened in Helsinki today .
SPAMMING	MegaFon 's subscriber base increased 16.1 % in 2009 to 50.5 million users as of December 31 , while its market share by the number of customers amounted to 24 % as of late 2009 , up from 23 % as of late 2008 , according to T
SPAMMING	Net income from life insurance doubled to EUR 6.8 mn from EUR 3.2 mn , and net income from non-life insurance rose to EUR 5.2 mn from EUR 1.5 mn in the corresponding period in 2009 .
SPAMMING	Net sales increased to EUR193 .3 m from EUR179 .9 m and pretax profit rose by 34.2 % to EUR43 .1 m. (EUR1 = USD1 .4)
SPAMMING	Net sales surged by 18.5 % to EUR167 .8 m. Teleste said that EUR20 .4 m , or 12.2 % , of the sales came from the acquisitions made in 2009 .
SPAMMING	Nordea Group 's operating profit increased in 2010 by 18 percent year-on-year to 3.64 billion euros and total revenue by 3 percent to 9.33 billion euros .

Figure 3.20: Textual features of dataset

label	text
1	The file with the name '0' and hash '0124e21d-018c-4ce0-92a3-b9e205a76bc0.dll' has properties including a file size of 79755c51e413ed3c6be4635fd729a6e1 bytes, multiple zero values indicating the absence of specific permissions or intents, a
1	The file with the name '1' and hash '05c8318f98a5d301d80000009c316005.vertdll.dll' has properties including a file size of 95e19f3657d34a432eada93221b0ea16 bytes, multiple zero values indicating the absence of specific permissions or intent
1	The file with the name '2' and hash '06054fba-5619-4a86-a861-ffb0464bef5d.dll' has properties including a file size of 85c32641d77a54e19ba8ea4ab305c791 bytes, multiple zero values indicating the absence of specific permissions or intents, a
1	The file with the name '3' and hash '075822ac99a5d301660400009c316005.adhapi.dll' has properties including a file size of 62e3b959d982ef534b66f819fe15f085 bytes, multiple zero values indicating the absence of specific permissions or intent
1	The file with the name '4' and hash '090607dd9ba5d301ca0900009c316005.SensorsNativeApi.V2.dll' has properties including a file size of ae38c5f7d313ad0ff3bfb8826476767f bytes, multiple zero values indicating the absence of specific permiss
1	The file with the name '5' and hash '0aedb43f9ba5d3014e0600009c316005.wlanapi.dll' has properties including a file size of 28c98e000d02f8bfd9d050eb4e4c5d2e bytes, multiple zero values indicating the absence of specific permissions or inter
1	The file with the name '6' and hash '0bc194f9-b102-4833-85bd-603e216a9274.dll' has properties including a file size of 706463ecc8e48e9e3a2a12a0cfcb8858 bytes, multiple zero values indicating the absence of specific permissions or intents, a
1	The file with the name '7' and hash '0c7f9fdc9ba5d301c60900009c316005.wscsvc.dll' has properties including a file size of 39da352fad220e83ce64de8dccb9736b bytes, multiple zero values indicating the absence of specific permissions or intent
1	The file with the name '8' and hash '1.0.154_chromesetup_154_59.exe' has properties including a file size of e11e70ba243800626d17e3ffa6c9fb71 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory u
1	The file with the name '9' and hash '1035f45d9ca5d3015b0a00009c316005.srclient.dll' has properties including a file size of a5a106d5d03e6e59b0282e72bc9420f1 bytes, multiple zero values indicating the absence of specific permissions or inter
1	The file with the name '10' and hash '109b6ed1-e133-407f-b839-3fdd1ebc0d85.dll' has properties including a file size of 212fdb291e0b5d8f34771bc1338c20b3 bytes, multiple zero values indicating the absence of specific permissions or intents, a
1	The file with the name '11' and hash '112ff2d99ca5d301590b00009c316005.mscoreei.dll' has properties including a file size of e313747a38c6bc267c9167d92bd2cb7f bytes, multiple zero values indicating the absence of specific permissions or int
1	The file with the name '12' and hash '11b25aa499a5d301370400009c316005.pngfilt.dll' has properties including a file size of a2f43fac128db67452726d096fb77768 bytes, multiple zero values indicating the absence of specific permissions or inter
1	The file with the name '13' and hash '128906669ca5d301730a00009c316005.TimeBrokerClient.dll' has properties including a file size of d2e6d99c9624a3f6ac7c8679f0e23e00 bytes, multiple zero values indicating the absence of specific permissi
1	The file with the name '14' and hash 'MSBuildTaskHost.resources (10).dll' has properties including a file size of fd3313b819e4d121cb1551362420609c bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '15' and hash 'MSBuildTaskHost.resources (11).dll' has properties including a file size of 462da78784b24814d0538fa6dc2ced28 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '16' and hash 'MSBuildTaskHost.resources (12).dll' has properties including a file size of d002e0973af380781e85fdc8b0ddf683 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '17' and hash 'MSBuildTaskHost.resources (13).dll' has properties including a file size of 333669b5ac0e3418e9d72393a2be2d8c bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '18' and hash 'MSBuildTaskHost.resources (14).dll' has properties including a file size of 5570b99ac52c4fd941331508b0ec2786 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '19' and hash 'MSBuildTaskHost.resources (15).dll' has properties including a file size of ec9de465ca85f523ea84c07368741fcc bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '20' and hash 'MSBuildTaskHost.resources (16).dll' has properties including a file size of de12836d9fb3050b1c4c5c47770624ee bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '21' and hash 'MSBuildTaskHost.resources (17).dll' has properties including a file size of 7bc94f59e94e9a6a38d83ead7b1eac34 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '22' and hash 'MSBuildTaskHost.resources (18).dll' has properties including a file size of 328f0ce88bd8a40fe85f0ba5d8e5f703 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '23' and hash 'MSBuildTaskHost.resources (19).dll' has properties including a file size of 5025f28b6052b19ea0a1c4332407d06b bytes, multiple zero values indicating the absence of specific permissions or intents, a memory
1	The file with the name '24' and hash 'MSBuildTaskHost.resources (2).dll' has properties including a file size of 493f30fa92f8a9328eb0fe7602d14967 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory u
1	The file with the name '25' and hash 'MSBuildTaskHost.resources (20).dll' has properties including a file size of e6868c88fa398c92624e071018fb72d1 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory

Figure 3.21: Textual features of ransomware dataset

Class	text
Trojan	73217 10.42.0.42-121.14.255.84-49975-80-6 10.42.0.42 49975 121.14.255.84 80 6 17/07/2017 01:18:33 10743584 4 4 372.0 672.0 372.0 0.0 93.0 186.0 672.0 0.0 168.0 336.0 97.17427629364651 0.7446304696831151 1534797.714285714 373
Trojan	72089 172.217.6.226-10.42.0.42-443-49169-17 10.42.0.42 49169 172.217.6.226 443 17 17/07/2017 10:25:25 254217 6 7 3191.0 5246.0 1350.0 38.0 531.8333333333334 645.2160620030058 1350.0 30.0 749.4285714285714 678.20642313454
Benign	96676 10.42.0.1-10.42.0.42-53-37749-17 10.42.0.42 37749 10.42.0.1 53 17 30/06/2017 07:16:12 1023244 1 1 30.0 179.0 30.0 30.0 30.0 0.0 179.0 179.0 179.0 0.0 204.2523581863172 1.954568020921696 1023244.0 0.0 1023244.0 1023244.0 0
Trojan	42891 10.42.0.1-10.42.0.42-53-41352-17 10.42.0.42 41352 10.42.0.1 53 17 13/07/2017 03:48:44 286483 1 1 40.0 106.0 40.0 40.0 40.0 0.0 106.0 106.0 106.0 0.0 509.6288435963042 6.981217035565811 286483.0 0.0 286483.0 286483.0 0.0 0.
Benign	169326 10.42.0.151-107.22.241.77-44353-443-6 10.42.0.151 44353 107.22.241.77 443 6 05/07/2017 10:47:35 65633087 12 10 767.0 5622.0 403.0 0.0 63.91666666666667 130.00171910285252 1448.0 0.0 562.1999999999999 649.53894759
Trojan	34510 10.42.0.211-10.42.0.1-6021-53-17 10.42.0.211 6021 10.42.0.1 53 17 11/07/2017 05:01:34 251336 1 1 37.0 182.0 37.0 37.0 37.0 0.0 182.0 182.0 182.0 0.0 871.3435401215902 7.957475252251966 251336.0 0.0 251336.0 251336.0 0.0 0.
Trojan	59506 10.42.0.42-74.217.63.24-38871-443-6 10.42.0.42 38871 74.217.63.24 443 6 14/07/2017 01:48:47 3096 3 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 968.9922480620156 1548.0 439.8204178980326 1859.0 1237.0 3096.0 1548.0 439.820
Benign	98047 10.42.0.42-66.231.239.96-49387-443-6 10.42.0.42 49387 66.231.239.96 443 6 02/07/2017 08:24:19 473 1 2 46.0 31.0 46.0 46.0 46.0 0.0 31.0 0.0 15.5 21.920310216782973 162790.6976744186 6342.494714587738 236.5 132.2289680
Trojan	44044 10.42.0.1-10.42.0.42-53-34743-17 10.42.0.42 34743 10.42.0.1 53 17 13/07/2017 04:03:57 557409 1 1 25.0 79.0 25.0 25.0 25.0 0.0 79.0 79.0 79.0 0.0 186.57754001101523 3.588029615596447 557409.0 0.0 557409.0 557409.0 0.0 0.0 0.
Trojan	84350 172.217.10.1-10.42.0.151-443-51786-6 172.217.10.1 443 10.42.0.151 51786 6 04/07/2017 10:07:47 314 2 0 55.0 0.0 55.0 0.0 27.5 38.8908729653 0.0 0.0 0.0 0.0 175159.23566879 6369.4267515924 314.0 0.0 314.0 314.0 314.0 314.0 0.
Benign	140308 180.76.184.128-10.42.0.42-80-47733-6 10.42.0.42 47733 180.76.184.128 80 6 05/07/2017 03:30:54 518061 3 3 551.0 0.0 551.0 0.0 183.66666666666663 318.1199983234838 0.0 0.0 0.0 0.0 1063.5813157137866 11.58164772102127 1
Benign	149339 172.217.9.227-10.42.0.42-443-38191-6 10.42.0.42 38191 172.217.9.227 443 6 05/07/2017 09:35:37 46143837 9 9 448.0 4792.0 242.0 0.0 49.77777777777778 99.18389206138488 1418.0 0.0 532.4444444444443 671.8467293793858 1
Benign	90887 10.42.0.42-112.80.248.220-39888-443-6 10.42.0.42 39888 112.80.248.220 443 6 30/06/2017 04:54:01 34659660 26 28 6359.0 9213.0 1452.0 0.0 244.5769230769231 492.9489363475225 1460.0 0.0 329.0357142857142 515.19773134
Trojan	30597 10.42.0.211-10.42.0.1-62945-53-17 10.42.0.211 62945 10.42.0.1 53 17 11/07/2017 03:57:53 663034 1 1 30.0 118.0 30.0 30.0 30.0 0.0 118.0 118.0 118.0 0.0 223.21630564948404 3.016436562830865 663034.0 0.0 663034.0 663034.0 0.
Benign	114034 140.205.230.8-10.42.0.42-80-35291-6 140.205.230.8 80 10.42.0.42 35291 6 02/07/2017 04:50:25 7 2 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 285714.28571428574 7.0 0.0 7.0 7.0 7.0 7.0 0.0 7.0 7.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 4
Benign	98060 10.42.0.42-66.198.24.250-35120-443-6 10.42.0.42 35120 66.198.24.250 443 6 02/07/2017 08:20:58 4745734 40 58 1578.0 64471.0 552.0 0.0 39.44999999999999 96.17237462507433 1460.0 0.0 1111.5689655172414 574.8538216041
Benign	139010 216.58.217.68-10.42.0.42-443-46251-6 10.42.0.42 46251 216.58.217.68 443 6 05/07/2017 02:03:16 72918838 20 20 4556.0 5871.0 1368.0 0.0 227.8 449.3343146431056 1418.0 0.0 293.54999999999995 453.13498337459527 142.994
Trojan	13077 172.217.12.162-10.42.0.211-443-60065-6 10.42.0.211 60065 172.217.12.162 443 6 11/07/2017 12:02:58 566018 31 42 2920.0 57213.0 978.0 0.0 94.19354838709675 292.52879281133323 1418.0 383.0 1362.2142857142856 220.34557
Benign	124498 10.42.0.1-10.42.0.42-53-64235-17 10.42.0.42 64235 10.42.0.1 53 17 02/07/2017 10:18:27 1482 1 1 36.0 246.0 36.0 36.0 36.0 0.0 246.0 246.0 246.0 0.0 190283.4008097166 1349.527665317139 1482.0 0.0 1482.0 1482.0 0.0 0.0 0.0 0.0
Benign	116573 202.77.129.230-10.42.0.42-80-56764-6 10.42.0.42 56764 202.77.129.230 80 6 02/07/2017 05:36:30 4542518 3 4 804.0 1964.0 804.0 0.0 268.0 464.1896164284591 1460.0 0.0 491.0 688.3051648796484 609.3536668429272 1.5409955
Benign	131470 208.80.154.224-10.42.0.42-80-47994-6 10.42.0.42 47994 208.80.154.224 80 6 04/07/2017 08:12:20 5410908 4 4 402.0 490.0 402.0 0.0 100.5 201.0 490.0 0.0 122.5 245.00000000000003 164.85218377396177 1.4784949217395675 772
Benign	113980 10.42.0.42-104.193.88.109-57723-80-6 10.42.0.42 57723 104.193.88.109 80 6 02/07/2017 04:50:03 21856196 2 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0915072320910738 21856196.0 0.0 21856196.0 21856196.0 21856196.0 218
Benign	157416 10.42.0.42-54.230.51.28-44346-443-6 54.230.51.28 443 10.42.0.42 44346 6 05/07/2017 02:13:33 143 2 0 31.0 0.0 31.0 0.0 15.5 21.920310216782973 0.0 0.0 0.0 0.0 216783.2167832168 13986.013986013986 143.0 0.0 143.0 143.0 143
Trojan	79012 205.185.216.10-10.42.0.42-80-33806-6 10.42.0.42 33806 205.185.216.10 80 6 17/07/2017 02:35:22 1397782 2 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.4308382852261654 1397782.0 0.0 1397782.0 1397782.0 1397782.0 1397782.0
Benign	146071 172.217.10.129-10.42.0.42-443-49472-6 10.42.0.42 49472 172.217.10.129 443 6 05/07/2017 06:37:23 65339441 9 8 653.0 5638.0 250.0 0.0 72.55555555555556 107.51408176503102 1418.0 0.0 704.75 681.6339088647848 96.281815
Benign	148759 10.42.0.42-98.139.180.149-35305-443-6 10.42.0.42 35305 98.139.180.149 443 6 05/07/2017 09:11:36 115680306 17 18 813.0 7881.0 297.0 0.0 47.823529411764696 84.69004907168672 1460.0 0.0 437.83333333333337 625.538379

Figure 3.22: Textual features of trojan dataset

text	Label
In our Android application, we utilize the ACCESS_NETWORK_STATE permission to check network connectivity, the CAI	malware
ACCESS_NETWORK_STATE BATTERY_STATS INTERNET READ_PHONE_STATE RECEIVE_BOOT_COMPLETED RECEIVE_S	malware
ACCESS_NETWORK_STATE DISABLE_KEYGUARD GET_TASKS INTERNET KILL_BACKGROUND_PROCESSES READ_PHONE	malware
BATTERY_STATS INTERNET READ_PHONE_STATE RECEIVE_BOOT_COMPLETED RECEIVE_SMS SEND_SMS Ljavax/crypt	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware
ACCESS_NETWORK_STATE INTERNET READ_EXTERNAL_STORAGE READ_PHONE_STATE RECEIVE_BOOT_COMPLETED	malware
ACCESS_NETWORK_STATE INTERNET READ_EXTERNAL_STORAGE READ_PHONE_STATE RECEIVE_BOOT_COMPLETED	malware
ACCESS_NETWORK_STATE INTERNET READ_PHONE_STATE RECEIVE_BOOT_COMPLETED WAKE_LOCK Ljava/lang/refl	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware
ACCESS_NETWORK_STATE DISABLE_KEYGUARD GET_TASKS INTERNET KILL_BACKGROUND_PROCESSES READ_PHONE	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware
ACCESS_NETWORK_STATE BIND_DEVICE_ADMIN CAMERA GET_ACCOUNTS GET_TASKS INTERNET READ_CONTACTS I	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware
ACCESS_NETWORK_STATE INTERNET READ_PHONE_STATE RECEIVE_BOOT_COMPLETED WAKE_LOCK Ljava/lang/refl	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware
ACCESS_NETWORK_STATE DISABLE_KEYGUARD GET_TASKS INTERNET KILL_BACKGROUND_PROCESSES READ_PHONE	malware
ACCESS_NETWORK_STATE CAMERA GET_TASKS INTERNET READ_EXTERNAL_STORAGE READ_PHONE_STATE RECEIVE	malware
ACCESS_NETWORK_STATE DISABLE_KEYGUARD GET_TASKS INTERNET KILL_BACKGROUND_PROCESSES READ_PHONE	malware
ACCESS_CHECKIN_PROPERTIES ACCESS_COARSE_LOCATION ACCESS_FINE_LOCATION ACCESS_LOCATION_EXTRA_CO	malware
ACCESS_NETWORK_STATE CAMERA GET_ACCOUNTS GET_TASKS INTERNET READ_CONTACTS READ_EXTERNAL_STOR	malware
ACCESS_NETWORK_STATE DISABLE_KEYGUARD GET_TASKS INTERNET KILL_BACKGROUND_PROCESSES READ_PHONE	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware
ACCESS_NETWORK_STATE DISABLE_KEYGUARD GET_TASKS INTERNET KILL_BACKGROUND_PROCESSES READ_PHONE	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware
DISABLE_KEYGUARD GET_TASKS INTERNET KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETED SYSTEM_A	malware
ACCESS_WIFI_STATE CHANGE_WIFI_STATE GET_TASKS KILL_BACKGROUND_PROCESSES RECEIVE_BOOT_COMPLETEI	malware

Figure 3.23: Textual features of tuandromd dataset

	combined	Sentence	Malware/Attack Type
0	WickedRose_andNCPH "Wicked Rose" And The Ncph Hacking Group iDefense https://app.box.co	The hacking group known as 'Wicked Rose' an	Wicked Rose (Hacking Group)
1	Fritz_HOW-CHINA-WILL-USE-CYBER-WARFARE(Oct-01-08) How China Will Use Cyber Warfare Ja	Jason Fritz's report titled 'How China Will Use	Chinese Cyber Warfare
2	556_10535_798405_Annex87_CyberAttacks Russian Cyberwar On Georgia Georgia Gov https://a	The Russian cyberwar on Georgia in 2008 mar	Russian Cyberwar (Georgia)
3	Ashmore_Impact-of-Alleged-Russian-Cyber-Attacks(Jan-18-09) Impact Of Alleged Russian Cyber /	William C. Ashmore's 2009 report addresses t	Russian Cyber Attacks
4	ghostnet Tracking Ghostnet: Investigating A Cyber Espionage Network Information Warfare Moni	'Tracking Ghostnet' investigates a complex cyl	Ghostnet (Cyber Espionage)
5	Case_Study_Operation_Aurora_V11 Case Study: Operation Aurora Triumfant https://app.box.co	No clear malware type identified.	Unknown
6	Aurora_Botnet_Command_Structure The Command Structure Of The Aurora Botnet Damballa ht	No clear malware type identified.	Unknown
7	McAfee_Operation_Aurora Combating Aurora McAfee https://app.box.com/s/jhy5k76ox6z8sy6t	No clear malware type identified.	Unknown
8	Aurora_HBGARY_DRAFT Operation Aurora: Detect, Diagnose, Respond HBGary https://app.box.c	No clear malware type identified.	Unknown
9	HBGary_Operation_Aurora Operation Aurora HBGary https://app.box.com/s/fjb89qr1vnk2ox0vll	No clear malware type identified.	Unknown
10	how_can_u_tell_Aurora How Can I Tell If I Was Infected By Aurora? McAfee https://app.box.com	No clear malware type identified.	Unknown
11	in-depth_analysis_of_hydraq_final_231538 In-Depth Analysis Of Hydraq: The Face Of Cyberwar E	No clear malware type identified.	Unknown
12	Shadowserver_shadows-in-the-cloud Shadows In The Cloud: Investigating Cyber Espionage 2.0 Sh	No clear malware type identified.	Unknown
13	WashingtonPost_2010-Defense-official-discloses-cyberattack(08-24-2010) Defense official disclo	No clear malware type identified.	Unknown
14	MSUpdaterTrojanWhitepaper The Msupdater Trojan And Ongoing Targeted Attacks Seculert, Zsc	No clear malware type identified.	Unknown
15	w32_stuxnet_dossier W32.Stuxnet Dossier Symantec https://app.box.com/s/rpdy3pk00bmkhgm	No clear malware type identified.	Unknown
16	wp-global-energy-cyberattacks-night-dragon Global Energy Cyberattacks: Night Dragon McAfee h	No clear malware type identified.	Unknown
17	Alerts DL-2011 Alerts-A-2011-02-18-01 Night Dragon Attachment 1 Night Dragon: Specific Protec	No clear malware type identified.	Unknown
18	Stuxnet_Under_the_Microscope Stuxnet Under The Microscope ESET https://app.box.com/s/2m	No clear malware type identified.	Unknown
19	C5_APT_ADecadeInReview Advanced Persistent Threats: A Decade In Review Command Five Pty	No clear malware type identified.	Unknown
20	shady_rat_vanity Operation Shady Rat: Unprecedented Cyber-Espionage Campaign And Intellect	No clear malware type identified.	Unknown
21	HTran_and_the_Advanced_Persistent_Threat Htran And The Advanced Persistent Threat Dell Sec	No clear malware type identified.	Unknown
22	wp-operation-shady-rat Revealed: Operation Shady Rat McAfee https://app.box.com/s/a086wzc	No clear malware type identified.	Unknown
23	wp_dissecting-lurid-apt The Lurid Downloader Trend Micro https://app.box.com/s/7s9bvquu64v	No clear malware type identified.	Unknown
24	C5_APT_SKHack Sk Hack By An Advanced Persistent Threat Command Five Pty Ltd https://app.bo	No clear malware type identified.	Unknown
25	tb_advanced_persistent_threats Alleged Apt Intrusion Set: 1.Php Group Zscaler, ThreatLabz https	No clear malware type identified.	Unknown

Figure 3.24: Textual features of APTNotes dataset

SCAREWA	On 27/06/2017 at 03:27:43, a suspicious flow with ID 10.42.0.211-123.125.125.56-59243-443-6 was detected, where the source IP 123.125.125.56 (port 443) communicated with the destination IP 10.42.0.211 (port 59243) using protocol 6.0. Th
RANSOMV	The file with hash '0124e21d-018c-4ce0-92a3-b9e205a76bc0.dll' has properties including a file size of 79755c51e413ed3c6be4635fd729a6e1 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory usage of
ADWARE	Your browser has been redirected to BestDeals.com for the best online shopping discounts!
ADWARE	In our Android application, we utilize the ACCESS_NETWORK_STATE permission to check network connectivity, the CAMERA permission to access the device's camera for taking pictures using Camera.open and Camera.takePicture, the GET_ACCC
RANSOMV	The file with hash '05c8318f98a5d301d80000009c316005.vertdll.dll' has properties including a file size of 95e19f3657d34a432eada93221b0ea16 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory usag
SMSMALW	In our Android application, we utilize the SmsManager class to send SMS messages with the SEND_SMS permission, register and unregister broadcast receivers using Context.registerReceiver and Context.unregisterReceiver for handling the RECEI
TROJAN	This malware utilizes several Android permissions and functionalities to compromise user privacy and device security. By leveraging ACCESS_NETWORK_STATE, it monitors network connectivity, while BATTERY_STATS allows it to access battery u
TROJAN	On March 17, 2021, at 8:02 AM, the application with the package name com.firstchoice.myfirstchoice generated a significant data entry with a unique identifier 000001A94F46A0C3DDA514E1F24E675648835BBA5EF3C3AA72D9C378534FCAD6, re
BENIGN	On June 16, 2017, at 03:55:47, a suspicious network flow was observed between IP addresses 10.42.0.211 (source) and 172.217.2.174 (destination), with source port 51023 and destination port 443, using protocol 6 (TCP). The flow, which lasted f
RANSOMV	The file with hash '090607dd9ba5d301ca0900009c316005.SensorsNativeApi.V2.dll' has properties including a file size of ae38c5f7d313ad0ff3bfb8826476767f bytes, multiple zero values indicating the absence of specific permissions or intents, a
TROJAN	On April 6, 2021, at 11:00 AM, the application with the package name com.firstchoice.myfirstchoice processed data associated with the unique identifier 000001A94F46A0C3DDA514E1F24E675648835BBA5EF3C3AA72D9C378534FCAD6, which in
BENIGN	On June 16, 2017, at 04:00:36, a network flow was detected between IP address 10.42.0.211 (source) and 199.59.148.73 (destination), using source port 33772 and destination port 443, with TCP protocol 6. The flow lasted for 453,887 millisecon
RANSOMV	The file with the hash '11b25aa499a5d301370400009c316005.pngfilt.dll' has properties including a file size of a2f43fac128db67452726d096fb77768 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory u
SMSMALW	A meeting is scheduled for 10 AM tomorrow, as noted in the message from evanmccann@gmail.com, received on April 19, 2024, at 12:21 AM, with a message length of 26 characters.. This message was sent on unknown date at unknown time by
SPYWARE	The application requests permissions such as ACCESS_NETWORK_STATE, BATTERY_STATS, and INTERNET, and utilizes Java methods like Cipher.doFinal and TelephonyManager.getNetworkCountryIso to manage network states and obtain the cou
SPYWARE	On January 1, 1980, at 0:00, the application with the package name com.fearless.teengirl version 3.0, having an identifier of 000010F33B578B4B5C33724520E38BD44485705DC1B6ADDE4975EBBA7F114355 DDE9CA9CC8C9FD67B27E68A637D101
RANSOMV	The file with hash 'MSBuildTaskHost.resources (12).dll' has properties including a file size of d002e0973af380781e85fdc8b0ddf683 bytes, multiple zero values indicating the absence of specific permissions or intents, a memory usage of 0 bytes, 81
SCAREWA	On June 27, 2017, at 03:17:21, a network flow was recorded between the source IP 23.203.49.224 (port 443) and the destination IP 10.42.0.211 (port 34092) using protocol 6.0. The flow, lasting 13 seconds, included 2 forward packets with a total
TROJAN	The application requests permissions such as ACCESS_NETWORK_STATE, INTERNET, READ_PHONE_STATE, RECEIVE_BOOT_COMPLETED, and WAKE_LOCK, and utilizes Java methods like Method.invoke and Cipher.doFinal to manage network sta
ADWARE	On January 1, 1980, at 00:00, the application com.fearless.teengirl version 3.0, with identifiers 000010F33B578B4B5C33724520E38BD44485705DC1B6ADDE4975EBBA7F114355 DDE9CA9CC8C9FD67B27E68A637D1015E3852AD4E 8CEE4A742DD0
SPYWARE	On June 16, 2017, at 03:55:40, a network flow was recorded from the source IP 10.42.0.211 (port 43070) to the destination IP 239.255.255.250 (port 1900) using protocol 17 (UDP). The flow lasted 598,272 seconds and included 6 forward packets
ADWARE	On June 14, 2017, at 01:54:51, a network flow was observed between source IP 10.42.0.151 (port 36635) and destination IP 172.217.2.106 (port 443), using protocol 6 (TCP). The flow lasted 22,287 seconds and included 1 forward packet and 1 ba
RANSOMV	The file with the hash 'MSBuildTaskHost.resources (16).dll' has properties including a file size of de12836d9fb3050b1c4c5c47770624ee bytes, multiple zero values indicating the absence of specific permissions or intents, a memory usage of 0 byte
SPYWARE	On February 8, 2021, at 13:59, the application com.placz.cricketpakistan version 2.0, identified by the hexadecimal strings 0000120C599BA69C7907706CA1BDBBF98CCD884891D68CDE4D87BB530AEAF015 9DB718A09CE7DEEFFAA2192A6A19A42
TROJAN	On January 1, 1981, at 01:01, the application com.northcube.sleepcycle, identified by the hexadecimal strings 00001438D92DC8AAA7B1BCEDDC4BEBC12E3104C02142FEAE1F929E3A0C2BC719 6AA10B10EE30A752552BF5A0A639836E181648E6 9
ADWARE	On June 16, 2017, at 04:04:52, a network flow was recorded between source IP 10.42.0.211 (port 45130) and destination IP 23.53.114.188 (port 443), using protocol 6 (TCP). The flow lasted 3 seconds and included 2 forward packets, with no data

Figure 3.25: Textual features of Final dataset

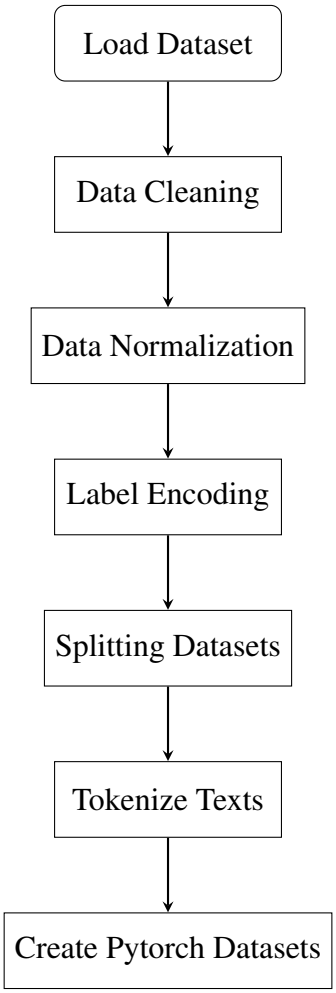


Figure 3.26: Flowchart for Preprocessing

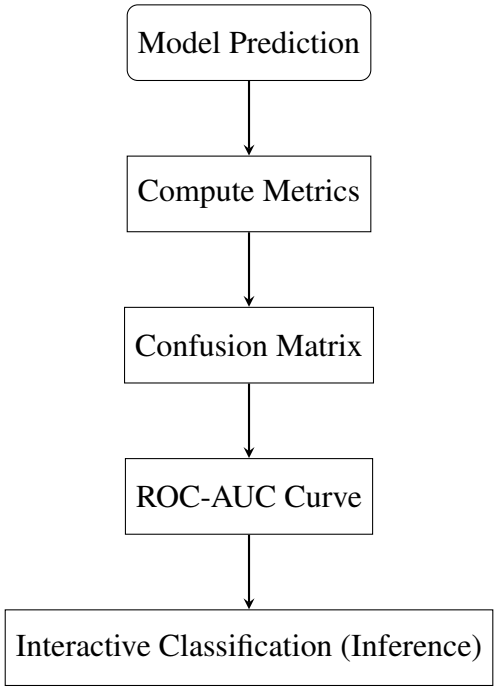


Figure 3.27: Flowchart for Postprocessing

4 RESULTS

4.1 Inference Output on SecureBERT

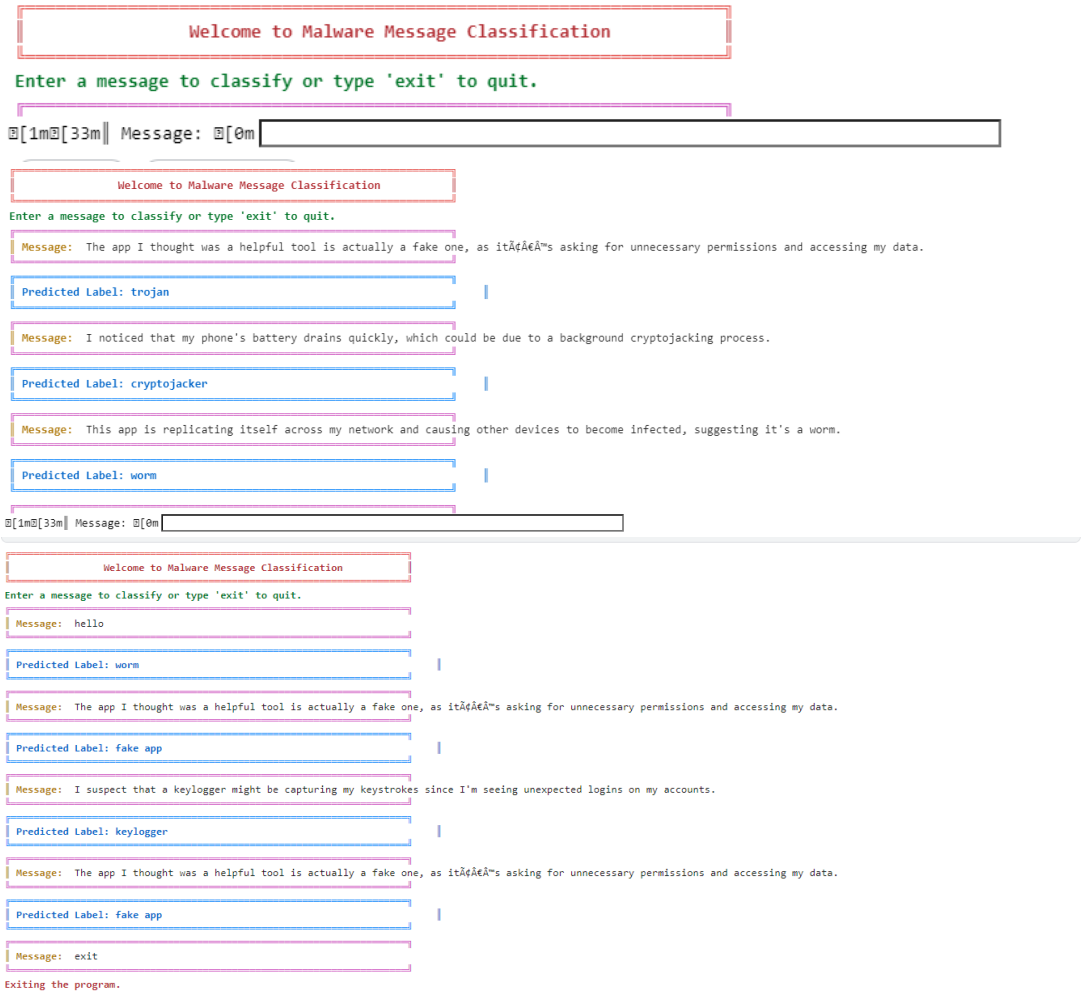


Figure 4.1: Interactive Classification (Inference) Screenshots

The SecureBERT model fine-tuned for cybersecurity tasks, employs the **RobertaTokenizer** for processing input text data. The tokenizer plays an essential role in preparing the text for model inference by converting raw text into tokens, which are numerical representations that the model can process.

4.2 Outputs for Various Scenarios

Malware Types with their labels

0. Scareware
1. Ransomware
2. Adware
3. SMS Malware

4. Trojan
5. Benign
6. Spyware
7. Polymorphic
8. Downloader
9. Cryptojacker
10. Worm
11. Fake App
12. Keylogger

4.2.1 Detailed Explanation

- **Text Input:** The user provides a text prompt, which could be a description of an incident related to Android malware. This text serves as input for the SecureBERT model to classify.
- **Tokenization:** The **RobertaTokenizer** processes the input by splitting the text into smaller subword units called tokens and mapping them to corresponding numerical IDs based on the pre-trained vocabulary of the tokenizer. For instance, the input sentence "This app may cause unauthorized access to data" would be tokenized into smaller units such as '[This, app, may, cause, unauthorized, access, to, data]' and subsequently mapped to their respective token IDs.
- **Input Formatting:** After tokenization, the tokenizer adds special tokens like '[CLS]' (indicating the start of a sequence) and '[SEP]' (indicating the end of a sequence). These ensure that the input format adheres to the SecureBERT model's requirements. Padding is also applied when necessary to standardize sequence lengths.
- **SecureBERT Model Processing:** The tokenized input is then passed through the SecureBERT model. SecureBERT, a fine-tuned variant of BERT, leverages self-attention mechanisms to understand relationships between tokens and the context of malware behavior.
- **Classification:** SecureBERT processes the input and outputs a probability distribution over various malware categories (e.g., Worm, Trojan,

Spyware, SMS Malware, etc.). The model selects the malware type with the highest probability, representing the input text's most likely classification.

- **Result:** The predicted malware label is returned to the user, indicating which category of Android malware the input text corresponds to.

4.3 Best-Case and Worst-Case Scenarios

In this section, we define the best-case and worst-case scenarios for the project .These scenarios gives the outline about ideal and least favorable outcomes for the project which can help to set realistic goals and expectations.

4.3.1 Best-Case Scenario

1. High Accuracy and Robust Performance:

Definition: The model achieves high accuracy in classifying malware into the correct categories with minimal errors. It performs consistently well across all classes, including both common and rare malware types.

2. Effective Malware Detection and Classification:

Definition: The model successfully identifies and classifies a broad spectrum of malware types, including sophisticated or novel variants.

3. Minimal Overfitting and Underfitting:

Definition: The model is well-tuned to the training data without overfitting or underfitting, demonstrating balanced performance on both training and validation datasets.

4. Scalable and Efficient Integration:

Definition: The model integrates seamlessly into existing cybersecurity frameworks or applications and performs efficiently in real-time scenarios.

5. Positive User and Stakeholder Feedback:

Definition: End-users and stakeholders find the model's predictions accurate, useful, and reliable for practical applications.

4.3.2 Worst-Case Scenario

1. Low Accuracy and Poor Performance:

Definition: The model struggles to achieve high accuracy and fails to classify malware correctly, leading to a high number of misclassifications.

2. Ineffective Malware Detection:

Definition: The model fails to detect and classify malware accurately, resulting in either false negatives (malware not detected) or false positives (benign messages classified as malware).

3. Significant Overfitting or Underfitting:

Definition: The model exhibits overfitting or underfitting, impacting its ability to generalize well to unseen data.

4. Integration and Performance Issues:

Definition: The model faces challenges in integration or exhibits performance issues in real-time scenarios.

5. Negative User and Stakeholder Feedback:

Definition: Users and stakeholders are dissatisfied with the model's performance, finding it unreliable or inadequate for practical use.

SecureBERT:(Best Case Scenario)

In our experiments, we evaluated the performance of the model using different configurations of the hyperparameters `num_train_epoch` and `gradient_accumulation`

4.3.3 Best Case Results

In the best case scenario of the model, we selected:

- `learningrate` = 1e-5
- `epoch` = 10,15
- `batchsize` = 32

The table describes columns representing different performance metrics across four epochs. Each row corresponds to the epoch number and its associated training loss, validation loss, accuracy, precision, recall, and F1 score, providing a structured way to view the progression of the machine learning model's training performance. It can be observed that there is a slight increase in training loss in 8th and 9th epoch, which might indicative bit of overfitting, but overall the performance is good.

Behavior of Training and Validation Loss Functions

Here, the x-axis represents the epochs starting from 0 to 9,i.e 10 epochs in total. There is a rapid drop in training loss after 1st epoch which indicates

[6580/6580 1:17:32, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.905200	0.579663	0.861689	0.837905	0.861689	0.837843
2	0.392600	0.230989	0.942719	0.945024	0.942719	0.942779
3	0.283100	0.175634	0.950983	0.952365	0.950983	0.950995
4	0.195800	0.158704	0.953643	0.954832	0.953643	0.953653
5	0.169200	0.147594	0.955163	0.956402	0.955163	0.955198
6	0.157300	0.142063	0.955448	0.956716	0.955448	0.955492
7	0.103100	0.138115	0.955733	0.956806	0.955733	0.955751
8	0.159300	0.136721	0.956778	0.957846	0.956778	0.956764
9	0.179000	0.135892	0.957063	0.958013	0.957063	0.957049
10	0.147100	0.135319	0.956873	0.957795	0.956873	0.956855

Figure 4.2: Table describing trainingloss,validation loss,accuracy, precision

that the model has adapted quickly to training data.Around epoch 2 the model begins to converge, which suggests that the model is learning effectively and is generalizing well to the validation data. The curves nearly flat after 6 epochs, which indicates that models performance does not improve significantly with further training. **Observations**

- Both loss curves decrease over time in few epochs, especially training loss, which is generally desirable during training.
- The convergence of *Validation Loss* toward *Training Loss* suggests that the model is learning effectively.

The figure above provides a comprehensive performance evaluation of a classification model.

- **Classification Metrics Table:**
  - Precision, recall, and F1-score are listed for each class (labeled 0 to 12).
  - Crptojacker,worm, have precision, recall,f1-score of 1.0,fakeapp have a precision of 1.0, followed by recall of 1.0 , keylogger have a recall of 1.0 .
  - The “weighted avg” class has a precision of 0.96, recall of 0.95 and

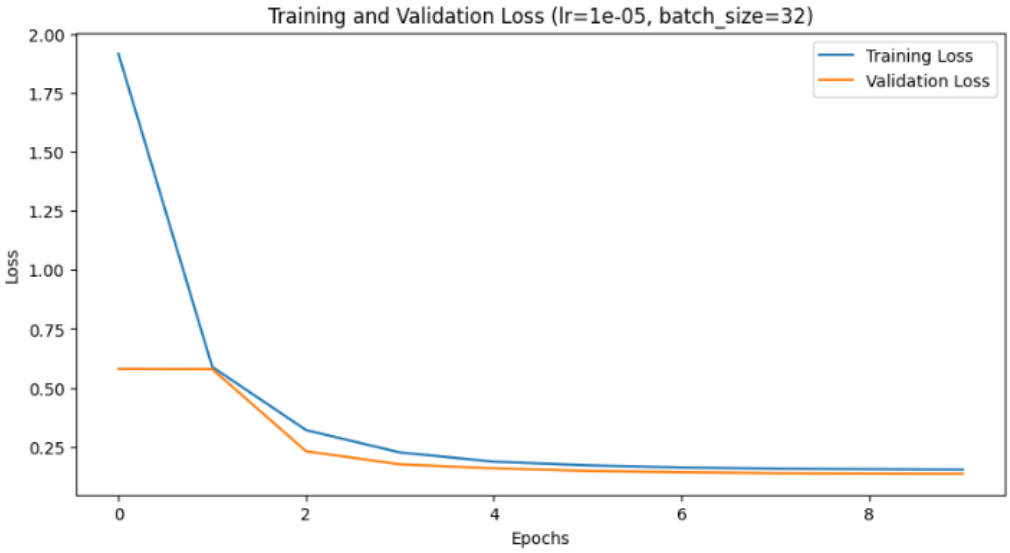


Figure 4.3: Line Plot describing training/validation plot

f1-score of 0.95.

* The "macro avg" class has an precision of 0.96, recall of 0.95 and f1-score of 0.95.
* The "support" values indicate the number of instances for each class.

– **Confusion Matrix Heatmap:**

* The heatmap represents true labels (y-axis) versus predicted labels (x-axis).
* High values along the diagonal indicate correct predictions.
* Off-diagonal cells show misclassifications.

Overall, this model has performed well, with high accuracy and minimal misclassifications.

– **ROC Curve:** The graph depicts a multi-class Receiver Operating Characteristic (ROC) curve, showcasing the Area Under the Curve (AUC) values for each class. Here's what we can infer:

– **X-Axis (False Positive Rate):** The x-axis represents the False Positive Rate (FPR), ranging from 0 to 1. It indicates the proportion of false positives (incorrectly predicted positive instances) relative to all actual negatives.

– **Y-Axis (True Positive Rate):** The y-axis represents the True Positive Rate (TPR), also ranging from 0 to 1. It signifies the proportion of true

Classification Report:

	precision	recall	f1-score	support
scareware	0.99	0.93	0.96	1000
ransomware	0.91	0.97	0.94	1635
adware	0.92	0.96	0.94	1365
smsmalware	0.95	0.94	0.95	1047
trojan	0.97	0.95	0.96	1377
benign	0.91	0.94	0.92	1062
spyware	0.95	0.94	0.94	1077
polymorphic	0.97	0.78	0.86	247
downloader	0.96	0.95	0.96	1260
cryptojacker	1.00	1.00	1.00	1000
worm	1.00	1.00	1.00	1269
fake app	1.00	0.96	0.98	356
keylogger	0.99	1.00	0.99	463
accuracy			0.96	13158
macro avg	0.96	0.95	0.95	13158
weighted avg	0.96	0.96	0.96	13158

Figure 4.4: Confusion Matrix and Classification Report

positives (correctly predicted positive instances) relative to all actual positives.

- **Individual ROC Curves:** Each colored line corresponds to a specific class. These curves illustrate how well the model distinguishes between that class and the rest. The closer the curve is to the top-left corner (True Positive Rate = 1, False Positive Rate = 0), the better the model's performance for that class.
- **AUC Values:** The legend lists different classes along with their corresponding AUC values. AUC quantifies the overall performance of the classifier. High AUC values (ranging from 0.97 to 1.00) indicate excellent classification ability.
- **Macro-average ROC Curve:** The highlighted curve represents the macro-average across all classes. Its AUC value (0.97) summarizes the overall model performance.
- **Model Performance:**

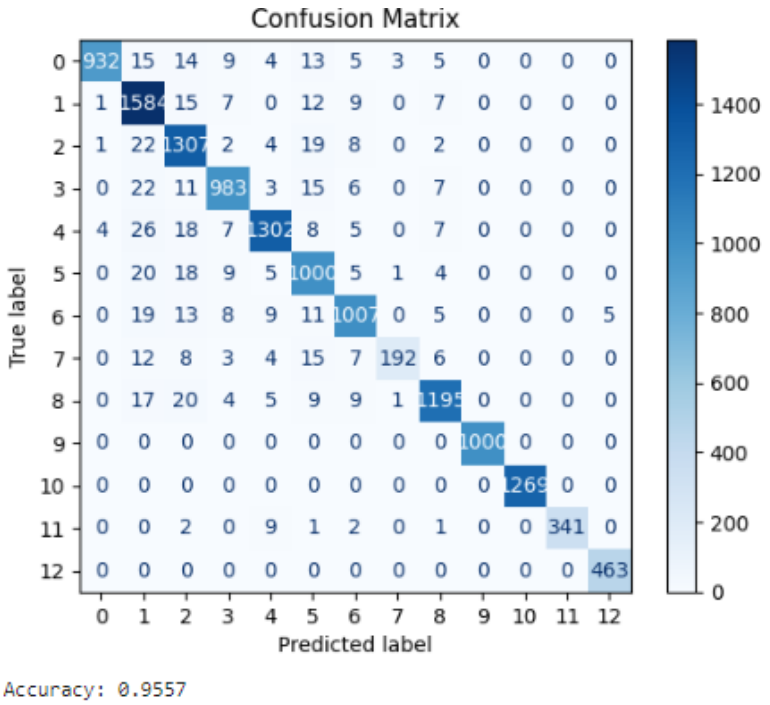


Figure 4.5: Confusion Matrix and Classification Report

- The model performs exceptionally well across all classes, as evidenced by the almost perfect AUC scores.
- This suggests that the model perfectly distinguishes between different classes without any misclassification.

Overall, this ROC curve demonstrates good classification performance. The table describes columns representing different performance metrics across four epochs. Each row corresponds to the epoch number and its associated training loss, validation loss, accuracy, precision, recall, and F1 score, providing a structured way to view the progression of the machine learning model's training performance. It can be observed that there is a slight increase in training loss in 8th and 9th epoch, which might indicative bit of overfitting, but overall the performance is good.

Behavior of Training and Validation Loss Functions

Here, the x-axis represents the epochs starting from 0 to 14,i.e 15 epochs in total. There is a rapid drop in training loss after 1st epoch which indicates that the model has adapted quickly to training data.Around epoch 2 the model begins to converge, which suggests that the model is learning effectively

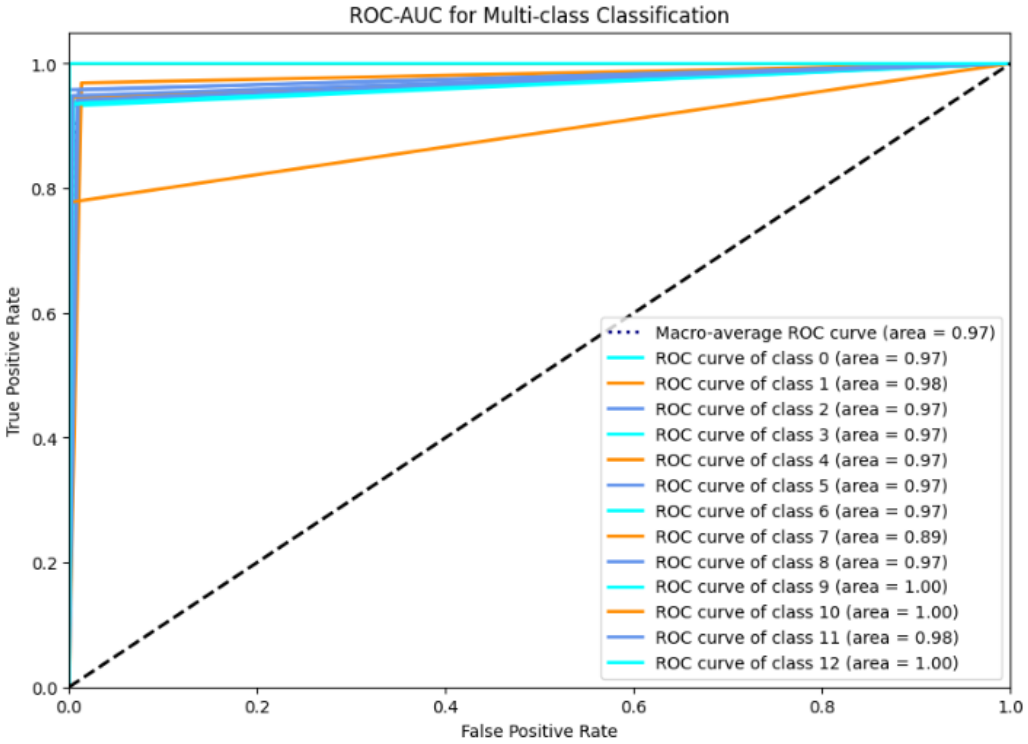


Figure 4.6: Plot for Confusion Matrix and ROC Curve
The figure above provides a Receiver Operating Characteristic (ROC) curve for a multi-class classification model.

and is generalizing well to the validation data. The curves nearly flat after 6 epochs, which indicates that models performance does not improve significantly with further training. **Observations**

- Both loss curves decrease over time in few epochs, especially training loss, which is generally desirable during training.
- The convergence of *Validation Loss* toward *Training Loss* suggests that the model is learning effectively.

The figure above provides a comprehensive performance evaluation of a classification model.

- **Classification Metrics Table:**
  - Precision, recall, and F1-score are listed for each class (labeled 0 to 12).
  - Crptojacker,worm, have precision, recall,f1-score of 1.0,fakeapp have a precision of 1.0, followed by recall of 1.0 , keylogger have a

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	1.660700	1.298315	0.765157	0.717394	0.765157	0.714884
2	0.694100	0.400591	0.906574	0.903809	0.906574	0.903211
3	0.446300	0.285729	0.914849	0.909724	0.914849	0.911580
4	0.403700	0.249736	0.920867	0.915783	0.920867	0.917656
5	0.267400	0.231484	0.930495	0.925657	0.930495	0.927428
6	0.258900	0.222350	0.930946	0.925851	0.930946	0.927795
7	0.259700	0.215159	0.930645	0.924804	0.930645	0.927240
8	0.271100	0.209107	0.931398	0.926000	0.931398	0.928147
9	0.218800	0.207782	0.932150	0.927134	0.932150	0.928894
10	0.242200	0.204563	0.932601	0.926886	0.932601	0.929253
11	0.238500	0.201759	0.932902	0.927344	0.932902	0.929635
12	0.260500	0.201864	0.932902	0.927428	0.932902	0.929561
13	0.237600	0.200394	0.933053	0.927488	0.933053	0.929719
14	0.191900	0.200505	0.933053	0.927627	0.933053	0.929775
15	0.194100	0.200191	0.933203	0.927876	0.933203	0.929928

Figure 4.7: Table describing trainingloss,validation loss,accuracy, precision

recall of 1.0, polymorphic has a low recall and f1-score of 0.78 and 0.87 respectively .

* The “weighted avg” class has a precision of 0.96, recall of 0.95 and f1-score of 0.95.
* The “macro avg” class has an precision of 0.96, recall of 0.95 and f1-score of 0.95.
* The “support” values indicate the number of instances for each class.

– **Confusion Matrix Heatmap:**

* The heatmap represents true labels (y-axis) versus predicted labels (x-axis).
* High values along the diagonal indicate correct predictions.
* Off-diagonal cells show misclassifications. Classes 0 to 8 have many off diagonal misclassifications.

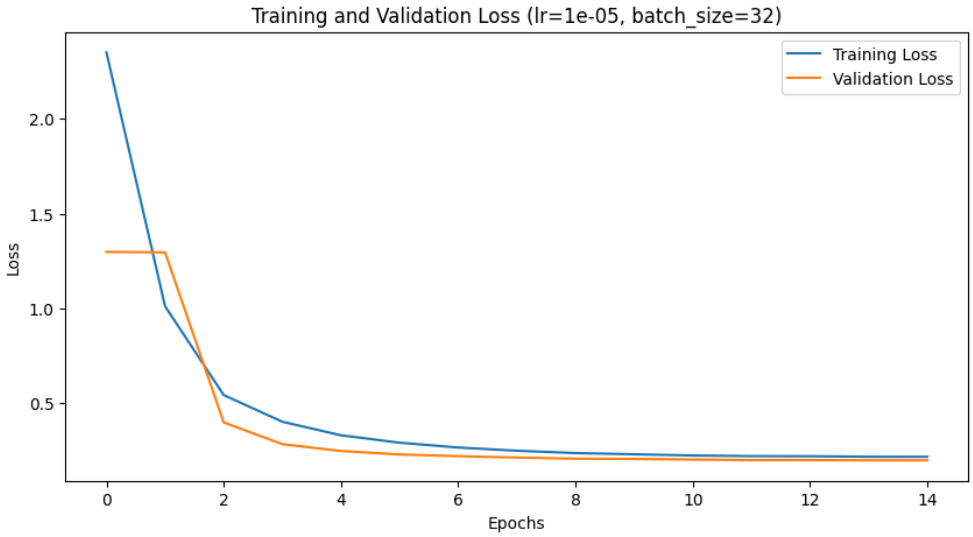


Figure 4.8: Line Plot describing training/validation plot

Overall, this model has performed well, with high accuracy and minimal misclassifications.

- **ROC Curve:** The graph depicts a multi-class Receiver Operating Characteristic (ROC) curve, showcasing the Area Under the Curve (AUC) values for each class. Here's what we can infer:
- **X-Axis (False Positive Rate):** The x-axis represents the False Positive Rate (FPR), ranging from 0 to 1. It indicates the proportion of false positives (incorrectly predicted positive instances) relative to all actual negatives.
- **Y-Axis (True Positive Rate):** The y-axis represents the True Positive Rate (TPR), also ranging from 0 to 1. It signifies the proportion of true positives (correctly predicted positive instances) relative to all actual positives.
- **Individual ROC Curves:** Each colored line corresponds to a specific class. These curves illustrate how well the model distinguishes between that class and the rest. The closer the curve is to the top-left corner (True Positive Rate = 1, False Positive Rate = 0), the better the model's performance for that class.
- **AUC Values:** The legend lists different classes along with their cor-

Classification Report:

	precision	recall	f1-score	support
scareware	0.91	0.95	0.93	1000
ransomware	0.95	0.96	0.95	1635
adware	0.91	0.96	0.93	1365
smsmalware	0.94	0.94	0.94	1047
trojan	0.97	0.95	0.96	1377
benign	0.90	0.94	0.92	1062
spyware	0.97	0.94	0.95	1077
polymorphic	0.98	0.78	0.87	247
downloader	1.00	0.95	0.97	1260
cryptojacker	1.00	1.00	1.00	1000
worm	1.00	1.00	1.00	1269
fake app	1.00	1.00	1.00	356
keylogger	1.00	1.00	1.00	463
accuracy			0.96	13158
macro avg	0.96	0.95	0.96	13158
weighted avg	0.96	0.96	0.96	13158

Figure 4.9: Confusion Matrix and Classification Report

responding AUC values. AUC quantifies the overall performance of the classifier. High AUC values (ranging from 0.97 to 1.00) indicate excellent classification ability. Here, from the ROC curve, it can be observed that class 7 which is polymorphic is having lowest value i.e 0.89, class 9, 10,12 is having highest ROC value which is 1.00.

- **Macro-average ROC Curve:** The highlighted curve represents the macro-average across all classes. Its AUC value (0.97) summarizes the overall model performance.
- **Model Performance:**
  - The model performs exceptionally well across all classes, as evidenced by the almost perfect AUC scores.
  - This suggests that the model perfectly distinguishes between different classes without any misclassification.

Overall, this ROC curve demonstrates good classification performance.

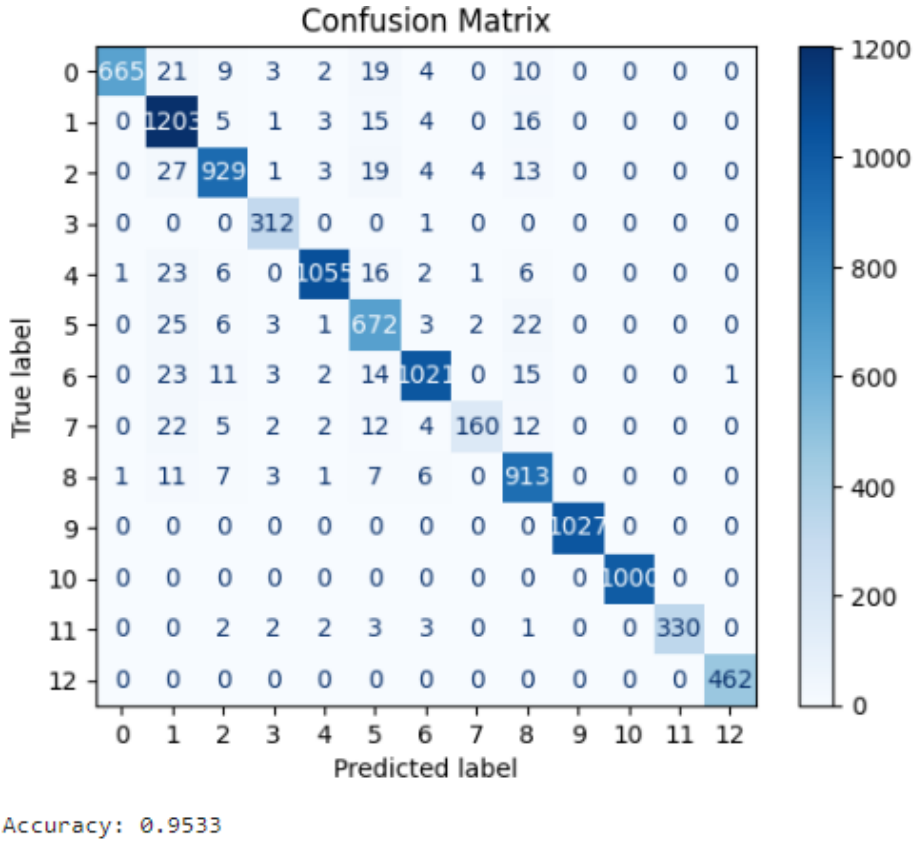


Figure 4.10: Confusion Matrix and Classification Report

4.3.4 Best Case Inference Result

All the input messages describe behavior typical of a computer worm (e.g., spreading across networks, causing multiple infections, exploiting vulnerabilities). The consistent label ”worm” suggests the model has successfully identified these descriptions as indicative of worm behavior.

4.3.5 Worst Case Results

SecureBERT:(Worst Case Scenario)

In the worst case scenario of the model, we selected:

- `learningrate` = 5e-5
- `epoch` = 10,15
- `batchsize` = 32

The table describes columns representing different performance metrics across 10 epochs. Each row corresponds to the epoch number and its associated training loss, validation loss, accuracy, precision, recall, and F1 score,

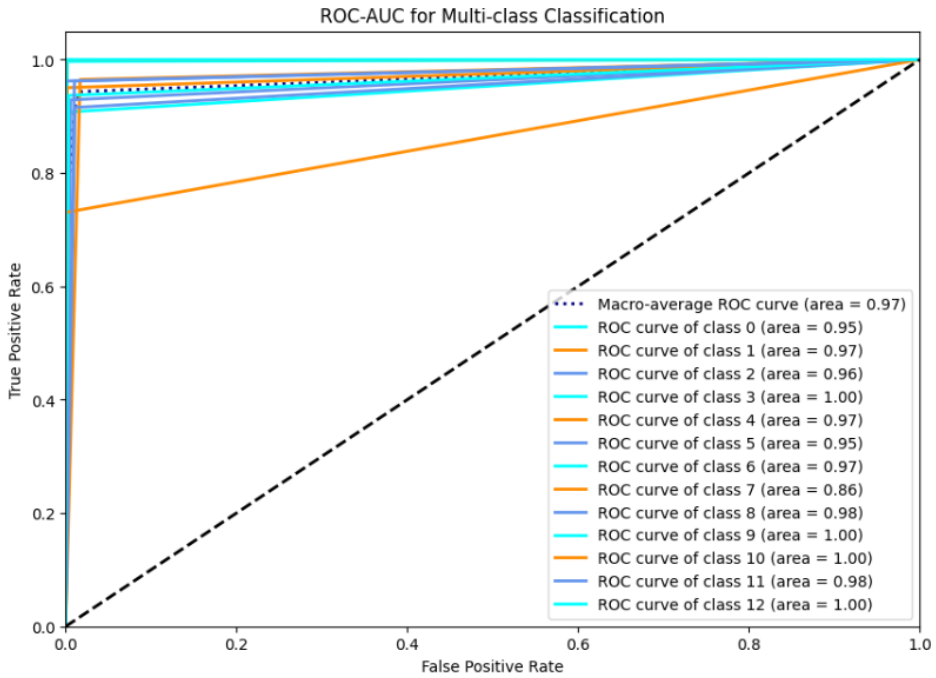


Figure 4.11: Plot for Confusion Matrix and ROC Curve
The figure above provides a Receiver Operating Characteristic (ROC) curve for a multi-class classification model.

providing a structured way to view the progression of the machine learning model's training performance. It can be observed that there is a slight increase in training loss in 7th,8th,10th,12th epoch, which might indicative bit of overfitting, but overall the performance is good.

Behavior of Training and Validation Loss Functions

The training loss increases initially and then slightly decreases, which may indicate a complex learning pattern where the model is initially trying to overfit the training data but later stabilizes. The validation loss consistently decreases, suggesting that the model is improving its generalization to unseen data over the epochs. The higher learning rate (5e-05) compared to the previous plot (1e-05) might have led to quicker adjustments in the model, but it still shows a trend of stabilization without drastic loss reduction. **Implications**:

The consistent decrease in validation loss is encouraging, as it shows the model is learning to generalize better. The pattern of the training loss suggests that with more epochs, the model might further stabilize and potentially reduce both training and validation losses. The current learning rate of 3e-05

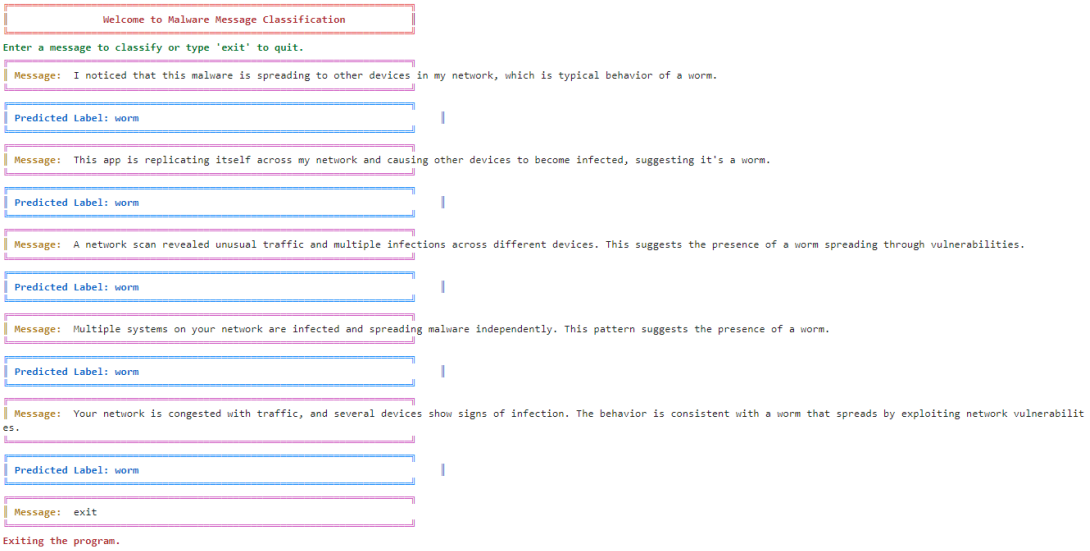


Figure 4.12: Best case result of class 10 i.e 'Worm'

seems to be a reasonable choice, though further tuning (e.g., testing slightly higher or lower values) could help find an optimal setting for faster convergence. Overall, this plot indicates that the model is training correctly, with validation loss steadily improving, though more epochs might be needed for the loss to fully converge.

Observations

The overall decrease in training loss is a good sign that the model is learning and fitting the training data better. The slight increase in validation loss suggests that the model might be starting to overfit

The figure above provides a comprehensive performance evaluation of a classification model.

- **Classification Metrics Table:**
  - Precision, recall, and F1-score are listed for each class (labeled 0 to 12).
  - Crptojacker,worm, have precision, recall,f1-score of 1.0,fakeapp have a precision of 1.0, followed by recall of 1.0 , keylogger have a recall of 1.0 .
  - The "weighted avg" class has a precision of 0.96, recall of 0.95 and f1-score of 0.95.
  - The "macro avg" class has an precision of 0.96, recall of 0.95 and

[6580/6580 1:18:50, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.175000	0.137151	0.955828	0.957613	0.955828	0.955947
2	0.121200	0.127114	0.957443	0.959599	0.957443	0.957697
3	0.126000	0.116548	0.957633	0.959593	0.957633	0.957762
4	0.105900	0.115021	0.957823	0.958787	0.957823	0.957798
5	0.108900	0.113760	0.958108	0.959883	0.958108	0.958233
6	0.102300	0.114265	0.956778	0.960839	0.956778	0.957510
7	0.079000	0.113232	0.957538	0.960192	0.957538	0.957861
8	0.133500	0.113113	0.957348	0.959349	0.957348	0.957517
9	0.151400	0.113445	0.956873	0.957507	0.956873	0.956752
10	0.113800	0.112852	0.957158	0.957756	0.957158	0.957074

Figure 4.13: Table describing trainingloss,validation loss,accuracy, precision

f1-score of 0.95.

* The "support" values indicate the number of instances for each class.

– **Confusion Matrix Heatmap:**

* The heatmap represents true labels (y-axis) versus predicted labels (x-axis).
* High values along the diagonal indicate correct predictions.
* Off-diagonal cells show misclassifications.

Overall, this model has performed well, with high accuracy and minimal misclassifications.

– **ROC Curve:** The graph depicts a multi-class Receiver Operating Characteristic (ROC) curve, showcasing the Area Under the Curve (AUC) values for each class. Here's what we can infer:

– **X-Axis (False Positive Rate):** The x-axis represents the False Positive Rate (FPR), ranging from 0 to 1. It indicates the proportion of false positives (incorrectly predicted positive instances) relative to all actual negatives.

– **Y-Axis (True Positive Rate):** The y-axis represents the True Positive Rate (TPR), also ranging from 0 to 1. It signifies the proportion of true

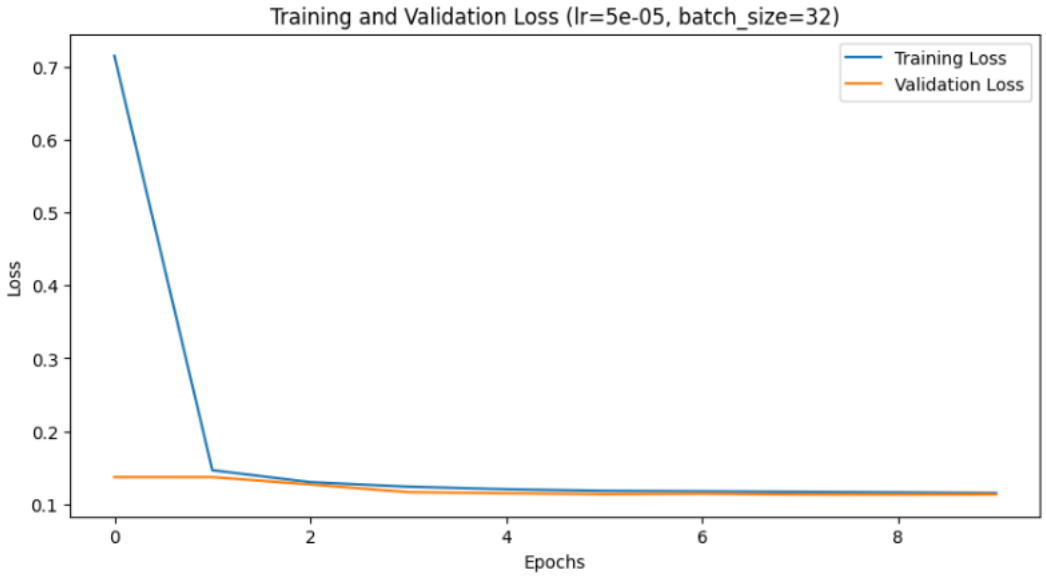


Figure 4.14: Line Plot describing training/validation plot

positives (correctly predicted positive instances) relative to all actual positives.

- **Individual ROC Curves:** Each colored line corresponds to a specific class. These curves illustrate how well the model distinguishes between that class and the rest. The closer the curve is to the top-left corner (True Positive Rate = 1, False Positive Rate = 0), the better the model's performance for that class.
- **AUC Values:** The legend lists different classes along with their corresponding AUC values. AUC quantifies the overall performance of the classifier. High AUC values (ranging from 0.97 to 1.00) indicate excellent classification ability.Here, from the curve, it can be observed that class 7 which is polymorphic is having lowest AUC value i.e 0.89, class 9, 10,12 is having highest ROC value which is 1.00.
- **Macro-average ROC Curve:** The highlighted curve represents the macro-average across all classes. Its AUC value (0.97) summarizes the overall model performance.
- **Model Performance:**
  - The model performs exceptionally well across all classes, as evidenced by the almost perfect AUC scores.
  - This suggests that the model perfectly distinguishes between differ-

Classification Report:

	precision	recall	f1-score	support
scareware	0.91	0.95	0.93	1000
ransomware	0.95	0.96	0.96	1635
adware	0.93	0.96	0.95	1365
smsmalware	0.94	0.94	0.94	1047
trojan	0.97	0.95	0.96	1377
benign	0.90	0.94	0.92	1062
spyware	0.95	0.94	0.95	1077
polymorphic	0.98	0.78	0.87	247
downloader	0.99	0.95	0.97	1260
cryptojacker	1.00	1.00	1.00	1000
worm	1.00	1.00	1.00	1269
fake app	1.00	1.00	1.00	356
keylogger	1.00	1.00	1.00	463
accuracy			0.96	13158
macro avg	0.96	0.95	0.96	13158
weighted avg	0.96	0.96	0.96	13158

Figure 4.15: Confusion Matrix and Classification Report

ent classes without any misclassification.

Overall, this ROC curve demonstrates good classification performance.

The table describes columns representing different performance metrics across fifteen epochs. Each row corresponds to the epoch number and its associated training loss, validation loss, accuracy, precision, recall, and F1 score, providing a structured way to view the progression of the machine learning model's training performance. It can be observed that there is a slight increase in training loss in 8th,9th,11th,14th epoch, which might indicative bit of overfitting, and slight increase in validation loss in 12th epoch but overall the performance is good.

Behavior of Training and Validation Loss Functions

The training loss increases initially and then slightly decreases, which may indicate a complex learning pattern where the model is initially trying to overfit the training data but later stabilizes. The validation loss consistently decreases, suggesting that the model is improving its generalization to unseen

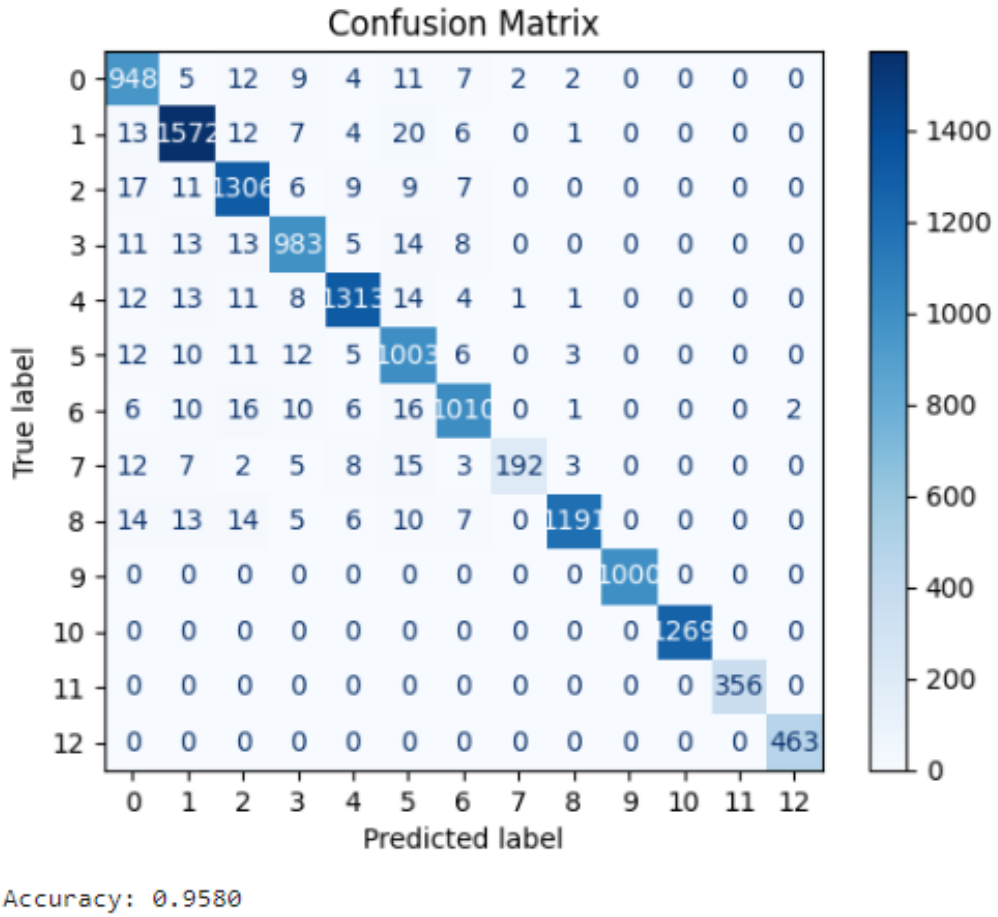


Figure 4.16: Confusion Matrix and Classification Report

data over the epochs. The slightly higher learning rate (5e-05) compared to the previous plot (1e-05) might have led to quicker adjustments in the model, but it still shows a trend of stabilization without drastic loss reduction, but lower learning rate generally performs better. **Implications**:

The consistent decrease in validation loss is encouraging, as it shows the model is learning to generalize better. The pattern of the training loss suggests that with more epochs, the model might further stabilize and potentially reduce both training and validation losses. The current learning rate of 5e-05 seems to be a reasonable choice, though further tuning (e.g., testing slightly higher or lower values) could help find an optimal setting for faster convergence. Overall, this plot indicates that the model is training correctly, with validation loss steadily improving, though more epochs might be needed for the loss to fully converge.

Observations

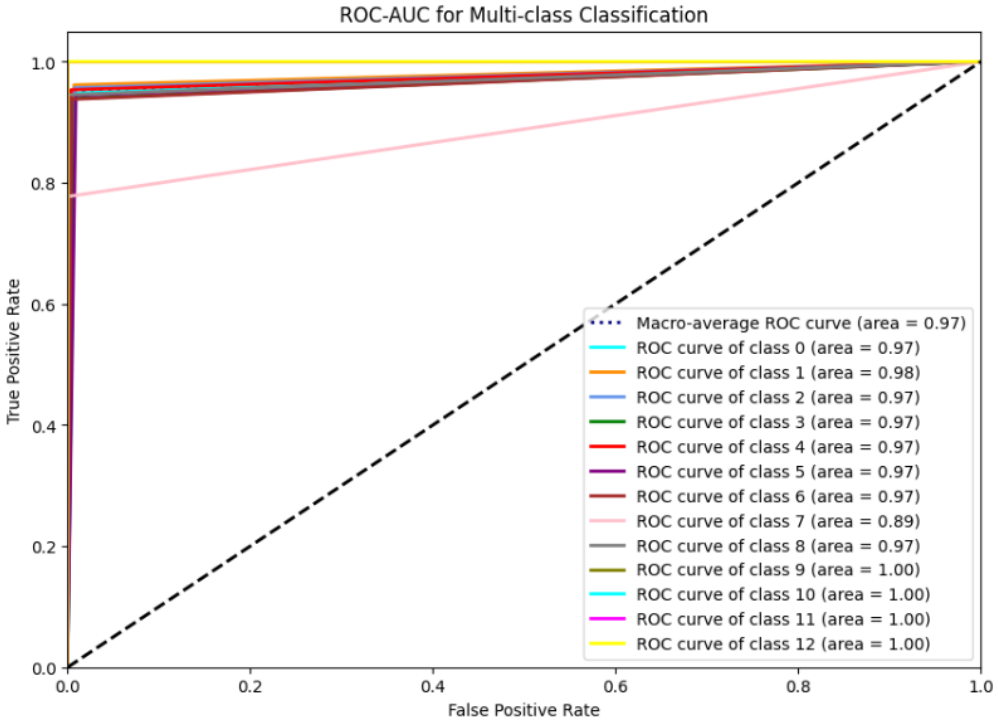


Figure 4.17: Plot for Confusion Matrix and ROC Curve
The figure above provides a Receiver Operating Characteristic (ROC) curve for a multi-class classification model.

The overall decrease in training loss is a good sign that the model is learning and fitting the training data better. The slight increase in validation loss suggests that the model might be starting to overfi.

The figure above provides a comprehensive performance evaluation of a classification model.

- **Classification Metrics Table:**
  - Precision, recall, and F1-score are listed for each class (labeled 0 to 12).
  - Crptojacker,worm, have precision, recall,f1-score of 1.0,fakeapp have a precision of 1.0, followed by recall of 1.0 , keylogger have a recall of 1.0 . Polymorphic class has a recall of 0.78 and f1-score of 0.87 which is lowest in terms of all classes.
  - The “weighted avg” class has a precision of 0.96, recall of 0.95 and f1-score of 0.95.
  - The “macro avg” class has an precision of 0.96, recall of 0.95 and f1-score of 0.95.

[9870/9870 1:58:54, Epoch 15/15]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.173400	0.136197	0.956018	0.957808	0.956018	0.956157
2	0.122700	0.125977	0.956968	0.959360	0.956968	0.957293
3	0.127400	0.116459	0.957728	0.960298	0.957728	0.957984
4	0.105900	0.115382	0.956398	0.957793	0.956398	0.956455
5	0.110300	0.114094	0.957823	0.959623	0.957823	0.957943
6	0.103400	0.114455	0.958298	0.963167	0.958298	0.959146
7	0.080300	0.112811	0.957253	0.957683	0.957253	0.957091
8	0.134300	0.112662	0.957633	0.959190	0.957633	0.957680
9	0.149000	0.110726	0.957443	0.958582	0.957443	0.957423
10	0.117700	0.110265	0.956493	0.957325	0.956493	0.956531
11	0.125600	0.110242	0.958298	0.959743	0.958298	0.958442
12	0.115400	0.110577	0.956873	0.958036	0.956873	0.956968
13	0.096900	0.110575	0.956778	0.957882	0.956778	0.956800
14	0.131200	0.110187	0.956683	0.957426	0.956683	0.956627
15	0.111600	0.110167	0.956398	0.957118	0.956398	0.956353

Figure 4.18: Table describing trainingloss,validation loss,accuracy, precision

    - The "support" values indicate the number of instances for each class.

- **Confusion Matrix Heatmap:**
- **Confusion Matrix Heatmap:**
    - The heatmap represents true labels (y-axis) versus predicted labels (x-axis).
    - High values along the diagonal indicate correct predictions.
    - Off-diagonal cells show misclassifications. Classes 0-8 have many off-diagonal cells.

Overall, this model has performed well, with high accuracy and minimal misclassifications.

- **ROC Curve:** The graph depicts a multi-class Receiver Operating Characteristic (ROC) curve, showcasing the Area Under the Curve (AUC) values for each class. Here's what we can infer:
- **X-Axis (False Positive Rate):** The x-axis represents the False Positive

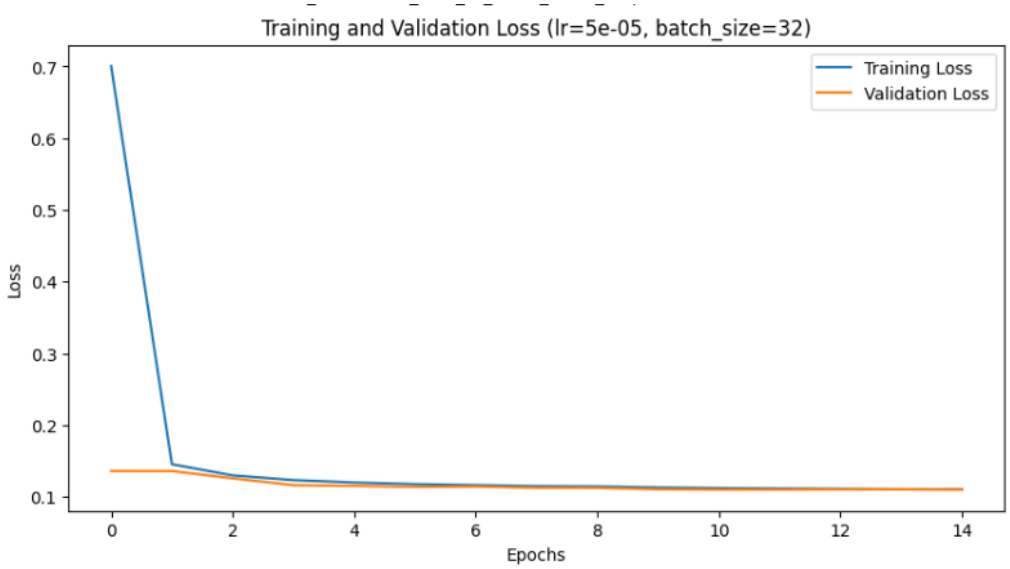


Figure 4.19: Line Plot describing training/validation plot

Rate (FPR), ranging from 0 to 1. It indicates the proportion of false positives (incorrectly predicted positive instances) relative to all actual negatives.

- **Y-Axis (True Positive Rate):** The y-axis represents the True Positive Rate (TPR), also ranging from 0 to 1. It signifies the proportion of true positives (correctly predicted positive instances) relative to all actual positives.
- **Individual ROC Curves:** Each colored line corresponds to a specific class. These curves illustrate how well the model distinguishes between that class and the rest. The closer the curve is to the top-left corner (True Positive Rate = 1, False Positive Rate = 0), the better the model's performance for that class.
- **AUC Values:** The legend lists different classes along with their corresponding AUC values. AUC quantifies the overall performance of the classifier. High AUC values (ranging from 0.97 to 1.00) indicate excellent classification ability. Here, from the curve, it can be observed that class 7 which is polymorphic is having lowest AUC value i.e 0.89, class 9, 10,12 is having highest AUC value which is 1.00.
- **Macro-average ROC Curve:** The highlighted curve represents the macro-average across all classes. Its AUC value (0.97) summarizes the

Classification Report:

	precision	recall	f1-score	support
scareware	0.90	0.95	0.92	1000
ransomware	0.95	0.96	0.95	1635
adware	0.92	0.96	0.94	1365
smsmalware	0.95	0.94	0.94	1047
trojan	0.97	0.95	0.96	1377
benign	0.92	0.94	0.93	1062
spyware	0.93	0.94	0.94	1077
polymorphic	0.98	0.78	0.87	247
downloader	1.00	0.95	0.97	1260
cryptojacker	1.00	1.00	1.00	1000
worm	1.00	1.00	1.00	1269
fake app	1.00	1.00	1.00	356
keylogger	1.00	1.00	1.00	463
accuracy			0.96	13158
macro avg	0.96	0.95	0.96	13158
weighted avg	0.96	0.96	0.96	13158

Figure 4.20: Confusion Matrix and Classification Report

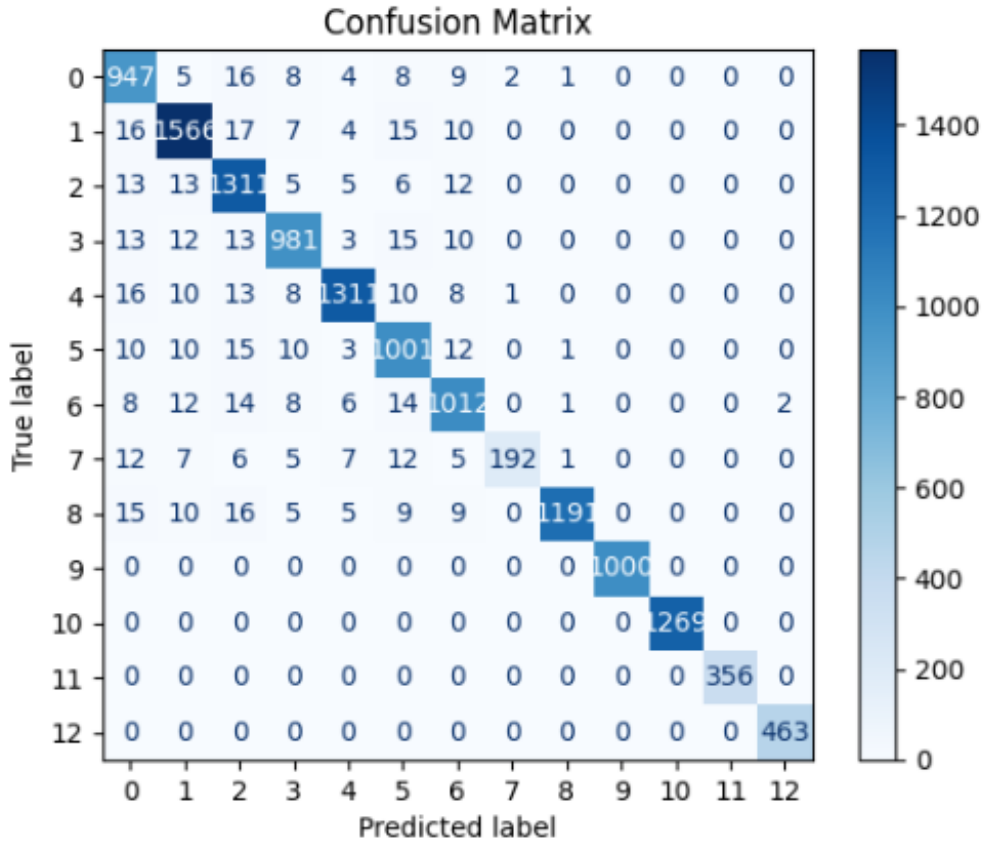
overall model performance.

- **Model Performance:**
  - The model performs exceptionally well across all classes, as evidenced by the almost perfect AUC scores.
  - This suggests that the model perfectly distinguishes between different classes without any misclassification.

Overall, this ROC curve demonstrates good classification performance.

4.3.6 Worst Case Inference Result

The image displays a text classification output where 3 out of 5 messages are correctly classified as ”polymorphic” based on their descriptions of self-modifying and encryption-evading malware. However, the first message, labeled as ”ransomware,” and the fifth message, labeled as ”smsmalware,” are misclassified, indicating errors in the model’s predictions for this specific case.




```
Accuracy: 0.9576
```

Figure 4.21: Confusion Matrix and Classification Report

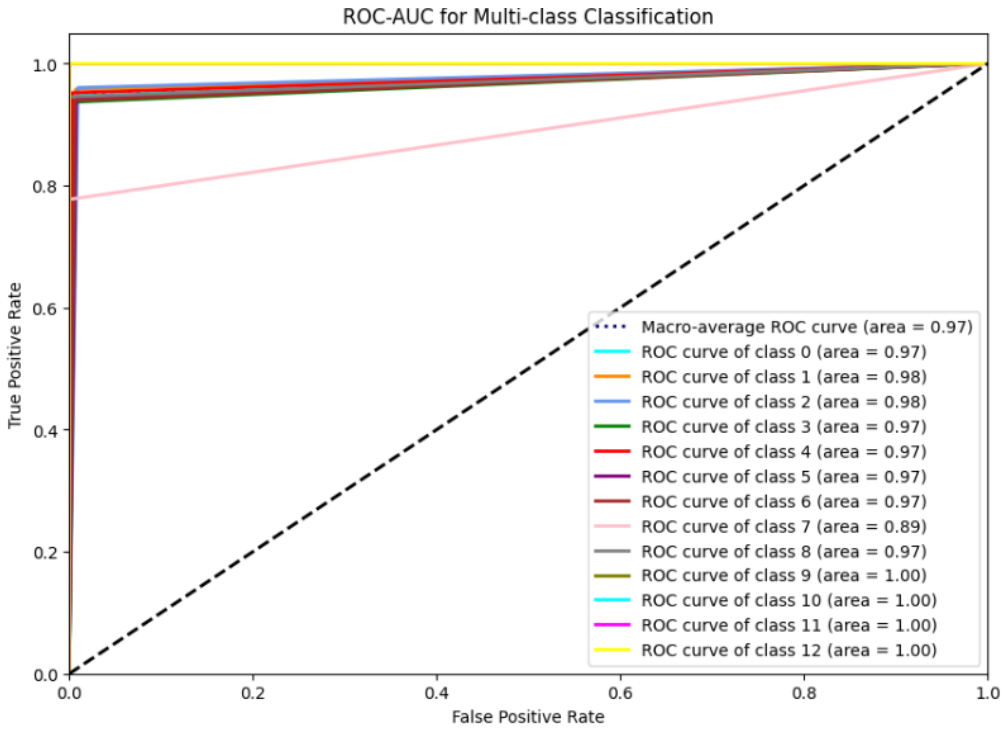


Figure 4.22: Plot for Confusion Matrix and ROC Curve

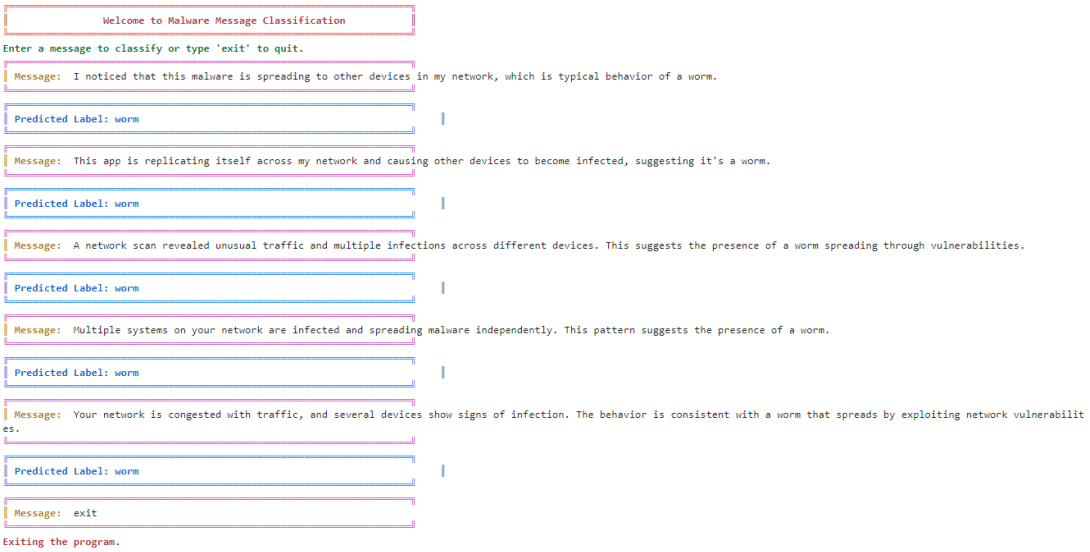


Figure 4.23: Worst case result of class 7 i.e 'Polymorphic'

5 DISCUSSION AND ANALYSIS

5.1 Comparison of Theoretical and Simulated Outputs

While comparing theoretical and simulated outputs we the differences between what was expected based on theoretical models and what was observed in practice. In this case, the theoretical output is an accuracy of 0.95-1, while the simulated output is 0.9557 and 0.9316 in best case and 0.9577 and 0.9574 accuracy in worst case . The reasons for discrepancies are as follows:

1. **Collect Data**:

   - *Theoretical Output*: 0-95-1 accuracy.
   - *Simulated Output*: 0.9557 (learning rate : 1e-5,epoch 10) and 0.9533 (learning rate:1e-5,epoch 15) in best case.
   - *Simulated Output*: 0.9580 (learning rate:5e-5, epoch 10) and 0.9576 (learning rate: 5e-5, epoch 15) accuracy in worst case (SecureBERT).

2. **Identify Differences**:

   - Theoretrical output and simulated output are similar for both models.

5.1.1 Potential Reasons for Discrepancies

Model Complexity,Underfitting and overfitting

- **Overfitting**: Overfitting occurs when the model learns the noise and peculiarities in the training data, which may not generalize well to unseen data. This phenomenon can result in misleadingly high simulated accuracies, as the model may memorize the training set rather than learning generalizable patterns. To mitigate overfitting, techniques such as regularization, dropout, and cross-validation are commonly used. Ensuring that the model is not too complex relative to the amount of training data can also help in reducing overfitting.
- **Model Architecture**: Differences in model architecture between the theoretical design and the actual implementation can impact model

performance. Complex models, such as SecureBERT, often achieve higher accuracies due to their deeper architectures and pre-trained nature. The architecture's depth, number of layers, and the type of layers used can all influence how well the model captures relevant features from the data and generalizes to new examples. Ensuring that the model architecture is well-suited to the problem domain and data characteristics is crucial for optimal performance.

- **Underfitting**: Underfitting occurs when the model is too simple to capture the underlying patterns in the training data. This can result in poor performance on both the training and test datasets. Underfitting can be caused by a model that is too shallow, uses inadequate features, or lacks sufficient complexity to represent the data effectively. To address underfitting, one might consider increasing the model's complexity, using more features, or training the model for a longer duration. Additionally, ensuring that the model's assumptions align with the data's characteristics can help in improving performance.

Data Quality and Distribution

- **Data Distribution**: The distribution of the training and test datasets used in simulations might differ from theoretical assumptions. Variations such as noise, outliers, and imbalanced class distributions in textual data can affect model performance. For instance, if certain classes are underrepresented, the model might struggle to learn their characteristics effectively. Proper data preprocessing, including normalization and balancing of classes, can help address these issues and improve the model's performance.
- **Data Quality**: The quality of the data significantly impacts model performance. Factors such as mislabeled data, incomplete entries, and inconsistencies can adversely affect the learning process. Clean, well-annotated data usually leads to better model performance and more reliable results. Implementing robust data cleaning and validation processes is essential for ensuring that the data used for training and evaluation is of high quality.

Training Process and Hyperparameters

- **Hyperparameter Tuning**: Differences in hyperparameter tuning (learning rate, batch size, epochs, etc.) between theoretical and simulated environments can lead to different outcomes. Optimal hyperparameters can significantly boost model performance.
- **Training Duration**: The duration and number of epochs for which the models were trained could vary, affecting the final accuracy.

Evaluation Metrics and Techniques

- **Evaluation Metrics**: Theoretical output might be based on simplified or different evaluation metrics compared to the simulated output, which might use more comprehensive or stricter evaluation criteria.
- **Validation Techniques**: Differences in validation techniques (e.g., cross-validation, train-test split ratios) can result in varied performance metrics.

Randomness and Initialization

- **Random Seed**: Variations in random seed initialization can lead to different weight initializations and training paths, which might impact the accuracy of the model.
- **Stochastic Nature of Training**: The stochastic nature of training neural networks, including random shuffling of data and dropout layers, can result in different outcomes even with the same model and data.

Implementation Details

- **Library and Framework Differences**: Differences in the libraries and frameworks used for implementation might lead to subtle variations in model behavior and performance.
- **Code Optimizations**: Optimizations or differences in the code implementation can also result in performance discrepancies.

5.1.2 Steps for Error Analysis

1. **Data Quality and Preprocessing**:
   - *Incorrect Labels*:

    * Review and verify the labels in the dataset to ensure accuracy. Mislabeled data points can mislead the model, causing it to learn incorrect patterns and leading to poor generalization on unseen data. Implementing a robust labeling process and cross-checking labels with domain experts can help mitigate this issue.
  - *Missing Values*:
    * Identify and address any missing values in the dataset. Common strategies include imputing missing values using statistical measures such as mean, median, or mode, or removing rows with missing data. Ensuring that missing values are handled appropriately is crucial to maintain data integrity and prevent biases in model training.
  - *Noise*:
    * Evaluate the dataset for noisy data that might need to be cleaned or removed. Noise can come from irrelevant or erroneous information and can obscure the true patterns the model needs to learn. Techniques such as data cleaning, outlier detection, and noise reduction algorithms can help in mitigating noise-related issues.
2. **Model Training and Evaluation**:
  - *Insufficient Training*:
    * Ensure that the model has been trained for a sufficient number of epochs to learn the underlying patterns in the data. Undertraining can result in a model that does not perform well. Monitoring training curves and validating the model on a separate dataset can help ensure that the model is adequately trained.
  - *Overfitting/Underfitting*:
    * Check for signs of overfitting (where the model performs well on training data but poorly on validation data) or underfitting (where the model performs poorly on both training and validation data). Techniques such as regularization, cross-validation, and early stopping can be used to address these issues effectively.

- *Hyperparameter Tuning*:
  - Optimize the model's hyperparameters (such as learning rate, batch size, and number of layers) to achieve the best performance. Hyperparameter tuning can significantly impact the model's effectiveness and overall performance. Utilize techniques such as grid search or random search to find the optimal set of hyperparameters.

3. *Regularization Techniques*:
   - **L1 Regularization (Lasso)**: Add a penalty equal to the absolute value of the magnitude of coefficients to the loss function:

$$\text{Loss}_{L1} = \text{Loss}_{\text{original}} + \lambda \sum_i |w_i| \tag{5.1}$$

     where λ is the regularization parameter, and w_i represents the model coefficients. This technique promotes sparsity and can help with feature selection.
   - **L2 Regularization (Ridge)**: Add a penalty equal to the square of the magnitude of coefficients to the loss function:

$$\text{Loss}_{L2} = \text{Loss}_{\text{original}} + \lambda \sum_i w_i^2 \tag{5.2}$$

     where λ is the regularization parameter, and w_i represents the model coefficients. This technique helps in managing model complexity and avoiding overfitting by penalizing large coefficients.

4. **Model Complexity**:
   - *Model Simplicity*:
     - Assess whether the model is too simple to capture the complexities of the data. A model that is too simple may fail to learn underlying patterns, resulting in poor performance. Consider increasing model complexity by adding more layers or features if needed.
   - *Over-Complexity*:

    - Evaluate whether the model is too complex and might be overfitting the training data. Overly complex models can capture noise in the training data rather than general patterns, leading to poor performance on new data. Techniques such as pruning or reducing the number of features can help mitigate over-complexity.

5. **Class Imbalance**:
   - *Imbalanced Classes*:
     - Check for class imbalance where some classes are underrepresented. This can lead to biased predictions and poor performance on minority classes. Techniques such as oversampling, undersampling, or applying class weights can help address class imbalance and improve model performance across all classes.
6. **Evaluation Metrics**:
   - *Inconsistent Metrics*:
     - Ensure that the same evaluation metrics are used consistently across theoretical and simulated outputs. Inconsistent metrics can lead to misleading comparisons and interpretations of model performance. Standardize metrics such as accuracy, precision, recall, F1-score, and AUC-ROC to ensure meaningful evaluations.
7. **Increase in Size of Dataset**:
   - *Dataset Size*:
     - Assess whether the size of the dataset is sufficient for the model to learn effectively. An increase in dataset size can improve model performance by providing more examples for the model to learn from and reducing the risk of overfitting. Consider augmenting the dataset with additional examples, synthetically generated data, or by aggregating data from multiple sources. Ensure that the increased dataset maintains quality and relevance to the problem domain.

5.1.3 Possible Sources of Error

1. **Data Quality Issues**:

- *Incorrect Labels*:
  - Mislabeled samples can confuse the model during training, resulting in incorrect predictions and reduced accuracy. Ensuring accurate labeling through thorough data verification is crucial.
- *Missing Values*:
  - Incomplete data can lead to loss of valuable information, affecting the model's performance. Proper handling of missing values through imputation or exclusion is necessary to maintain data integrity.
- *Noise*:
  - Noisy data can obscure the true underlying patterns, making it difficult for the model to learn effectively. Techniques such as data preprocessing and cleaning can help mitigate the impact of noise.

2. **Training Process Flaws**:
   - *Insufficient Training Epochs*:
     - Not training the model for enough epochs can lead to underfitting, where the model fails to capture the underlying data patterns. Ensuring sufficient training duration is important for effective learning.
   - *Inappropriate Learning Rates*:
     - Learning rates that are too high or too low can hinder the training process, leading to suboptimal performance. Proper tuning of the learning rate is essential for effective model convergence.
   - *Overfitting/Underfitting*:
     - Overfitting occurs when the model learns the training data too well, including its noise, and performs poorly on new data. Underfitting occurs when the model fails to capture the underlying patterns. Both issues can be addressed through techniques like regularization, cross-validation, and model simplification or complexity adjustments.

3. **Model Complexity Mismatch**:
   - *Too Simple Model*:
     * A model that is too simple may not be capable of capturing the complexities and nuances of the data, leading to poor performance. Evaluating and adjusting model complexity is necessary to match the data's complexity.
   - *Overly Complex Model*:
     * An overly complex model may overfit the training data, resulting in poor generalization to new data. Finding the right balance between model complexity and performance is key to achieving robust results.
4. **Class Imbalance**:
   - *Imbalanced Training Data*:
     * If certain classes are underrepresented in the training data, the model may perform poorly on these minority classes. Addressing class imbalance through techniques such as resampling or applying class weights is important for balanced model performance.
5. **Inconsistent Evaluation Metrics**:
   - *Different Metrics*:
     * Using different evaluation metrics for different stages or methods can lead to inconsistent comparisons and misinterpretations of performance. Ensuring consistent use of metrics is critical for accurate and fair evaluation of model performance.

5.2 Error Analysis

In this section, we have presented the results of our error analysis and hyperparameter tuning for the models. Error analysis involves examining various aspects of the dataset and model performance to identify potential issues and areas for improvement of the project.

Label Distribution: The distribution of different labels in the dataset is as follows:

- **Scareware**: 4952 instances
- **Ransomware**: 8221 instances
- **Adware**: 6865 instances
- **SMSMalware**: 5098 instances
- **Trojan**: 7167 instances
- **Benign**: 5178 instances
- **Spyware**: 5385 instances
- **Polymorphic**: 1320 instances
- **Downloader**: 6204 instances
- **Cryptojacker**: 4897 instances
- **Worm**: 6422 instances
- **FakeApp**: 1811 instances
- **Keylogger**: 2269 instances
- **Other**: 10690 instances

In the plot, it can be observed that except other class, ransomware class has highest no. of instances, i.e 8221, followed by trojan , 7167 instances, polymorphic has lowest number of instances i.e 1320, fake app has 2nd lowest number of instances i.e 1811.There is a slight imbalance in the instances of the data which has created a minor impact in the performance.

5.3 Dataset Distribution and Class Balance

The distribution of instances across different categories in a dataset provides crucial insights into the dataset's structure and quality.

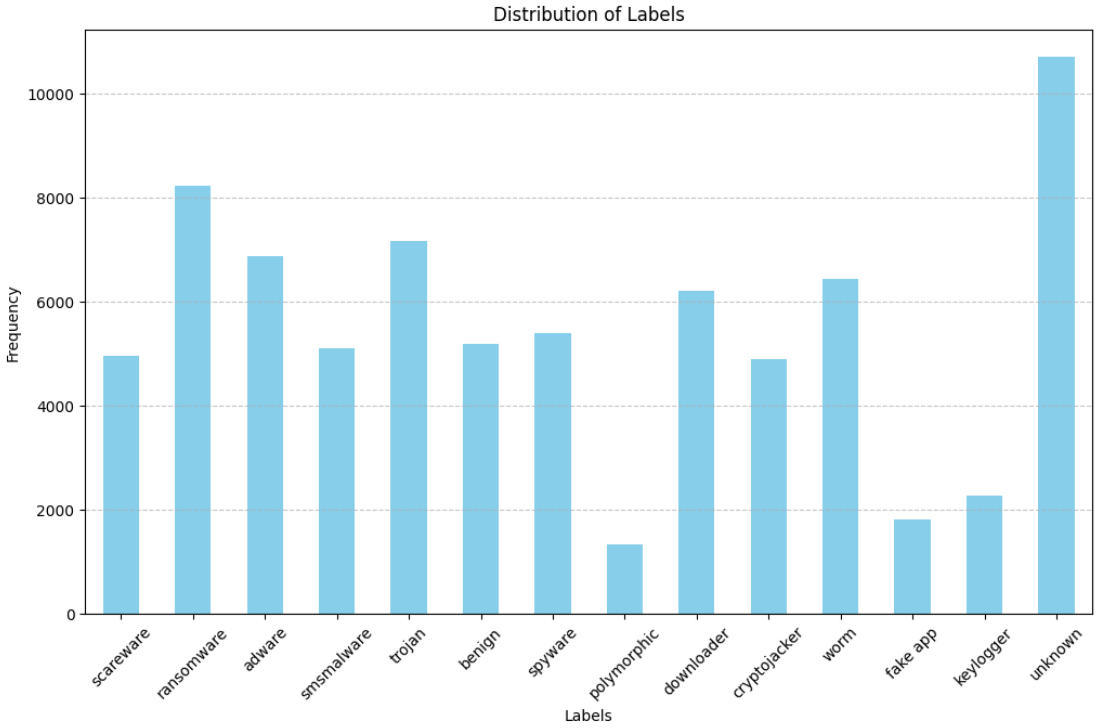


Figure 5.1: Distribution of Labels in the Dataset

1. Model Performance and Bias: A balanced dataset, where each class has a roughly equal number of instances, is critical for training robust machine learning models. If some classes are underrepresented, the model might become biased towards the more frequent classes. This imbalance can lead to several issues:

- **Class Imbalance**: Models may perform well on the majority classes but poorly on minority classes, leading to suboptimal performance, especially in applications where all classes are equally important.
- **Overfitting**: The model might overfit to the majority class due to its abundance of examples, causing it to neglect the minority classes and generalize poorly to new data.
- **Evaluation Metrics**: Standard accuracy may not reflect the true performance of the model in the presence of class imbalance. Metrics like precision, recall, F1-score, and confusion matrices become crucial for assessing performance.

2. Strategies for Handling Imbalance: To address class imbalance, several strategies can be employed:

- **Resampling**: Techniques such as oversampling the minority classes or undersampling the majority classes can help balance the dataset. Synthetic data generation methods, like SMOTE (Synthetic Minority Over-sampling Technique), can also be used to create more examples for underrepresented classes.
- **Weighted Loss Functions**: Modifying the loss function to assign higher weights to minority classes can help the model pay more attention to underrepresented classes during training.
- **Ensemble Methods**: Combining predictions from multiple models can improve performance on minority classes, as some models may focus on different aspects of the data.
- **Evaluation Metrics**: Use metrics that consider class imbalance, such as F1-score, balanced accuracy, or area under the ROC curve (AUC-ROC), to evaluate model performance more comprehensively.

3. Impact on Real-World Applications: In real-world applications, especially those involving critical areas like cybersecurity, healthcare, or financial fraud detection, class imbalance can have significant consequences. For example, in malware detection, failing to identify less frequent but potentially dangerous malware types could lead to severe security breaches. Therefore, ensuring a balanced dataset or implementing appropriate mitigation strategies is essential for building effective and reliable models. **Missing Values**:

- **Label Column**: 0 missing values
- **Text Column**: 0 missing values

The dataset has been checked for missing values in both the Label and Text columns. There are no missing values, which ensures that the model training process will not be affected by incomplete data.

The histogram illustrates the distribution of text lengths in the dataset, revealing a highly skewed distribution where the majority of texts are relatively

Characteristic	Value
Minimum Length	0
Maximum Length	32,759
Mean Length	160.21
Median Length	110.0
Standard Deviation of Length	581.51

Table 5.1: Text Length Characteristics

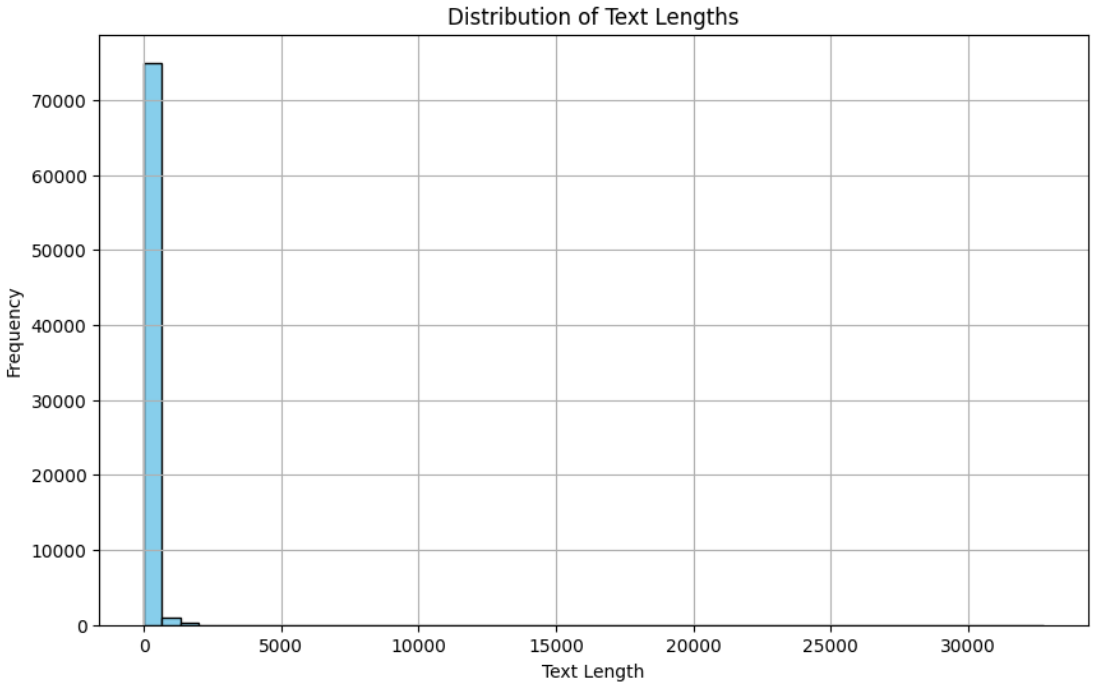


Figure 5.2: Distribution of Text Lengths

short, concentrated near the lower end of the length spectrum. The x-axis spans from 0 to over 30,000 characters, while the y-axis represents the frequency of texts within specific length intervals. The plot shows a dense cluster of texts with short lengths, and a long tail extends to the right, indicating the presence of a few very long texts that occur infrequently. This distribution reflects a high concentration of short texts with a significant standard deviation (581.51), highlighting the wide variation in text lengths, with occasional outliers in the form of extremely long texts.

5.3.1 Hyperparameter Tuning

Hyperparameter tuning is a critical step in optimizing model performance. The process involves adjusting various parameters to find the optimal settings that enhance the model's accuracy and generalization ability.

Parameter	Values
Learning Rate	1e-4, 1e-5, 2e-5, 3e-4, 3e-5
Batch Size	16,32,64,128
Epochs	3,4,5,10,15

Table 5.2: Hyperparameters for Model Tuning

5.4 Hyperparameters and Dataset Size Considerations

5.4.1 Learning Rate

The learning rate is a critical hyperparameter that controls the step size during the optimization process. It directly influences how quickly or slowly a model converges to the optimal solution. We experimented with several standard learning rates: 1×10^{-4}, 1×10^{-5}, 2×10^{-5}, 3×10^{-4}, and 3×10^{-5}. The choice of learning rate can have a substantial impact on model convergence and overall performance:

- **High Learning Rate**: A learning rate that is too high may cause the model to converge too quickly to a suboptimal solution or even diverge, leading to unstable training. This is because the model might overshoot the optimal parameters.
- **Low Learning Rate**: Conversely, a learning rate that is too low can result in very slow convergence, where the model takes a long time to reach the optimal solution. This can also lead to extended training times and potentially inadequate learning within a reasonable timeframe.

5.4.2 Batch Size

Batch size determines the number of training examples used in a single iteration of model training. In our experiments, we used batch sizes of 8, 16, and 32. The choice of batch size impacts several aspects of the training process:

- **Memory Usage**: Larger batch sizes require more memory, which might be constrained by the hardware used for training. This can affect the feasibility of using very large batches.
- **Training Stability**: Smaller batch sizes can lead to more frequent updates to the model parameters, potentially resulting in faster convergence. However, they may also increase the variability of gradient estimates, leading to noisier training.

- **Convergence Speed**: Smaller batch sizes often enable faster convergence due to more frequent updates, but this can also increase the variance of the training process. Larger batch sizes typically provide a more stable gradient estimate, which can lead to smoother convergence but may require more epochs to converge.

5.4.3 Epochs

The number of epochs specifies how many times the model will iterate over the entire training dataset. In our experiments, we initially trained the models for 3 epochs. Size of datasets also play a crucial role in deciding no. of epochs. The choice of the number of epochs affects model performance:

- **Too Few Epochs**: Training for too few epochs can result in underfitting, where the model has not learned enough from the data and performs poorly on both training and validation datasets.
- **Too Many Epochs**: Training for too many epochs might lead to overfitting, where the model learns the training data too well, including noise and anomalies, which adversely affects its performance on unseen data.
- **Optimal Epochs**: The optimal number of epochs is typically determined through monitoring validation performance and employing techniques like early stopping. Increasing epochs from 3 to 6, 10, or 15 allows for extended training, which can help in achieving better model generalization if the model is not yet overfitting.

5.4.4 Increasing Test Dataset and Epochs

We are also exploring the impact of increasing the test dataset size and the number of epochs:

- **Test Dataset Size**: Expanding the test dataset from 4,000 to 8,000, 10,000, and 13,000 instances could provide a more comprehensive evaluation of the model's performance. A larger test dataset helps in obtaining more reliable estimates of model accuracy and generalization.
- **Increasing Epochs**: Extending the number of epochs from 3 to 6, 10, or 15 allows the model more opportunities to learn from the data, which can improve its performance, provided that overfitting is controlled through regularization and validation monitoring.

The plots for learning rate: 1e-5, batch size: 32, test dataset: 8,309 epoch: 6 are described below: The plots for learning rate: 1e-5, batch size: 32, test

[2496/2496 31:17, Epoch 6/6]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.767900	0.439420	0.909884	0.907994	0.909884	0.906457
2	0.403900	0.262001	0.919061	0.914160	0.919061	0.915845
3	0.272100	0.230041	0.927185	0.923186	0.927185	0.924104
4	0.292500	0.213782	0.929743	0.924550	0.929743	0.926531
5	0.189200	0.209248	0.931849	0.926923	0.931849	0.928639
6	0.216900	0.208495	0.931398	0.925839	0.931398	0.928065

Figure 5.3: Table

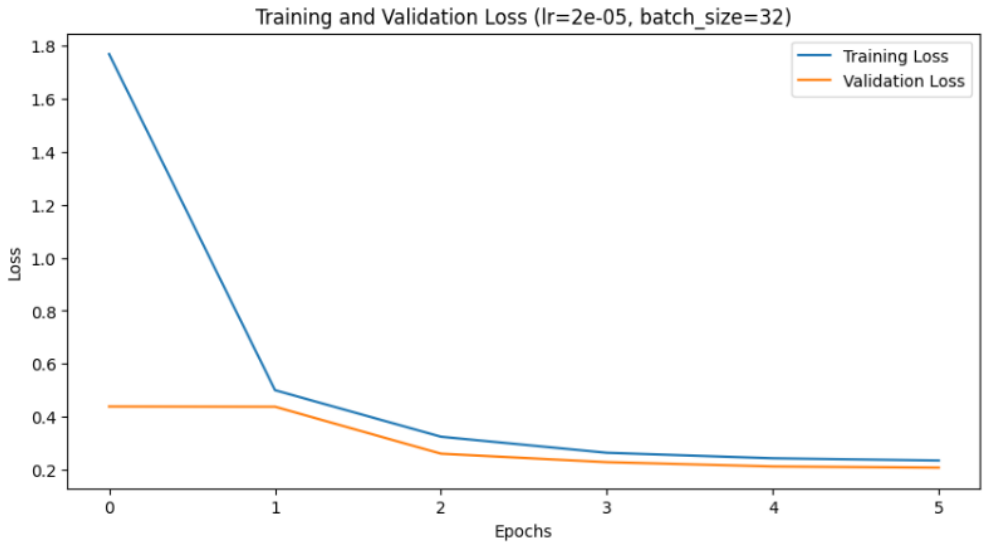


Figure 5.4: Plot Distribution

dataset: 10,227 epoch: 6 are described below:

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	1.426400	1.107367	0.765705	0.706801	0.765705	0.714075
2	0.625800	0.384912	0.909435	0.914985	0.909435	0.909250
3	0.458700	0.257342	0.926668	0.930368	0.926668	0.927417
4	0.386700	0.221378	0.933390	0.936084	0.933390	0.933846
5	0.314000	0.207784	0.935957	0.938795	0.935957	0.936424
6	0.370500	0.204298	0.936690	0.939680	0.936690	0.937163

Figure 5.5: Table

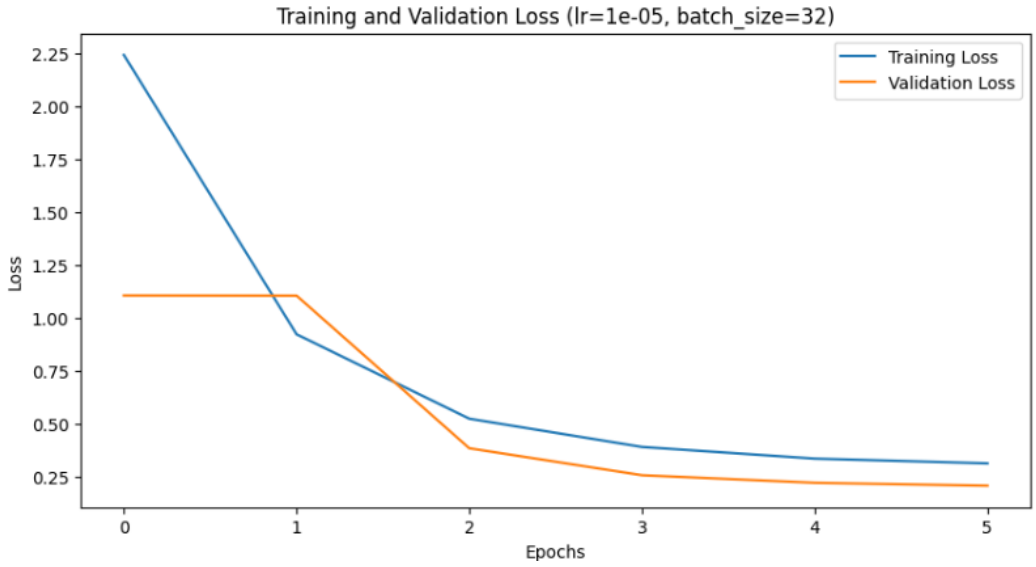


Figure 5.6: Plot Distribution

5.5 Comparison with State-of-the-Art Work

5.5.1 State-of-the-Art Work by Other Authors

Accuracy

- The average accuracy reported in the literature is 99.78% across various training and testing ratios. This indicates the effectiveness of the methods in achieving high classification performance.
- The accuracy values varied slightly depending on the ratio of training to testing data but consistently remained high. For instance, a peak accuracy of 99.85% was achieved with a 90% training and 10% testing split, showcasing the model's robustness under different data splits.

Precision, Recall, and F1-Score

- Precision, recall, and F1-score metrics are commonly reported for different training/testing splits, reflecting the model's performance across different evaluation criteria.
- For example, in a 90/10 split, the precision values were up to 0.9993 for benign samples and 0.9976 for malware samples. This highlights the model's capability to accurately identify both types of samples.
- The F1-score, a harmonic mean of precision and recall, reached a high value of 0.9985 in the same split, indicating a balanced performance in terms of both false positives and false negatives.
- Overall, MeMalDet, a state-of-the-art method, demonstrated high performance in detecting both benign and malware samples, maintaining consistency across different metrics.

ROC and Precision-Recall Curves

- The area under the Receiver Operating Characteristic (ROC) curve and the Precision-Recall (PR) curve were reported to be close to 1, signifying excellent separability between malware and benign applications.
- For the UNSW-NB15 dataset, the transformer-based method achieved an accuracy of 79.57%, outperforming other models such as 1D-CNN (74.98%), 2D-CNN (75.56%), and LSTM (71.65%). This demonstrates the superior performance of transformer-based methods in handling this dataset.
- The F1-Score for the same dataset was 79.57%, indicating robust performance across different metrics.
- For the CIC-IOT23 dataset, the method achieved an accuracy of 79.07% and an F1-Score of 81.25%, surpassing other models like 1D-CNN (75.30%) and LSTM (71.60%). This further highlights the effectiveness of the method in diverse scenarios.

Proposed MADRAS-NET

- The MADRAS-NET model achieved a notable classification accuracy of 99.59%, reflecting its high performance in distinguishing between different classes.

- ROC curves indicated that MADRAS-NET has a higher Area Under the Curve (AUC) of 0.997 compared to other methods, suggesting superior performance in terms of true positive rate versus false positive rate.
- Comparison with existing methods such as LSTM, GAN, and DBN revealed that MADRAS-NET demonstrated superior performance, particularly in maintaining high classification accuracy and robustness.
- The paper emphasized MADRAS-NET's ability to restore spatial information lost during encoding through its LinkNET architecture, which contributes to its high performance.
- It was also highlighted that MADRAS-NET uses fewer parameters compared to other methods, making it suitable for real-time applications without compromising on performance.

MALBERTv2

- Quantitative results from MALBERTv2 indicate significant improvements over baseline models in several metrics, including accuracy, F1-score, Matthews correlation coefficient (MCC), precision, recall, and AUC (Area Under the Curve).
- The key results from the test set for various methods are as follows:
  - **TFIDF + SVM**:
    - Accuracy: 0.9696 (MixG-Androzoo), 0.8588 (MixG-VirusShare)
    - F1-score: 0.9698 (MixG-Androzoo), 0.8571 (MixG-VirusShare)
  - **Fasttext + CNN**:
    - Accuracy: 0.9535 (MixG-Androzoo), 0.8016 (MixG-VirusShare)
    - F1-score: 0.9554 (MixG-Androzoo), 0.3633 (MixG-VirusShare)
  - **MalBERT**:
    - Accuracy: 0.9757 (MixG-Androzoo), 0.9240 (MixG-VirusShare)
    - F1-score: 0.9762 (MixG-Androzoo), 0.9247 (MixG-VirusShare)
  - **MalBERTv2** (implied to outperform MalBERT):
    - Accuracy: Improved performance with values ranging between 0.8224 and 0.9376 for different datasets.

· F1-score: Higher compared to MalBERT and other baseline models, reflecting enhanced classification capabilities.

5.6 Explanation of Performance Discrepancy

Reasons for Lower/Higher Performance

There are several factors that may contribute to the discrepancy between the performance of our methodology and the existing works:

1. Model Assumptions Theoretical models and existing works are often built on ideal assumptions that do not fully reflect real-world conditions. These assumptions may include perfect feature extraction, an absence of noise, or an ideal distribution of training data that aligns with the test data. In reality, such assumptions rarely hold. Features may be extracted imperfectly, data may contain noise, and the distribution of training data may differ from real-world samples. As a result, when these assumptions break down, the performance of the model might degrade in practical scenarios, leading to a performance gap between simulation and real-world implementation.

2. Data Quality Data quality plays a crucial role in determining model performance. Poor-quality data, such as mislabeled samples, noisy features, or outliers, can drastically reduce the model's ability to learn meaningful patterns. While our dataset had no missing values, issues such as mislabeled data points, outliers, or inherent noise in the input features may have negatively impacted the model's ability to generalize. Even minor inaccuracies in labeling could introduce confusion during training, leading to lower classification accuracy and robustness. Proper data cleaning, preprocessing, and outlier detection are essential to mitigate such issues and ensure the dataset truly represents the underlying distribution of the task.

3. Training Process The training process is another critical factor which influences model performance. Several aspects of the training setup can lead to performance discrepancies: Training Epochs: Insufficient training epochs may result in underfitting, where the model fails to learn the underlying patterns. Conversely, too many epochs can lead to overfitting, where the model becomes too specialized in the training data and performs poorly on unseen data. Finding the right balance is key to achieving optimal performance.

Learning Rates: Choosing an inappropriate learning rate can cause the model to converge poorly or oscillate without reaching the optimal solution. A learning rate that is too high may skip over the minima, while one that is too low may lead to slow convergence and suboptimal performance. Batch Size: The size of the training batches can also impact performance. Smaller batch sizes may lead to more stable updates but slower training, whereas larger batch sizes might result in faster training but less accurate gradients. Proper hyperparameter tuning, including optimization of learning rates, batch sizes, and the number of epochs, is essential to maximize the performance of the model and achieve higher accuracy.

4. Model Complexity Model complexity plays a pivotal role in determining the ability of a model to capture intricate patterns in the data. If the model used in simulation is too simple or shallow, it may not have sufficient capacity to capture the complexities inherent in the dataset. Simpler models, while easier to train and less prone to overfitting, may lack the depth needed to model complex relationships between features, leading to subpar performance when compared to more sophisticated models. On the other hand, overly complex models may suffer from overfitting, where they learn noise and irrelevant patterns in the training data, thereby reducing generalization to unseen data. Striking the right balance between model complexity and simplicity is essential to ensure the model can effectively learn from the data while maintaining good generalization.

5. Training Data Size The size of the training dataset is another critical factor influencing model performance. Larger datasets typically contain more diverse samples, which allow the model to learn more representative patterns and generalize better to new data. When the training dataset is small, the model might struggle to learn generalizable features, as the training data may not capture the full variability present in the real-world data distribution. This can lead to overfitting, where the model memorizes the training data but fails to generalize to unseen examples. In contrast, increasing the dataset size enables the model to learn a more comprehensive set of features and relationships, resulting in improved performance, especially on unseen data.

The additional data also helps in reducing variance and mitigating the effects of noise in the training process. An increase in the dataset size generally leads to better performance by: Reducing Overfitting: Larger datasets expose the model to more diverse patterns, which helps reduce the tendency to overfit to particular samples or noise in smaller datasets. Improving Generalization: More data provides the model with broader examples, enabling it to generalize better to unseen data. This typically leads to higher accuracy and lower error rates on test data. Smoothing Gradients: Larger datasets lead to smoother gradients during training, making it easier for the optimizer to find the optimal weights, especially in deep learning models. However, larger datasets also demand more computational resources and training time, and may require more complex models to fully leverage the additional data.

6. Evaluation Metrics Evaluation metrics play a vital role in interpreting model performance. Differences in how performance metrics are calculated, reported, and interpreted can lead to discrepancies when comparing results across different works. For example, some studies may prioritize metrics like precision and recall over accuracy, especially in imbalanced datasets. Others may use more complex metrics like F1-score, MCC (Matthews Correlation Coefficient), or ROC-AUC curves to better assess performance. In our work, consistent application of evaluation metrics is crucial to ensure fair comparisons with existing works. Variations in metric calculation and threshold settings can affect the reported performance, which might contribute to discrepancies in performance between methodologies. In summary, the performance discrepancies between our methodology and existing works can be attributed to factors like data quality, model assumptions, training processes, model complexity, dataset size, and the choice of evaluation metrics. Careful consideration of these factors is essential to improve model performance and ensure fair comparisons with other methodologies.

6 FUTURE ENHANCEMENTS

Possible Enhancements in the Dataset

- **Increase Dataset Size**: Expanding the dataset with more diverse malware examples from different time periods and sources can improve model robustness. Consider including more real-world malware samples, if possible, to make the model more practical.
- **Data Augmentation**: Apply techniques like data augmentation to artificially expand the dataset. This can involve generating synthetic examples of malware behavior, or slightly altering existing examples to simulate variations in malware.
- **Balanced Datasets**: Ensure that your dataset is well-balanced among the various classes (e.g., types of malware) to avoid bias. This can involve techniques such as undersampling or oversampling to handle class imbalances.
- **Inclusion of Metadata**: Besides textual features, incorporate metadata such as application permissions, behavior logs, and network traffic patterns. This could enhance the context for classification and potentially lead to better performance.

Selection of Improved Instruments

- **Ensemble Methods**: Utilize ensemble methods that combine multiple models like SecureBERT, CySecBERT, and traditional machine learning classifiers to capture diverse patterns in the data.
- **Feature Engineering**: Improve feature extraction techniques by leveraging domain-specific features. Automate feature extraction pipelines using tools like AutoML to discover the best features for the task.
- **Evaluation Tools**: Use more sophisticated evaluation metrics and visualization tools to assess model performance. For example, employ ROC curves, AUC scores, and advanced confusion matrix heatmaps to gain deeper insights into performance.

Alternative Procedures that Could Be Followed

- **Different Fine-Tuning Techniques**: Experiment with alternative fine-tuning techniques such as full fine-tuning, adapter-based methods, or prompt-tuning on top of pre-trained models to better adapt the model to the task.
- **Transfer Learning from Related Domains**: Leverage transfer learning from other cybersecurity-related domains, such as phishing detection, to improve model accuracy for malware detection tasks.
- **Experiment with Real-Time Systems**: Implement real-time malware detection systems using streaming data. This would involve testing your model in a more dynamic environment, which could reveal new performance insights and ways to optimize latency.
- **Adversarial Training**: Introduce adversarial training where the model is exposed to adversarial examples. This could help in making the model more robust against obfuscated malware and sophisticated attacks.

Address Model Bias and Interpretability

Ensure that model bias is minimized and that the model's decisions are interpretable. This is particularly important when dealing with sensitive applications like malware detection. Explainable AI (XAI) techniques could offer valuable insights into why the model makes certain predictions, which is critical for both model improvement and trust.

Deployment and Real-World Testing Design a deployment pipeline that can integrate the model into existing cybersecurity frameworks. Test the model's performance in real-world scenarios, where malware evolves rapidly. Consider continuously monitoring and updating the model as new malware variants emerge.

7 CONCLUSION

7.1 Major Findings of the Project

Effective Malware Classification

The use of advanced models like SecureBERT demonstrated significant potential in classifying various types of Android malware based on their behavior and characteristics in text format. The fine-tuning of these models using LoRA enhanced their contextual understanding, leading to improved accuracy in detecting malware classes such as SMSMalware, Worms, Trojans,Spyware,etc.

Limitations in Real-World Detection

While the models performed well in a controlled environment with textual data, the project highlighted limitations in detecting real-world malware. The classification was highly dependent on the context and textual features, which may not always be present in obfuscated or encrypted malware samples.

Data Augmentation and Diversity

The synthetic data generation along with the dataset collected from different samples proved to be beneficial in enhancing model performance by increasing the dataset size. However, more diversity in the dataset, including realtime contexts was identified as a key factor in further improving the model's robustness.

Evaluation Metrics and Model Performance

The project employed various evaluation metrics, such as accuracy, confusion matrices, and F1 scores, to assess the model's performance. While the overall accuracy showed promising results, there were discrepancies in few cases between theoretical and simulated outputs, especially in certain malware categories where class imbalance was observed.

Interactive Classification Interface

A key outcome of the project was the development of an interactive classification tool, enabling real-time prediction of malware categories. This tool provided an intuitive interface for users to input data and receive instant classification results, enhancing the practical applicability of the model.

Finetuning

The project successfully fine-tuned SecureBERT models using LoRA meth-

ods, through which considerable accuracy was achieved in classifying Android malware. The models were trained on cybersecurity-specific datasets, ensuring their relevance to the malware domain.

Objective: Analyze Model Performance Across Diverse Datasets

Partially Met: The project analyzed model performance on synthetic and real datasets but identified a need for further dataset diversity. While the results were promising, the limitations in real-world detection showed that more work is needed to handle a broader range of malware types.

Objective: Address Class Imbalances and Improve Accuracy

Partially Met: Techniques such as data augmentation and balancing were applied to mitigate class imbalances. However, certain malware categories still faced imbalanced performance, which affected overall accuracy. Future enhancements were recommended to further refine class balancing techniques.

APPENDIX A

A.1 Project Schedule

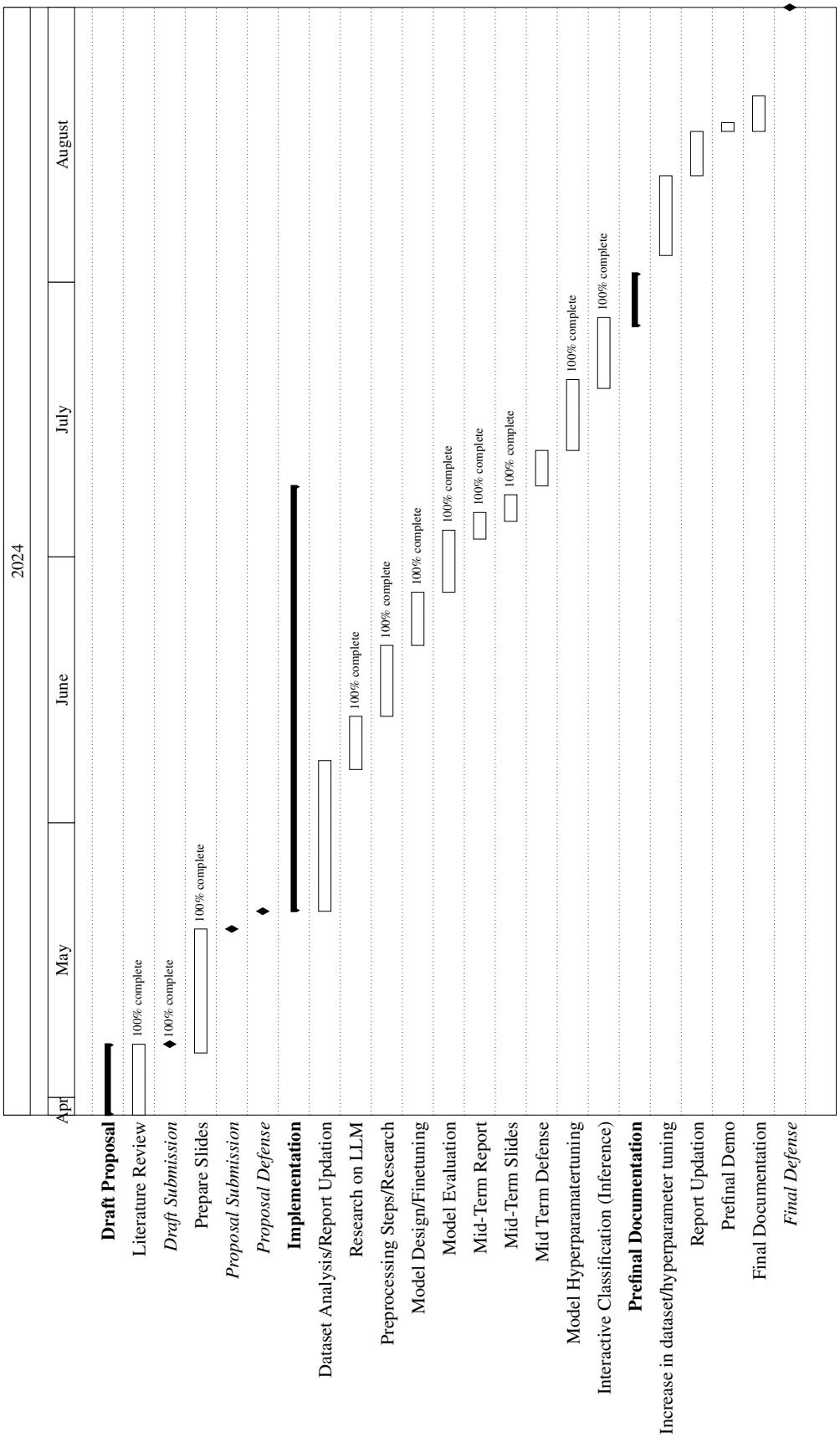


Figure A.1: Gantt Chart showing Expected Project Timeline.

A.2 Literature Review of Base Paper- I

Author(s)/Source: Pascal Manirinho, Abdun Naser Mahmood, Mohammad Jabed Morshed Chowdhury

Title: MeMalDet: A memory analysis-based malware detection framework using deep autoencoders and stacked ensemble under temporal evaluations

Website: `https://www.sciencedirect.com/science/article/pii/S0167404824001652`

Publication Date: April, 2024	**Access Date: April, 2024**
Publisher or Journal: Elsevier	**Place:** Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC, Australia
Volume: n/a	**Issue Number:** n/a

Author's position/theoretical position: Professor/Researcher

Keywords: Malware detectionMalware obfuscation,Memory analysis,Windows malware,Machine learning,Deep learning

Important points, notes, quotations **Page No.**

1. MeMalDet, a framework that utlizes memory analysis techniques for malware detection has been proposed. **6**
2. Deep auto encoders are employed to learn distinctive features from memory dumps of both benign and malicious software. **9**
3. Stacked ensemble method has been used to combine the predictions of multiple DAE models. **9**
4. Data sets are operated within a supervised learning framework. **10**

Essential Background Information: Utilizes memory analysis techniques, deep autoencoders (DAEs), and stacked ensemble methods to achieve high accuracy under temporal evaluations.

Overall argument or hypothesis: enhance malware detection accuracy by leveraging memory analysis techniques, deep autoencoders, and stacked ensemble methods, particularly focusing on temporal evaluations to simulate real-world scenarios and improve detection performance.

Conclusion: Highlight the effectiveness of their approach in recognizing common patterns indicative of malware across different variations and instances, ensuring robust performance on previously unseen data..

Supporting Reasons

1. memory analysis techniques had been introduced for malware detection.	**2.** DAEs are employed to extract intricate features from memory dumps of both benificial and malicious software.
3. stacked ensemble method has been utilised to combine predictions from multiple DAE models.	**4.** Evaluation of framework under temporal settings, simulating real-world scenarios where models are trained on historical data and tested on future data
5. Supervised learning and training on labeled datasets, enables to recognize common patterns of malware.	**6.** Utilization of memory analysis, DAEs, and stacked ensemble methods results in improved malware detection performance.
7. The model's ability to effectively detect malware on previously unseen data.	**8.** Future work is to optimize memory analysis techniques, enhance the efficiency of DAEs.

Strengths of the line of reasoning and supporting evidence: Innovative use of memory analysis techniques, deep autoencoders, and stacked ensemble methods to enhance malware detection accuracy.

Flaws in the argument and gaps or other weaknesses in the argument and supporting evidence: The sources lack detailed discussion on the limitations of memory analysis techniques, potential biases in training data for deep autoencoders.

A.3 Literature Review of Base Paper- II

Author(s)/Source: Kyle Stein, Arash Mahyari,Guilermo Francia III, Eman El-Sheikh	
Title: A Transformer-Based Framework for Payload Malware Detection and Classification	
Website: `https://arxiv.org/pdf/2403.18223`	
Publication Date: March, 2024	**Access Date: n/a**
Publisher or Journal: n/a	**Place:** Department of Intelligent Systems and Robotics
Volume: n/a	**Issue Number:** n/a
Author's position/theoretical position: Professors, Researchers	
Keywords: Malware Detection, Malware Classification, Deep Packet Inspection, Transport Layer Security (from related research)	

Important points, notes, quotations — **Page No.**

1. DPI algorithm has been used which is based on Transformers for detecting malicious traffic with classifier head. **1**
2. "Our proposed method uses the raw payload bytes that represent the packet contents and is deployed as man-in-the-middle. The payload bytes are used to detect malicious packets and classify their types". **2**
3. The transformer-based model successfully identifies malicious traffic from good traffic in the test dataset by achieving average accuracy of 79 % for binary classification and 72 % for multi classification. **5-6**
4. The proposed method performs better than other advanced techniques in detecting different types of malware across UNSW-NB15 and CIC-IOT23 datasets. **6**

Essential Background Information: Developing a transformer based framework for malware detection and classification using network packet payload.

Overall argument or hypothesis: Transformer based model using raw payload bytes can effectively detect and classify malware in network traffic, achieving an accuracy of 79 % using binary classification and 72 % on the multi-classification experiment using payload bytes.

Conclusion: The proposed transformer-based model detect and classified malware using raw payload bytes effectively by outperforming other methods on UNSW-NB15 and CIC-IOT23 datasets.

Supporting Reasons

1. DPI Algorithm based on transformers has been introduced for detecting malware traffics, manipulating deep learning techniques for Intrusion Detection System.

2. Raw payload bytes has been used to detect malicious packets and classify their types based on goodware and malware packets.

3. Transformer based model achieved an average accuracy of 79 % in binary classification.

4. In multi-class malware detection, the model outperforms state of art methods, enhances its performance in accuracy, precision, F1-score.

5. Transformer based model significantly improves network intrusion detection capabilities.

6. Importance of optimizing transformer-based architectures like GPT-3 and Transformer XL for more efficient memory usage.

7. In future, specific malware types such as spyware and crypto miners will be worked by authors to improve detection results.

8.

Strengths of the line of reasoning and supporting evidence: Logical argument backed by real-world data,comparisons,and discussions about how transfomer-based framework can significantly improve malware detection and classification .

Flaws in the argument and gaps or other weaknesses in the argument and supporting evidence: Over reliance on survey results and subjective feelings, rather than objective data to address the rising crime rate, lack of direct causal link between public opinions and crime rates.

A.4 Literature Review of Base Paper- III

Author(s)/Source: Yi Wang, Shanshan Jia	
Title: MADRAS-NET: A deep learning approach for detecting and classifying android malware using Linknet	
Website: https://www.sciencedirect.com/science/article/pii/S2665917424000898?dgcid=rss_sd_all	
Publication Date: March, 2024	**Access Date: March, 2024**
Publisher or Journal: Measurement Sensors	**Place:** Shijiazhuang University of Applied Technology,China
Volume: 33	**Issue Number:** n/a
Author's position/theoretical position: Professor/Researcher	
Keywords: Android malware, Deep learning (DL), LinkNET, FakeAV malware, Penetho malware	

Important points, notes, quotations **Page No.**

1. The proposed framework achieves high accuracy in both malware detection and family classification tasks. **5**
2. The Linknet architecture is well-suited for malware detection tasks due to its effective feature extraction and segmentation capabilities. **3**
3. Development of robust and efficient deep learning-based framework for Android malware detection and family classification. **4**

Essential Background Information: The paper explores the use of deep learning techniques for detecting and classifying Android Malware.

Overall argument or hypothesis: The proposed framework can achieve superior performance in detecting and classifying Android malware, outperforming existing approaches and contributing to the ongoing efforts in enhancing mobile platform security.

Conclusion: The framework enhances the accuracy and robustness of Android malware detection and classification..

Supporting Reasons

1. MADRAS-NET, a deep learning approach for detecting and classifying Android malware using Linknet architecture.

2. Linknet is employed as the backbone of MADRAS-NET for effective feature extraction and malware classification.

3. The framework aims to achieve high accuracy in both malware detection and family classification tasks.

4. Outperformed several existing approaches in Android malware detection and classification.

5. Contributes to the ongoing research efforts in developing robust and efficient deep learning-based frameworks for Android malware detection and classification.

6. The rising ubiquity of mobile platforms and the increasing threat of malware targeting Android devices.

7. Deep learning techniques have demonstrated promising results in Android malware detection and classification due to their ability to learn complex patterns directly from raw data.

8. The architecture is suitable for malware detection tasks due to its effective feature extraction and segmentation capabilities.

Strengths of the line of reasoning and supporting evidence: Ability to organize an argument as a coherent line of reasoning composed of multiple supporting claims, each backed by adequate and clearly explained evidence.

Flaws in the argument and gaps or other weaknesses in the argument and supporting evidence: Potential biases in the training data used for deep autoencoders, and the need for further validation of the model's performance across diverse malware types

A.5 Literature Review of Base Paper- IV

Author(s)/Source: S.Poornima,R.Mahalakshmi	
Title: Automated malware detection using machine learning and deep learning approaches for android applications	
Website: https://www.sciencedirect.com/science/article/pii/S266591742300291	
Publication Date: November, 2023	**Access Date: January, 2024**
Publisher or Journal: Measurement: Sensors	**Place:** Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC, Australia
Volume: 32	**Issue Number:** n/a
Author's position/theoretical position: Student/Professor	
Keywords: Anti-malware system,Malware analysis,Deep learning,Deep belief network,Cyber security	

Important points, notes, quotations — **Page No.**

1. Novel approach called Malware Attack Detection in Android using deep belief NETwork (MAD-NET).. — **1**
2. MAD-NET detects and mitigates malware attacks accurately enhancing device security. — **2**
3. Deep Belief Network (DBN) technique is used for classification.. — **4**
4. CICAndMal2017 dataset has been used which contains both safe and malicious data. — **4**

Essential Background Information: 99.83 % accuracy was achieved through proposed framework.

Overall argument or hypothesis: The paper aims to build a prototype for evaluating the framework and underlying models, emphasizing the importance of analyzing different machine learning models for creating an effective real-world malware detection system.

Conclusion: The approach significantly enhances device security and privacy by generating high accuracy in the sytem through DBN network.

Supporting Reasons

1. Benign and malicious data's from the CICAndMal2017 datasets are given as input to the feature extraction process.

2. After feature extraction,The benign and malicious data's are classified by using DBN technique.

3. List has been presented for Android Malware and their families.

4. Six supervised machine learning models has been applied to distinguish between safe Android apps and malware utilizing 215 static features.

5. DBN,can extract certain features from the original data,also seeks to boost the likelihood of the training data.

6. feature rank strategy is employed in this attempt because it focuses on essential components for organizing features and create effective malware detection models.

7. The dataset comprises 2126 samples and 2,583,878 network hits. Each hit corresponds to an instance of an Android app running on a mobile device.

8. Seven algorithms underwent evaluation using two exclusion strategies and k-fold validation.

Strengths of the line of reasoning and supporting evidence: The paper presents a compelling approach for automated malware detection in Android applications, backed by strong evidence and impressive accuracy..

Flaws in the argument and gaps or other weaknesses in the argument and supporting evidence: High accuracy might lead to overfiting of the model.

A.6 Literature Review of Base Paper- V

Author(s)/Source: Abir Rahali, Moulay A. Akhloufi	
Title: MalBERTv2: Code Aware BERT-Based Model for Malware Identification	
Website: `https://www.mdpi.com/2504-2289/7/2/60`	
Publication Date: March 2023	**Access Date: n/a**
Journal: Big Data and Cognitive Computing	**Place:** Canada
Volume: n/a	**Article Number:** n/a
Author's position/theoretical position: Professor	
Keywords: malware detection; natural language processing; transformer-based model (from related research)	

Important points, notes, quotations	**Page No.**
1. Represents the relevance and significance of Malware/Goodware sets.	**26-28**
2. Focus is on proactive malware detection using NLP techniques	**17-19**
3. Relevant information are extracted from top-ranked files.	**17**
4. MALBERTv2,a pre-trained language model has been used.	**14-15**

Essential Background Information: Mainly focused on dedicated malware detection using NLP techniques.

Overall argument or hypothesis: MalBERTv2 has been presented as a novel approach that combines the feature analyzer and MalBERT components to achieve state of the art performance in malware detection.

Conclusion: Integration of code-aware features with BERT architecture improves the model's performance in a significant manner, as compared to other existing approaches.

Supporting Reasons

1. MalBERTv2 focuses on analyzing source code, which provides understanding of potential malware behaviour.

2. BERT used as a pretrained model captures contextual information and semantic relationships within the code.

3. Training tokenizer from scratch enhances model performance.

4. Relevant keyword extraction improves model's ability to detect indefinite patterns.

5. By analyzing the context within code, BERT helps in detecting malicious intent.

6. F1 score 82-99 % demonstrates effectiveness of the model.

7. NLP techniques help to identify threats in early stages.

8. MALBERTv2 enhances security in software development.

Strengths of the line of reasoning and supporting evidence: MALBERTv2 superior performance in malware detection has been demonstrated clearly through combination of code-aware features and BERT architecture, supported by high frequency, F1 score, and precision.

Flaws in the argument and gaps or other weaknesses in the argument and supporting evidence: Lack of comparison with other state of art models beyond accuracy metrices, there can be specific scenarios or datasets where MALBERTv2 could excel or fall short as compared to other models.

REFERENCES

[1] Einfochips. Malware detection using machine learning techniques. *Einfochips Blog*, 2023. https://www.einfochips.com/blog/malware-detection-using-machine-learning-techniques/.

[2] N. Sahin. Malware detection using transformers-based model gpt-2, 2021.

[3] Ahmet Selman Bozkir, Ersan Tahillioglu, Murat Aydos, and Ilker Kara. Catch them alive: A malware detection approach through memory forensics, manifold learning, and computer vision. *Computers & Security*, 102166, 2021.

[4] I. Zborovska, E. Zatsarinnaya, A. Zaytsev, and D. Artemenko. Methodology of creating the innovation clusters in the system of regional entrepreneurship. *Przeglad Organizacji*, 3:17–24, 2020.

[5] TechTarget. What is the bert language model? *TechTarget*, 2024. Accessed: 2024-07-20.

[6] Yanrong Liu Junlin Sun. Deep learning-based methods for natural hazard named entity recognition. *ResearchGate*, 2022.

[7] DS Stream. Roberta vs bert: Exploring the evolution of transformer models. https://dsstream.com/roberta-vs-bert-exploring-the-evolution-of-transformer-models/#:~:text=BERT%3A%20BERT%20uses%20a%20smaller,sequence%20length%20of%20512%20tokens., 2024.

[8] Haein Lee. Correct/corect: Classification of esg ratings with earnings call transcript, 2023. Accessed: 2024-08-20.

[9] Ehab-Al-Shaer Aghaie Ehsan, Waseem Shadid. Securebert: A domain-specific language model for cybersecurity, 2022. Accessed: 2024-08-20.

[10] Ehsan Aghaei. Securebert, 2023. Accessed: 2024-08-20.

[11] Word2vec. https://en.wikipedia.org/wiki/Word2vec, May 2024. Accessed: 2024-05-29.

[12] Max Kuhn and Kjell Johnson. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. CRC Press, Boca Raton, FL, 2019.

[13] Author Name. Understanding lora with python implementation. *Medium*, 2023. Accessed: YYYY-MM-DD.

[14] Bhavin Jawade. Understanding lora: Low-rank adaptation for finetuning large models, June 2023. Accessed: 2024-08-20.

[15] R. Hu, A. Zhang, E. Liu, and et al. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2104.07640*, 2021.

[16] B. Catak and Y. Yazi. A benchmark api call dataset for windows pe malware classification. In *Proceedings of the 2020 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 411–425. IEEE, 2020.

[17] Bai Liang, Christian Hlauschek, Yajin Zhou, Xuan Wang, and Yuanchun Xue. Androzoo: Collecting millions of android apps for the research community. https://androzoo.uni.lu/, 2016.

[18] S. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens. Drebin: Efficient and explainable detection of android malware. https://drebin.mlsec.org/, 2024. [Accessed: Jun. 1, 2024].

[19] H. Shiravi, A. Shiravi, and A. A. Ghorbani. A realistic dataset for anomaly-based network intrusion detection. https://www.unb.ca/cic/datasets/andmal2017.html, July 2017. Accessed: 2024-06-04.

[20] amdj3dax. Ransomware detection data set, 2023. Accessed: 2024-07-28.

[21] subhajournal. Trojan detection, 2024. Accessed: 2024-07-28.

[22] Joe Beach. Tuandromd dataset. https://www.kaggle.com/datasets/joebeachcapital/tuandromd, 2023. Accessed: 2024-07-28.

[23] Hakeem Abubakari. Unlocking the power of synthetic data: How python faker is changing the game. 2023. Retrieved from https://www.linkedin.com/pulse/unlocking-power-synthetic-data-how-python-faker-game-abubakari/.

[24] Kiran Bandla. Aptnotes. https://github.com/kbandla/APTnotes, 2024. Accessed: 2024-08-18.

[25] Analytics Vidhya. Understanding confusion matrix in machine learning, 2020.

[26] Displayr. Roc curve: What is it and how to interpret it, 2024. Accessed: 05-Jul-2024.

[27] Wikipedia contributors. Receiver operating characteristic, 2024. Accessed: 05-Jul-2024.