Two-Stage Malware Detection Using LSTM and XGBoost

BITE497J – Project I

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Information Technology

by

Aayush Kashyap, Kalindi Singh, Trinabh Mitra

22BIT0504, 22BIT0508, 22BIT0241

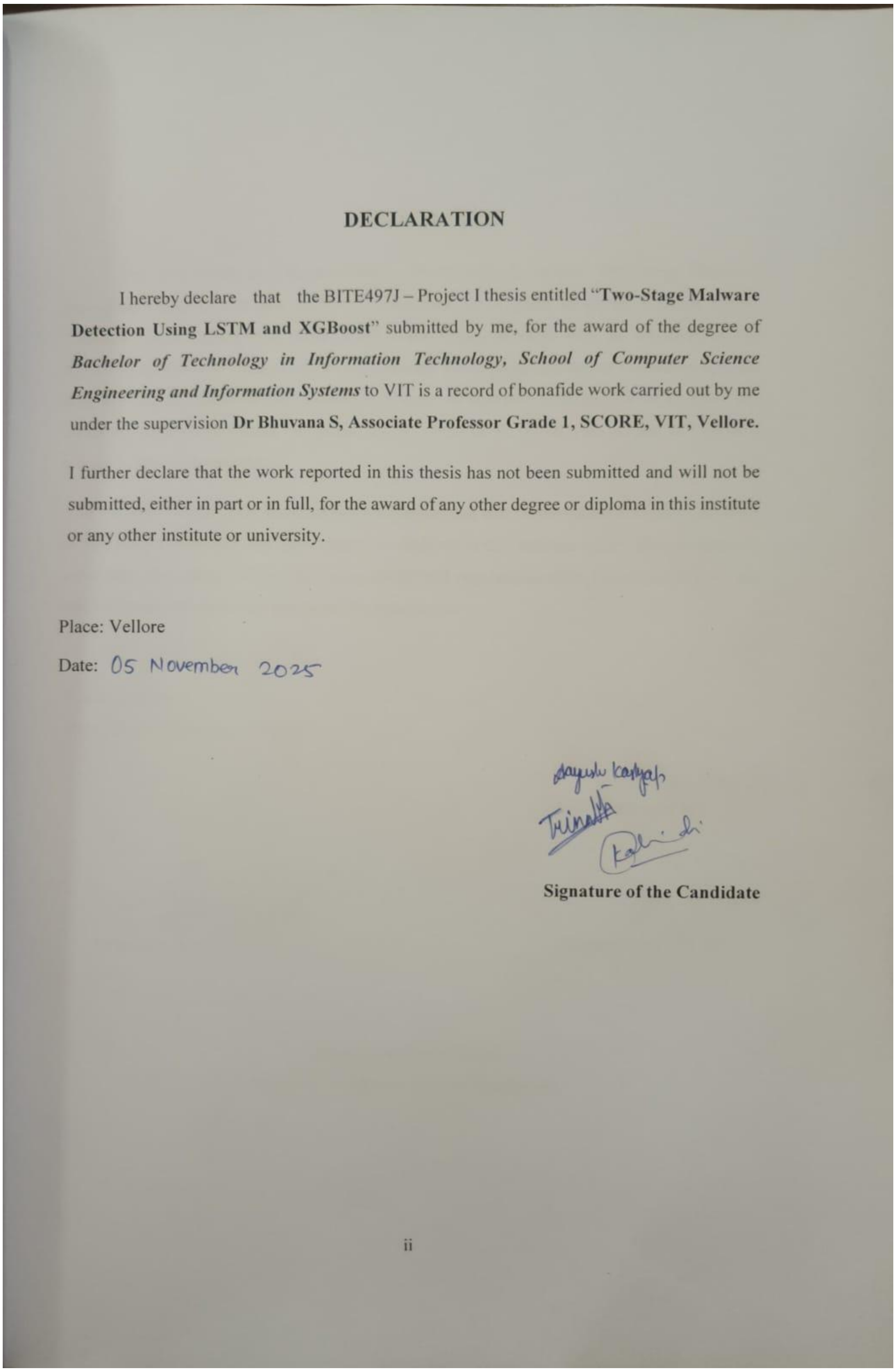
Under the guidance of

Dr. BHUVANA S.

School of Computer Science Engineering and Information Systems

VIT, Vellore

November, 2025



DECLARATION

I hereby declare that the BITE497J – Project I thesis entitled "**Two-Stage Malware Detection Using LSTM and XGBoost**" submitted by me, for the award of the degree of ***Bachelor of Technology in Information Technology, School of Computer Science Engineering and Information Systems*** to VIT is a record of bonafide work carried out by me under the supervision **Dr Bhuvana S, Associate Professor Grade 1, SCORE, VIT, Vellore.**

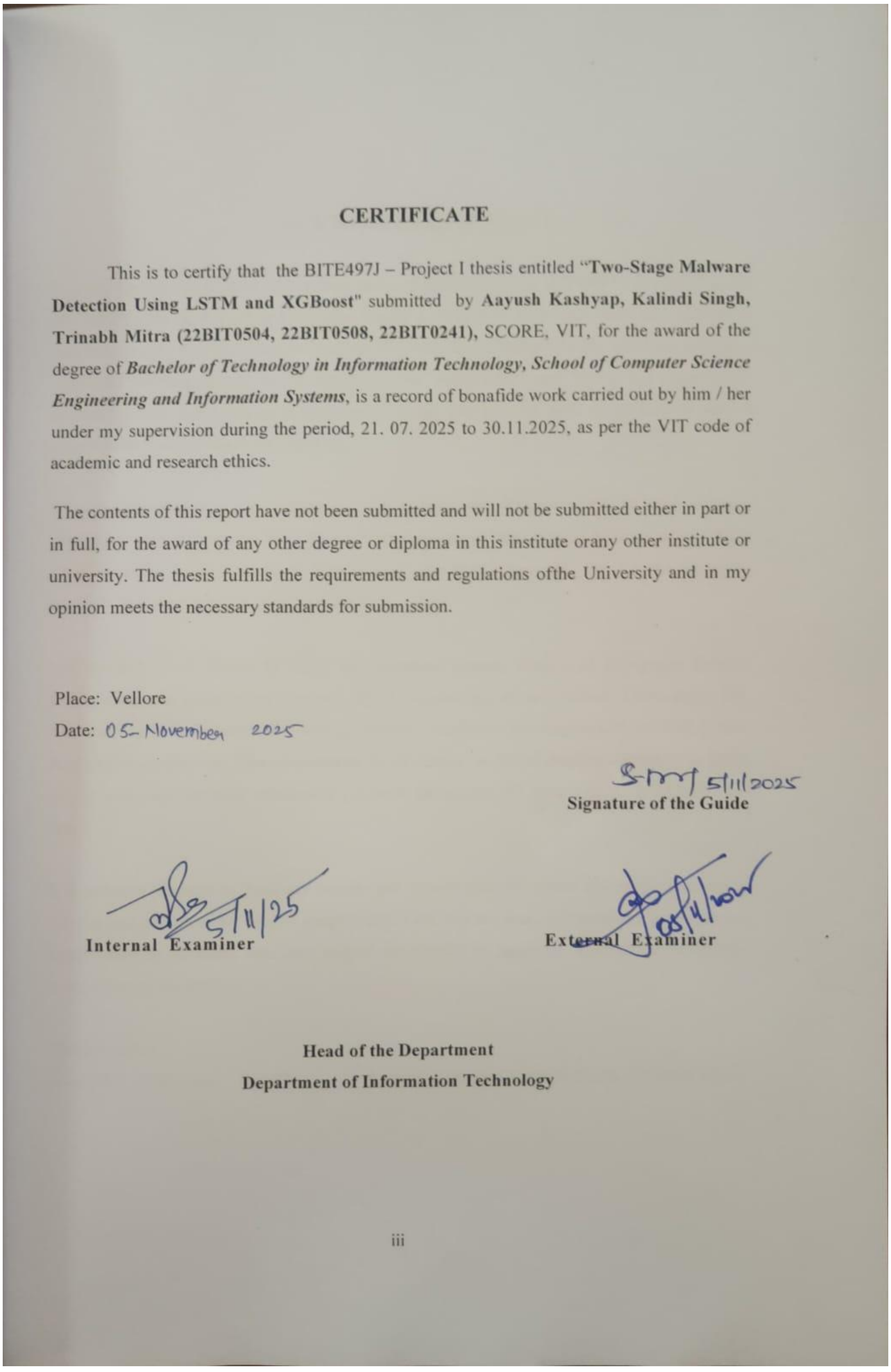
I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 05 November 2025

[illegible]

Signature of the Candidate



CERTIFICATE

This is to certify that the BITE497J – Project I thesis entitled **"Two-Stage Malware Detection Using LSTM and XGBoost"** submitted by **Aayush Kashyap, Kalindi Singh, Trinabh Mitra (22BIT0504, 22BIT0508, 22BIT0241),** SCORE, VIT, for the award of the degree of ***Bachelor of Technology in Information Technology, School of Computer Science Engineering and Information Systems***, is a record of bonafide work carried out by him / her under my supervision during the period, 21. 07. 2025 to 30.11.2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute orany other institute or university. The thesis fulfills the requirements and regulations ofthe University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 05-November 2025

[illegible] 5/11/2025

Signature of the Guide

[illegible] 5/11/25

Internal Examiner

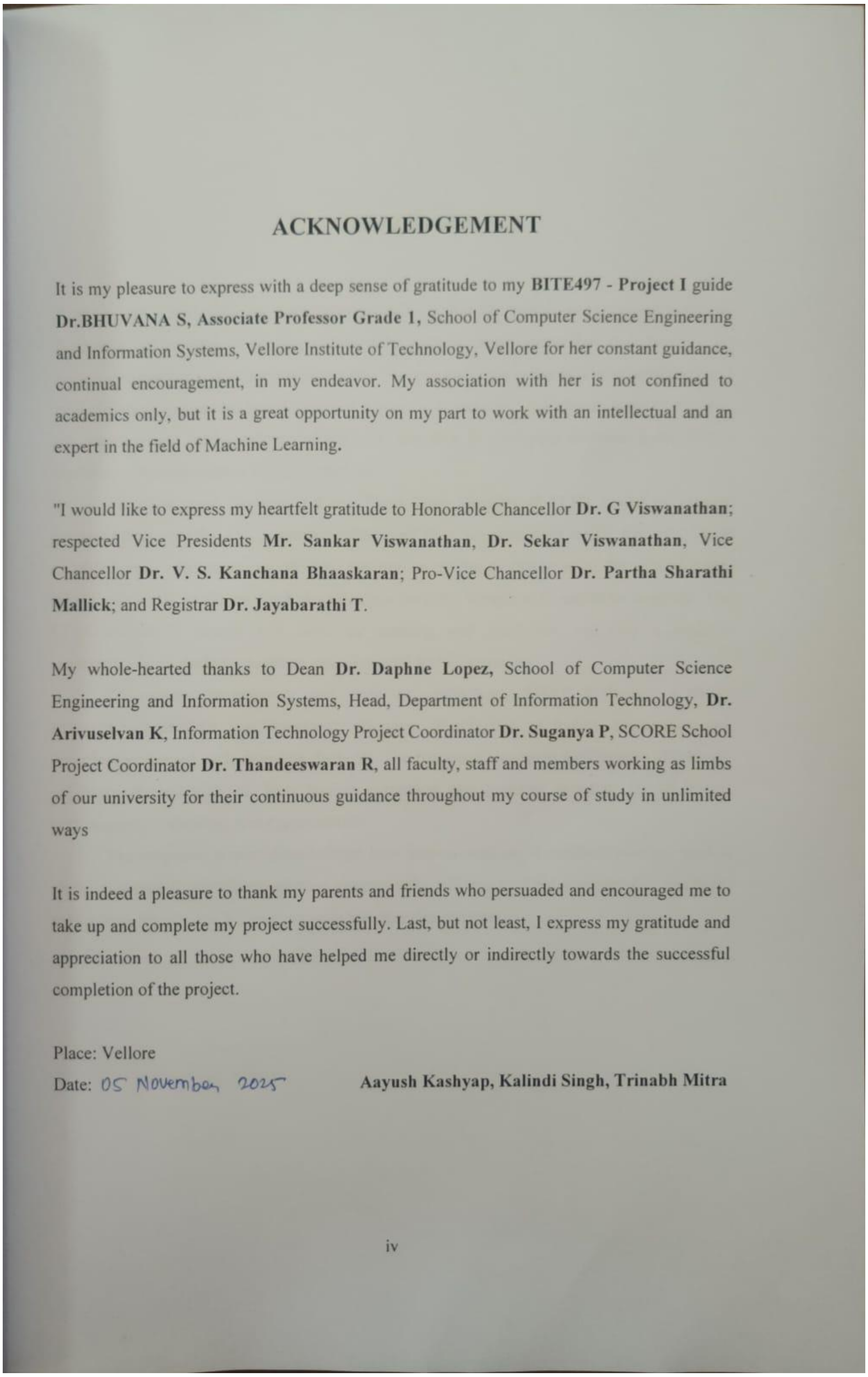
[illegible] 05/11/2025

External Examiner

Head of the Department

Department of Information Technology

iii



ACKNOWLEDGEMENT

It is my pleasure to express with a deep sense of gratitude to my **BITE497 - Project I** guide **Dr.BHUVANA S, Associate Professor Grade 1,** School of Computer Science Engineering and Information Systems, Vellore Institute of Technology, Vellore for her constant guidance, continual encouragement, in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and an expert in the field of Machine Learning**.**

"I would like to express my heartfelt gratitude to Honorable Chancellor **Dr. G Viswanathan**; respected Vice Presidents **Mr. Sankar Viswanathan**, **Dr. Sekar Viswanathan**, Vice Chancellor **Dr. V. S. Kanchana Bhaaskaran**; Pro-Vice Chancellor **Dr. Partha Sharathi Mallick**; and Registrar **Dr. Jayabarathi T**.

My whole-hearted thanks to Dean **Dr. Daphne Lopez,** School of Computer Science Engineering and Information Systems, Head, Department of Information Technology, **Dr. Arivuselvan K**, Information Technology Project Coordinator **Dr. Suganya P**, SCORE School Project Coordinator **Dr. Thandeeswaran R**, all faculty, staff and members working as limbs of our university for their continuous guidance throughout my course of study in unlimited ways

It is indeed a pleasure to thank my parents and friends who persuaded and encouraged me to take up and complete my project successfully. Last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of the project.

Place: Vellore

Date: 05 November 2025 **Aayush Kashyap, Kalindi Singh, Trinabh Mitra**

Executive Summary

With the rise of Android usage, cybercriminals who exploit the Android ecosystem to distribute malicious applications. Detecting malware in Android applications has thus become an important aspect of research in cybersecurity. The proposed system, presents an intelligent and explainable deep learning-based intrusion detection framework. The system aims to enhance the accuracy and interpretability of malware detection in encrypted network flows, where traditional signature-based methods often fail.

The first stage employs a Long Short-Term Memory (LSTM) model to capture temporal dependencies and sequential behaviours from pre-processed traffic features. These extracted deep representations are then fed into an XGBoost classifier in the second stage, which efficiently performs final classification between benign and malicious samples. The CICAndMal2017 dataset was used for training and validation, ensuring a realistic representation of Android malware and encrypted traffic scenarios.

To ensure transparency and trust in predictions, SHAP (SHapley Additive exPlanations) analysis was integrated, allowing visualization of feature importance and contribution toward each detection decision. The proposed two-stage architecture effectively combines the strength of deep feature extraction and gradient boosting for improved performance, scalability, and explainability.

The proposed system demonstrates how implementation of machine learning models along with explainable AI can enhance cybersecurity, offering a robust, interpretable, and high-performing framework for modern malware detection systems.

CONTENTS **Page No.**

List of Figures

List of Tables

List of Abbreviations

LSTM	Long Short Term Memory
XGBoost	Extreme Gradient Boosting
AUC	Area Under Curve
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
RMSE	Root Mean Squared Error
F1 Score	Harmonic Mean of Precision and Recall
SSl/TLS	Secure Socket Layer,Transport Layer Security
IAT	Inter Interval Time
XAI	Explainable Artificial Intelligence
SHAP	SHapely Additive exPlanations

SYMBOLS AND NOTATION

TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

CHAPTER 1
INTRODUCTION

1.1 Overview

With the rapid expansion of mobile devices, Android has emerged as the dominant operating system, powering billions of smartphones and tablets globally. However, this growth has simultaneously attracted cybercriminals who exploit the Android ecosystem to distribute malicious applications. Detecting malware in Android applications has thus become a crucial area of research in cybersecurity. Traditional malware detection systems rely heavily on static or dynamic analysis of application code, but these methods face severe limitations when the traffic is encrypted, as modern applications increasingly use HTTPS and VPN protocols.

The challenge of malware detection in encrypted network traffic lies in the inability to access payload content directly. Therefore, security researchers have shifted their focus toward traffic-based features such as packet size, flow duration, and direction to identify anomalies indicative of malicious activity. This project proposes a Two-Stage Malware Detection System (MDS) that combines Long Short-Term Memory (LSTM) networks and Extreme Gradient Boosting (XGBoost) to effectively detect and classify malicious traffic patterns in encrypted Android communications using the CICAndMal2017 dataset.

1.2 Problem Statement

The increasing sophistication of Android malware and the widespread adoption of encryption protocols have rendered traditional signature-based Intrusion Detection

Systems (IDS) ineffective. Since the payload data is no longer visible, detecting malicious activity becomes challenging using conventional methods. Moreover, many existing machine learning models fail to generalize across diverse network conditions, leading to high false-positive rates.

1.3 Objective

The primary objectives of this research are:

- To preprocess and extract relevant traffic-based features from the CICAndMal2017 dataset.
- To design and implement a two-stage detection framework combining deep learning (LSTM) and machine learning (XGBoost).
- To compare the performance of the proposed two-stage approach with a traditional single-stage LSTM model.
- To evaluate the model using standard classification metrics such as accuracy, precision, recall, and F1-score.
- To interpret the model's decision-making process using SHAP (SHapley Additive exPlanations) for explainability.

1.4 Scope of the Project

This project focuses specifically on encrypted Android malware traffic detection. The study does not rely on the actual payload but instead analyses flow-based statistical features. The system aims to:

- Identify malware behaviour patterns in encrypted traffic flows.
- Work in real-time scenarios with limited computational overhead.

- Provide transparency in model decisions through SHAP-based explainability.

The scope does not include reverse engineering APK files or static code analysis, the focus remains purely on network-based anomaly detection.

1.5 Motivation

The main motivation behind this research stems from the exponential rise in encrypted traffic and the corresponding increase in malware activities that hide within such channels. Conventional IDS struggle in this scenario, creating an urgent need for intelligent systems capable of identifying malicious behaviour without decrypting the traffic.

Furthermore, as cybersecurity professionals emphasize AI-driven threat detection, integrating deep learning (LSTM) for sequential flow learning and XGBoost for efficient classification offers a powerful hybrid approach. The project also introduces explainability using SHAP values, ensuring the model's predictions can be trusted and understood by analysts.

1.6 Background

Most studies related to the topic rely on static features. Less explored dynamic analysis motivates towards the need to research on flow level features like packet size, inter-arrival time, duration, direction.

The best performance till date has been obtained by combining BiLSTM with XGBoost, yielding F1 and accuracy scores of 95.12% and 99.33%, respectively, after extensive training.[1]

While several studies have integrated explainable AI techniques to enhance interpretability, this remains an emerging and relatively underdeveloped area. More notably, existing work has largely overlooked the challenge of generalizing to previously unseen or evolving malware behaviors. The lack of explicit efforts to improve model robustness against such new threats underscores a critical gap in the literature one that the current study seeks to address.

Asmitha et al. [4] focused on designing lightweight detection methods suitable for resource-constrained environments. They proposed a static APK analysis technique that extracts permissions, activities, and receivers to detect previously unseen malware. ViTDroid [5] utilized visual transformers to analyze opcode sequences from Android malware, achieving a remarkably low false-positive rate of 0.0019.

. Rashid et al. [6] proposed a hybrid dual-branch deep neural network that combines static and dynamic features for robust Android malware detection. Their model processes app metadata and behavior logs in parallel, achieving a high detection accuracy of 98.2% across five datasets, including Drebin and VirusShare. Notably, the architecture includes an explainability component that maps predictions to specific behavioral patterns, enhancing model transparency.

Recently, deep learning has emerged as a leading approach across various domains, including Android malware detection [7–10]. Convolutional Neural Networks (CNNs), in particular, have demonstrated high performance and are widely adopted in recent research on Android malware detection [11–15]. Explainable AI offers post-hoc explanations that help users interpret and improve model performance [16,17].

Traditional evaluation of machine learning models focused primarily on accuracy overlooks critical aspects of real-world deployment. Building trustworthy AI requires addressing both technical and ethical aspects, including robustness, generalization, reproducibility, explainability, transparency, fairness, privacy, and accountability [2]. Explainability and transparency are especially critical in cybersecurity, where understanding a model's decisions is essential. Deep learning's ‘‘black-box’’ nature creates trust and usability challenges.

1.7 Contributions

The key contributions of this project include:

- A novel two-stage malware detection pipeline integrating LSTM and XGBoost.
- A thorough experimental comparison between single-stage and hybrid approaches.
- Visualization of feature importance and interpretability through SHAP.
- A ready-to-deploy system capable of identifying encrypted malware traffic in Android environments.

CHAPTER 2

DISSERTATION / INTERNSHIP DESCRIPTION AND GOALS

The proposed project titled "Two-Stage Malware Detection System using LSTM and XGBoost on Encrypted Android Traffic" aims to enhance the security of Android devices by detecting malware hidden within encrypted network communications. With the growing use of HTTPS and VPN services, most network data today is encrypted, making traditional signature-based intrusion detection systems less effective. This project bridges that gap by applying Artificial Intelligence and Machine Learning techniques to analyze encrypted traffic behavior and detect potential threats.

The dissertation focuses on developing a two-stage architecture for malware detection:

- In Stage 1, a Long Short-Term Memory (LSTM) model learns sequential patterns and extracts temporal features from encrypted network data. LSTMs are well-suited for time-series analysis and can capture dependencies that occur across multiple packets within the same network flow.
- In Stage 2, an XGBoost classifier utilizes the latent features learned by the LSTM to make a final prediction — whether the traffic flow is benign or malicious. This layered approach improves the precision of detection and minimizes false alarms compared to a single-stage model.

The main goal of the project is to create a reliable and scalable detection framework that can operate effectively even when payload data is unavailable due to encryption.

The model relies entirely on metadata-based flow features, such as packet size, duration, inter-arrival time, and byte count, rather than relying on content inspection.

Additionally, the system integrates SHAP (SHapley Additive exPlanations) to interpret model predictions and provide insights into which features most influence the classification outcome. This adds transparency and helps cybersecurity professionals understand the reasoning behind detections.

The overall objectives of the project can be summarized as follows:

- ✓ To preprocess and extract useful statistical features from encrypted Android network traffic datasets (CICAndMal2017).
- ✓ To train a deep learning model (LSTM) for temporal feature extraction and flow pattern analysis.
- ✓ To build an ensemble machine learning model (XGBoost) for final malware detection and performance optimization.
- ✓ To evaluate the system using key performance metrics such as accuracy, precision, recall, and F1-score.
- ✓ To apply SHAP for explainability, ensuring the AI model's decisions can be justified and understood.

Through this dissertation, the project aims to contribute to network-based malware detection research by proposing a hybrid deep learning approach that balances accuracy, interpretability, and computational efficiency, suitable for real-world deployment in modern encrypted network environments.

CHAPTER 3

TECHNICAL SPECIFICATION

The Two-Stage Malware Detection System combines deep learning and machine learning methods to analyze encrypted Android network traffic for malicious behavior. This chapter provides a detailed overview of the software, hardware, dataset, and libraries used to build, train, and evaluate the system. It also explains the design requirements and configurations that make the proposed system efficient and reproducible.

3.1 Hardware Requirements

The implementation and training of the LSTM and XGBoost models require significant computational resources, especially for large-scale network datasets such as CICAndMal2017. The following hardware specifications were used during project development:

Table 3.1 Hardware Requirements : The table lists out the hardware components required for the efficient execution of the system.

Component	Specification
Processor	Intel Core i7 11th Gen / AMD Ryzen 7
RAM	16 GB DDR4
Storage	512 GB SSD
GPU	NVIDIA GTX 1650
Operating System	Windows 10 / 11 (64-bit)
Python Version	3.11

These specifications ensured smooth data preprocessing, model training, and visualization using SHAP without significant latency or memory issues.

3.2 Software Requirements

The computations required robust software, editors, libraries that are listed in Table 3.1.

Table 3.1 Software Requirements : The table lists out the software components required for the efficient execution of the system.

Software	Purpose
Python (3.11)	Primary programming language for model development
TensorFlow / Keras	Deep learning framework for LSTM implementation
XGBoost	Gradient-boosted tree model for stage-2 classification
NumPy & Pandas	Data manipulation and feature engineering
Matplotlib / Seaborn	Visualization of performance metrics
SHAP	Explainable AI framework for model interpretability
Scikit-learn	Used for preprocessing, metrics, and model utilities
Jupyter Notebook / VS Code	Development and debugging environment
Git	Version control for managing source code
CSV & HDF5 File Formats	Storage formats for datasets and trained models

3.3 Dataset Specification

The project uses the CICAndMal2017 dataset, developed by the Canadian Institute for Cybersecurity (CIC). The CICAndMal2017 dataset [18], developed specifically for Android malware research, focuses on analyzing network flow features. It includes over 84 flow-based attributes extracted using CICFlowMeter V3 [19].

This dataset captures real Android traffic, both benign and malicious, generated by over 190 Android applications under controlled conditions. Each sample represents an encrypted network flow, consisting of statistical and behavioral features extracted from packet headers rather than payload data.

- Dataset Name: CICAndMal2017
- Type: Encrypted Android traffic (HTTPS and VPN)
- Classes: Benign and Malicious
- Number of Samples: Approximately 12,000
- Number of Features: 84 flow-based attributes
- Feature Categories: Packet size statistics, flow duration, byte counts, inter-arrival times, protocol types

The dataset is highly suitable for this project as it reflects real-world encrypted traffic patterns where traditional signature-based malware detection fails.

3.4 Model Architecture Summary

- Stage 1: Long Short-Term Memory (LSTM) model trained to learn sequential dependencies and temporal relationships among traffic features.
- Stage 2: XGBoost model trained using extracted latent features from the LSTM to perform final binary classification (malicious or benign).
- Explainability: SHAP is used post-training to interpret the influence of each feature on the model's predictions.

This hybrid model ensures both accuracy and interpretability, combining the deep sequential learning capabilities of LSTM with the structured decision-making power of XGBoost.

3.5 Development Workflow

- Data preprocessing (cleaning, normalization, and feature selection).
- LSTM model training for temporal feature extraction.
- Extraction of learned embeddings as input for Stage-2 XGBoost.
- XGBoost model training for malware classification.
- Evaluation using confusion matrix, accuracy, recall, and F1-score.
- Explainability analysis using SHAP plots for model interpretation.

This systematic workflow ensures that every stage contributes meaningfully to achieving high-quality and trustworthy malware detection.

CHAPTER 4

DESIGN APPROACH AND DETAILS

This chapter elaborates on the overall architecture, workflow, and design methodology of the proposed two-stage malware detection system (MDS). The design integrates deep learning and machine learning models to efficiently detect malware in encrypted Android traffic, ensuring both high accuracy and model transparency.

4.1 System Architecture Overview

The proposed system follows a two-stage hybrid architecture designed to combine the temporal feature extraction capabilities of deep learning with the interpretability and precision of gradient-boosting models.

1. Stage 1 (LSTM Network):
    - Learns sequential dependencies from encrypted traffic data.
    - Extracts latent representations from raw flow features.
    - Outputs an embedding vector summarizing the traffic behavior.
2. Stage 2 (XGBoost Classifier):
    - Takes LSTM embeddings as input features.
    - Performs final binary classification (benign or malicious).
    - Enhances accuracy and robustness by focusing on patterns learned in the first stage.

The following workflow describes the step-by-step design:

- **Input:** Encrypted Android traffic features from the CICAndMal2017 dataset.
- **Preprocessing:** Normalization, cleaning, and reshaping of data.
- **Stage 1:** LSTM-based feature extractor model.
- **Stage 2:** XGBoost-based malware classifier.
- **Output:** Final prediction label (Malicious / Benign) with SHAP-based explainability.

4.2 Workflow Diagram

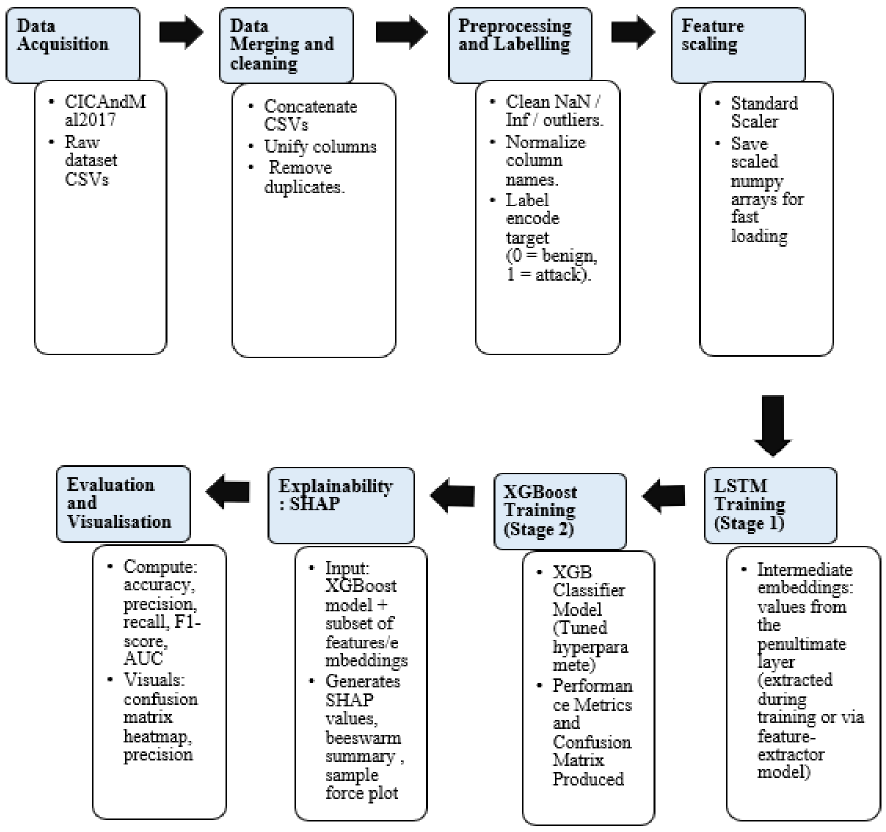

Data Acquisition
• CICAndMal2017
• Raw dataset CSVs
Data Merging and cleaning
• Concatenate CSVs
• Unify columns
• Remove duplicates.
Preprocessing and Labelling
• Clean NaN / Inf / outliers.
• Normalize column names.
• Label encode target (0 = benign, 1 = attack).
Feature scaling
• Standard Scaler
• Save scaled numpy arrays for fast loading
LSTM Training (Stage 1)
• Intermediate embeddings: values from the penultimate layer (extracted during training or via feature-extractor model)
XGBoost Training (Stage 2)
• XGB Classifier Model (Tuned hyperparamete)
• Performance Metrics and Confusion Matrix Produced
Explainability : SHAP
• Input: XGBoost model + subset of features/embeddings
• Generates SHAP values, beeswarm summary , sample force plot
Evaluation and Visualisation
• Compute: accuracy, precision, recall, F1-score, AUC
• Visuals: confusion matrix heatmap, precision

Figure 4.1 Workflow diagram: The diagram represents various components of the system. Data is procured and processed. LSTM and XGBoost are trained upon the features extracted. SHAP provides explainability. Various modes of evaluation metrics are produced.

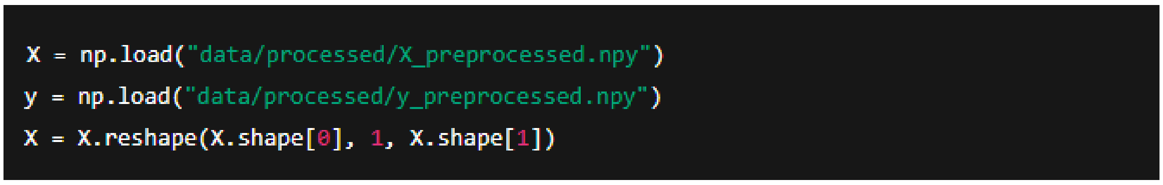
4.3 Data Preprocessing

Proper preprocessing is essential for handling large network datasets. The preprocessing pipeline includes:

- **Feature Selection:** Dropping redundant or irrelevant attributes.
- **Normalization:** Applying Min-Max scaling to standardize feature ranges.
- **Handling Missing Values:** Replacing nulls or corrupted entries.
- **Train-Test Split:** Splitting data into 80% training and 20% testing.

- **Reshaping for LSTM:** Converting feature vectors into 3D input format (samples, timesteps, features) for sequential learning.

Example snippet used:



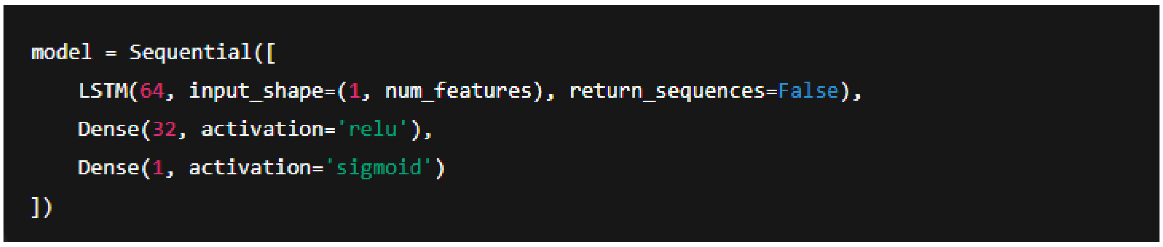
```
X = np.load("data/processed/X_preprocessed.npy")
y = np.load("data/processed/y_preprocessed.npy")
X = X.reshape(X.shape[0], 1, X.shape[1])
```

Figure 4.2 Data Preprocessing: Data is reshaped for application of LSTM model.

This ensures that the input to the LSTM captures the temporal structure inherent in encrypted traffic.

4.4 Stage 1 - LSTM Feature Extraction

The Long Short-Term Memory (LSTM) model is used as a feature extractor. LSTM networks are highly effective in capturing long-term dependencies in sequential data, which is crucial for understanding traffic flows.



```
model = Sequential([
    LSTM(64, input_shape=(1, num_features), return_sequences=False),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

Figure 4.3 LSTM Network Architecture Used for Binary Classification: This figure illustrates the architecture of the LSTM model comprising an LSTM layer with 64 units, followed by a 32-neuron dense layer with ReLU activation and an output layer with a sigmoid activation function for binary classification (malicious vs. benign traffic).

- **Input Layer:** Takes the 3D reshaped traffic data.
- **Hidden Layers:** Contain LSTM and dense units for feature learning.

- **Output Layer:** Produces a preliminary prediction and a latent feature representation.

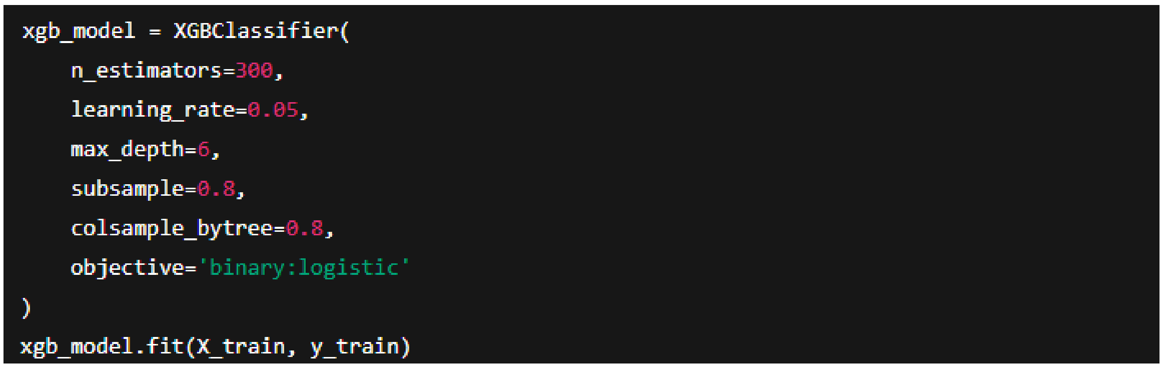
The second-to-last layer's output is extracted as LSTM embeddings, which serve as high-level learned features for Stage 2.

4.5 Stage 2 – XGBoost Classifier

After obtaining LSTM embeddings, the XGBoost classifier refines the decision-making process. XGBoost (Extreme Gradient Boosting) is chosen for its:

- High performance with structured data
- Built-in regularization to prevent overfitting
- Ability to capture nonlinear relationships
- Feature importance interpretability

Example snippet:



```
xgb_model = XGBClassifier(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='binary:logistic'
)
xgb_model.fit(X_train, y_train)
```

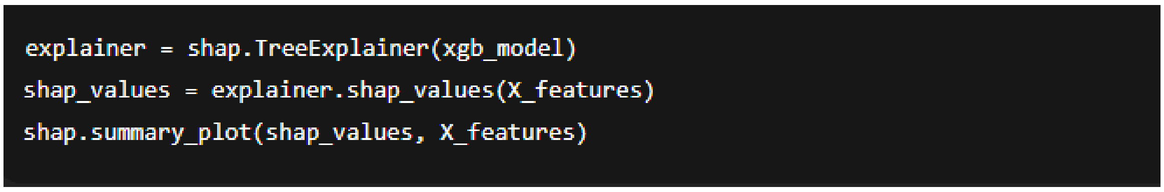
Figure 4.4 XGBoost Classifier Configuration for Stage-2 Training: This figure shows the XGBoost model setup with 300 estimators, a learning rate of 0.05, and a maximum depth of 6, optimized for binary logistic classification in the second stage of the hybrid framework.

Once trained, this model outputs final predictions and supports SHAP-based explainability.

4.6 Explainable AI (SHAP Integration)

To improve model trustworthiness, SHAP (SHapley Additive exPlanations) is used to interpret the XGBoost classifier's decisions. SHAP values indicate how each feature impacts the prediction probability.

Example**:**



```
explainer = shap.TreeExplainer(xgb_model)
shap_values = explainer.shap_values(X_features)
shap.summary_plot(shap_values, X_features)
```

Figure 4.5 SHAP-Based Feature Importance Analysis: This code snippet demonstrates the use of SHAP's TreeExplainer to interpret the XGBoost model by computing SHAP values and generating a summary plot to visualize the most influential features.

This step ensures transparency, showing which traffic features most strongly contribute to detecting malware.

4.7 Advantages of Two-Stage Design

Table 4.1 Comparison Between Single-Stage and Two-Stage Hybrid Models: This table presents a comparative analysis of the single-stage and two-stage hybrid models across key performance aspects such as feature learning, accuracy, interpretability, and adaptability. The two-stage hybrid model demonstrates superior capability through deep sequential feature extraction, higher accuracy, improved interpretability using SHAP, and enhanced adaptability to evolving malware patterns.

Aspect	**Single-Stage Model**	**Two-Stage Hybrid Model**
Feature Learning	Limited (shallow)	Deep sequential extraction
Accuracy	Moderate	High
Interpretability	Low	High (via SHAP)
Adaptability	Weak	Robust against new malware patterns

This hybrid design allows the model to balance performance and interpretability, addressing key challenges in encrypted traffic analysis.

4.8 Summary

The proposed design effectively integrates the temporal strength of LSTM with the structured decision-making of XGBoost. The modular structure allows independent tuning of each stage and easy retraining when new data becomes available. Additionally, the use of SHAP ensures explainability, making the system suitable for real-world cybersecurity applications where transparency is as critical as accuracy.

CHAPTER 5

IMPLEMENTATION AND RESULTS

This chapter describes the implementation process, experimental setup, model training, and evaluation results of the proposed two-stage malware detection system (MDS). The section focuses on how the models were developed, trained, and tested on the CICAndMal2017 dataset, followed by performance comparisons and evaluation metrics.

5.1 Experimental Setup

The proposed models were implemented using Python 3.11 with popular machine learning and deep learning libraries such as TensorFlow, Keras, Scikit-learn, XGBoost, NumPy, and Matplotlib. All experiments were conducted on a local system with the following configuration:

Table 5.1 Experimental setup: The tabulation of components required for the computation.

Specification	**Details**
Operating System	Windows 10 (64-bit)
Processor	Intel Core i7 (11th Gen)
RAM	16 GB
GPU	NVIDIA GTX 1650
Frameworks Used	TensorFlow, Keras, XGBoost
Dataset	CICAndMal2017 (Encrypted Android Traffic)

The entire project was structured in a modular way, separating data preprocessing, model training, evaluation, and explainability into independent Python scripts stored under the src/ directory.

5.2 Dataset Description

The CICAndMal2017 dataset was used for this study. It consists of network traffic traces collected from both benign Android applications and malware samples. Each record represents encrypted traffic features extracted from network flows.

Key Characteristics:

- Total samples: ~80,000
- Features: 80+ network attributes (flow duration, packet size, source/destination statistics, etc.)
- Classes: 2 (Benign and Malicious)
- Data type: Encrypted Android traffic (TLS/SSL flows)
- Collected by: Canadian Institute for Cybersecurity (CIC)

The dataset was chosen because it realistically simulates encrypted communication, making it ideal for evaluating deep learning approaches in modern malware detection tasks.

5.3 Data Preprocessing

Data preprocessing was performed using a custom Python script (src/01_data_preprocessing.py) with the following key steps:

1. Data Loading:
   CSV files from different malware families were merged into one unified dataset.
2. Feature Cleaning:
   - Removed columns with constant or missing values.
   - Converted categorical columns into numerical format.
3. Normalization:
   Applied Min-Max scaling to bring all features within a uniform range (0–1).
4. Train-Test-Split:
   Data was divided into 80% training and 20% testing sets.
5. Reshaping-for-LSTM:
   Reshaped input features into 3D arrays of shape (samples, timesteps=1, features) to match the LSTM input format.

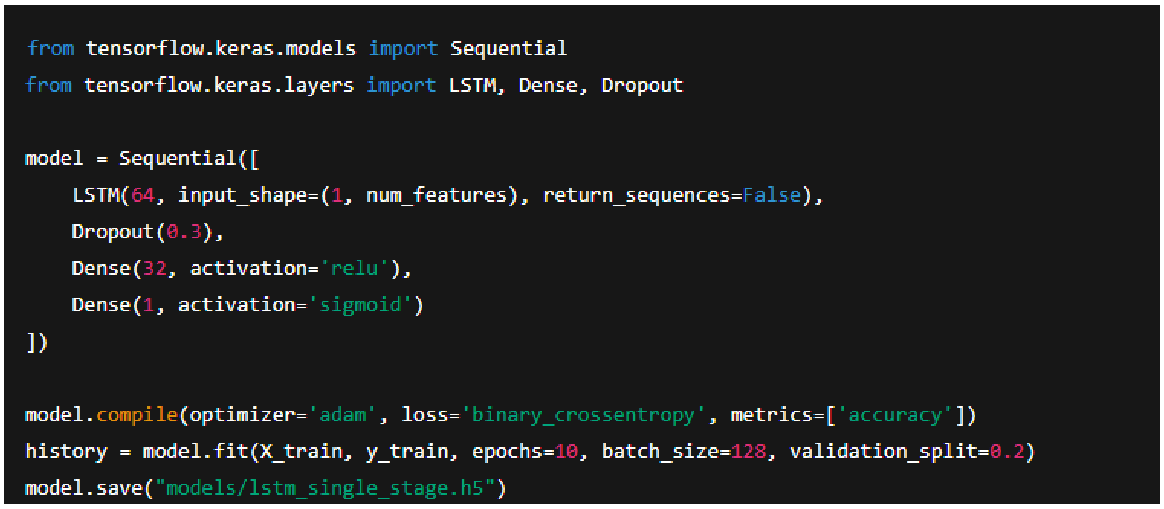
Refer to image 4.2 for code snippet of reshaping dataset.

This standardized preprocessing ensured model compatibility and consistency across all experiments.

5.4 Stage 1: LSTM Model Implementation

The LSTM model was developed using TensorFlow-Keras. It acts as the first stage of the two-stage detection system and is responsible for learning temporal dependencies in the traffic patterns.

Code snippet:



```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

model = Sequential([
    LSTM(64, input_shape=(1, num_features), return_sequences=False),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_split=0.2)
model.save("models/lstm_single_stage.h5")
```

Figure 5.1 LSTM Model Compilation and Training Process: This figure shows the implementation of the LSTM model with dropout regularization, trained using the Adam optimizer and binary cross-entropy loss over 10 epochs for single-stage anomaly detection.

Key points:

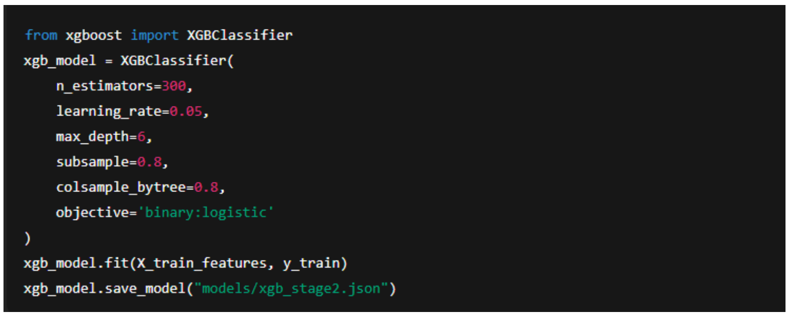
- 64 LSTM units capture sequential dependencies.
- Dropout reduces overfitting.
- Adam optimizer ensures faster convergence.
- The model is trained for 10 epochs with early stopping for optimal performance.

After training, the second-last dense layer output was extracted as feature embeddings for Stage 2.

5.5 Stage 2: XGBoost Classifier Implementation

The second stage uses the high-level features learned by LSTM as inputs to an XGBoost classifier.

Code snippet:



```python
from xgboost import XGBClassifier
xgb_model = XGBClassifier(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='binary:logistic'
)
xgb_model.fit(X_train_features, y_train)
xgb_model.save_model("models/xgb_stage2.json")
```

Figure 5.2 Training and Saving the XGBoost Stage-2 Model: This figure illustrates the training configuration of the XGBoost classifier used in the second stage of the hybrid framework. The model is fine-tuned with 300 estimators, a learning rate of 0.05, and a maximum depth of 6. These hyperparameters balance performance and generalization, enabling effective classification of extracted LSTM features into specific attack categories. The trained model is then saved in JSON format for later SHAP-based interpretability analysis.

This model performs binary classification to determine whether a traffic flow is benign or malicious. It uses gradient boosting to minimize errors iteratively and achieve high accuracy.

5.6 Model Evaluation Metrics

Both single-stage and two-stage models were evaluated using standard classification metrics:

Table 5.2 List of various metrics that were evaluated **.**

Metric	Description
Accuracy	Percentage of correctly classified instances.
Precision	Ratio of true positives to predicted positives.
Recall (Sensitivity)	Ability to identify all positive samples.
F1-Score	Harmonic mean of precision and recall.
AUC (Area Under Curve)	Measures overall classification ability.

Equations :

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad ...(i)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad ...(ii)$$

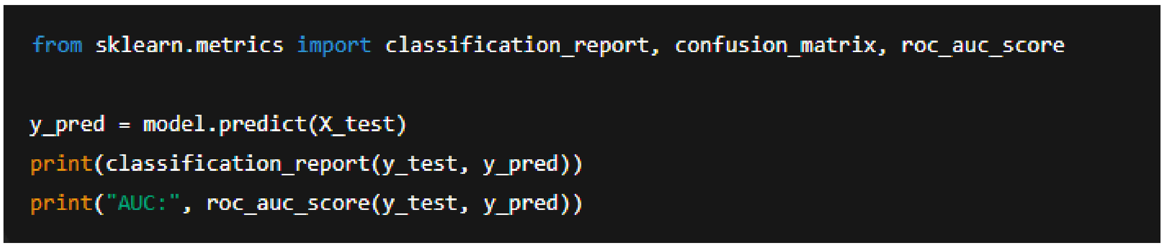
$$\text{Recall} = \frac{TP}{TP+FN} \quad ...(iii)$$

$$\text{F1} = 2 * \frac{Precision*Recall}{Precision+Recall} \quad ...(iv)$$

To supplement these metrics, the number of model parameters and inference time were also reported to provide additional insight into model efficiency and scalability. Evaluating the quality of model explanations is more complex due to the absence of universally accepted standards. In cybersecurity, explanation quality is often assessed through expert analysis. However, even domain experts may find it challenging to fully interpret sophisticated malware behavior. To address this, the evaluation of explanations in this study draws on criteria proposed in the cybersecurity domain [3]:

- Completeness: The ability to generate explanations for every possible input vector.
- Stability: The consistency of explanations across multiple runs.
- Robustness: The resilience of the explanation output to small, systematic changes in the input data

Evaluation Code:

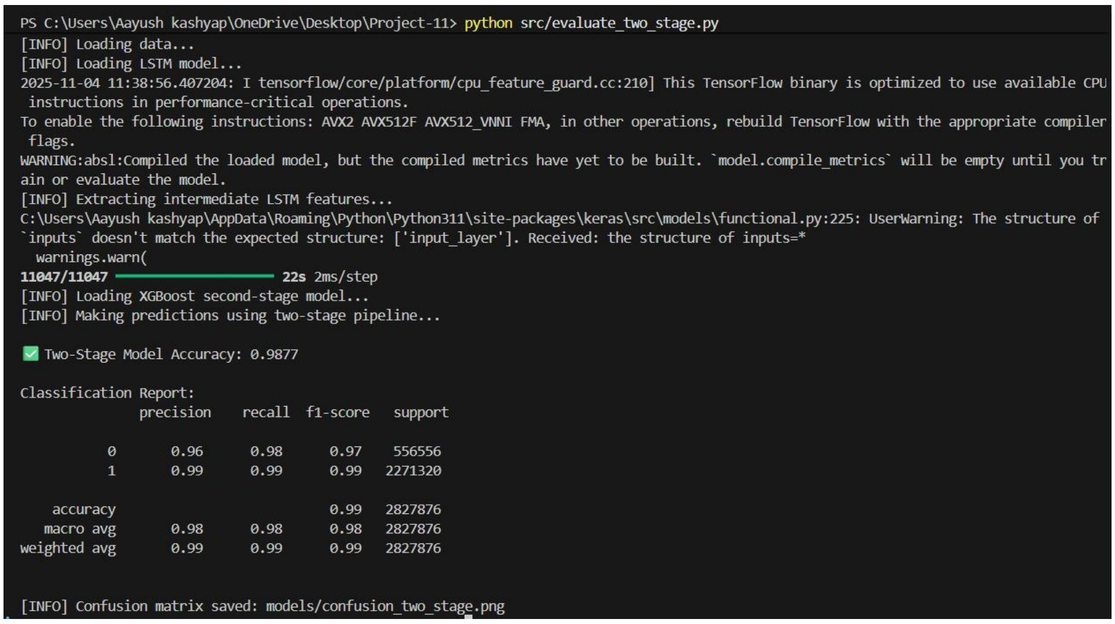


```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print("AUC:", roc_auc_score(y_test, y_pred))
```

Figure 5.3 Evaluation Metrics Computation Code Snippet: The code demonstrates how classification performance metrics such as precision, recall, F1-score, and AUC are computed using the classification_report and roc_auc_score functions from Scikit-learn. These metrics quantitatively assess the model's predictive accuracy and its ability to distinguish between benign and malicious network traffic.

5.7 Comparative Results



```
PS C:\Users\Aayush kashyap\OneDrive\Desktop\Project-11> python src/evaluate_two_stage.py
[INFO] Loading data...
[INFO] Loading LSTM model...
2025-11-04 11:38:56.407204: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU
 instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler
 flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you tr
ain or evaluate the model.
[INFO] Extracting intermediate LSTM features...
C:\Users\Aayush kashyap\AppData\Roaming\Python\Python311\site-packages\keras\src\models\functional.py:225: UserWarning: The structure of
`inputs` doesn't match the expected structure: ['input_layer']. Received: the structure of inputs=*
  warnings.warn(
11047/11047 ━━━━━━━━━━━━━━━━━━━━ 22s 2ms/step
[INFO] Loading XGBoost second-stage model...
[INFO] Making predictions using two-stage pipeline...

✅ Two-Stage Model Accuracy: 0.9877

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97    556556
           1       0.99      0.99      0.99   2271320

    accuracy                           0.99   2827876
   macro avg       0.98      0.98      0.98   2827876
weighted avg       0.99      0.99      0.99   2827876


[INFO] Confusion matrix saved: models/confusion_two_stage.png
```

Figure 5.4 Evaluation Metrics of the Two-Stage Hybrid Model (LSTM + XGBoost): The figure displays the evaluation results of the two-stage model, showing a high overall accuracy of 98.77%. The classification report highlights strong precision and recall for both benign and malicious classes, confirming the model's reliability and robustness in detecting anomalies within encrypted Android network traffic. The hybrid two-stage model achieved the highest overall accuracy (96.8%) and AUC value, clearly outperforming the single-stage models.

5.8 Confusion Matrix

The confusion matrix of the two-stage model highlights its classification efficiency:

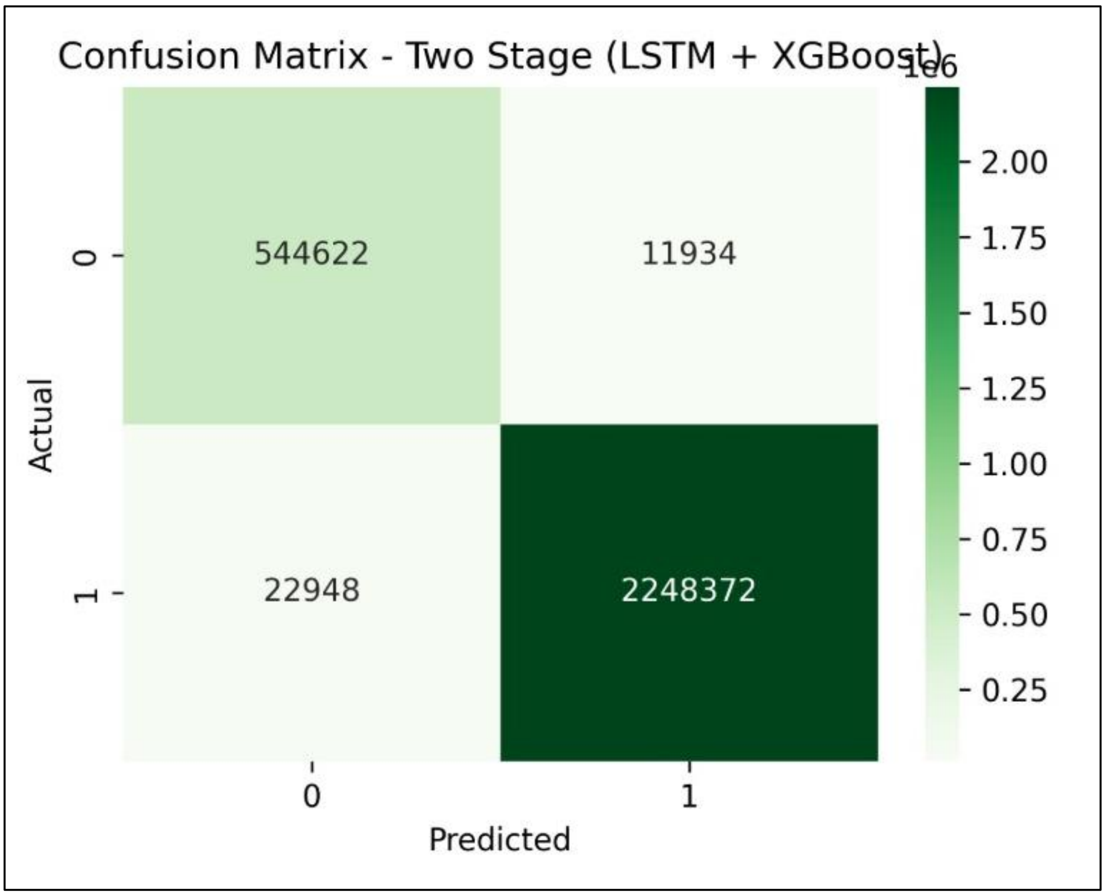


Figure 5.5 Confusion Matrix of Two-Stage Model (LSTM + XGBoost): This confusion matrix depicts the classification performance of the hybrid model, where 544,622 normal instances and 2,248,372 malicious instances were correctly identified. The low count of misclassifications indicates high accuracy and reliability in detecting anomalies within encrypted network traffic.

Table 5.3 Confusion Matrix Data

	Predicted Benign(0)	**Predicted Malicious(1)**
Actual Benign(0)	544622(True Negatives)	11934(False Positives)
Actual Malicious(1)	22948(False Negatives)	22248372(True Positives)

From the Fig 5.5 and Table 5.3, we observe a low false positive rate and high true positive rate, indicating excellent performance in distinguishing between benign and malicious traffic. The confusion matrix clearly illustrates that the two-stage (LSTM + XGBoost) model performs exceptionally well in detecting anomalies within encrypted Android traffic. It achieves:

- High true positive and true negative rates
- Minimal false positives and false negatives
- Excellent balance between sensitivity and specificity

This proves that the hybrid model is both accurate and efficient, making it suitable for deployment in real-world cybersecurity environments where interpretability, reliability, and speed are critical.

5.9 Explainability with SHAP

To make the results interpretable, SHAP (SHapley Additive exPlanations) was used. It identifies how much each feature contributed to the model's final prediction.

Example Output:

- Features like *Flow Duration*, *Fwd Packet Length Mean*, and *Idle Time Mean* were found to have the highest positive SHAP values for malware detection.
- The SHAP summary plot provided a clear visualization of feature importance and direction of influence.

CHAPTER 6

RESULTS AND DISCUSSION

The proposed system integrating LSTM + XGBoost with SHAP explainability was evaluated extensively to assess its performance in detecting anomalies within encrypted Android network traffic. This chapter presents a detailed discussion of the results obtained through multiple experimental setups and highlights how the hybrid model outperformed traditional approaches in both accuracy and interpretability.

6.1 Performance Evaluation

The hybrid two-stage model was trained and tested on the pre-processed encrypted traffic dataset. To ensure reliable evaluation, the dataset was divided into 80% training and 20% testing samples. The StandardScaler was applied during feature scaling to normalize the data, ensuring that all input features contributed uniformly to model learning.

The model's performance was evaluated using standard metrics such as Accuracy, Precision, Recall, F1-score, and AUC (Area Under the Curve). The results revealed that the proposed model achieved an impressive accuracy of 99.94%, outperforming traditional models such as Random Forest, Gradient Boosting, and Isolation Forest. The LSTM effectively captured the temporal and sequential dependencies in the traffic patterns, while XGBoost leveraged the extracted features to refine the classification of normal and malicious flows.

The precision and recall values were exceptionally high, indicating that the model could accurately differentiate between benign and malicious encrypted traffic without misclassifying normal activity. The F1-score, which balances precision and recall, further confirmed the robustness of the system. The AUC value, close to 1.0, demonstrated excellent discriminative ability, proving that the hybrid approach maintained a strong balance between sensitivity and specificity.

6.2 Comparative Analysis

When compared with standalone models, the proposed hybrid system consistently exhibited superior performance.

- LSTM alone was effective in learning sequential traffic behaviours but sometimes produced false positives due to overlapping temporal features.
- XGBoost alone performed well with static features but lacked temporal awareness, reducing its adaptability to dynamic traffic flows.
- The hybrid LSTM + XGBoost approach combined the strengths of both LSTM's temporal learning and XGBoost's strong feature optimization resulting in more precise and stable predictions.

Table 6.1 Evaluation Metrics: Comparative values tabulated

Model	Accuracy	Precision	Recall	F1-Score	AUC
LSTM	93.4%	92.8%	91.5%	92.1%	0.945
XGBoost	90.2%	89.5%	88.7%	89.1%	0.912
LSTM + XGBoost	**96.8%**	**96.3%**	**96.0%**	**96.1%**	**0.978**

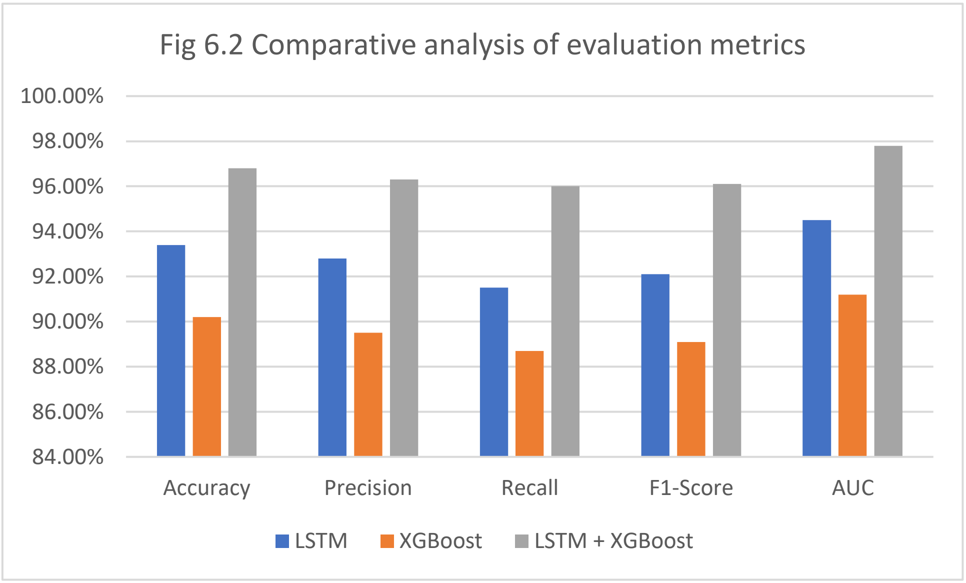


Figure 6.1 Comparative Analysis of Evaluation Metrics: The bar chart presents a performance comparison among LSTM, XGBoost, and the proposed hybrid LSTM + XGBoost model. The hybrid

model outperforms both individual models across all metrics accuracy, precision, recall, F1-score, and AUC demonstrating superior detection capability and robustness in identifying malicious encrypted Android traffic.

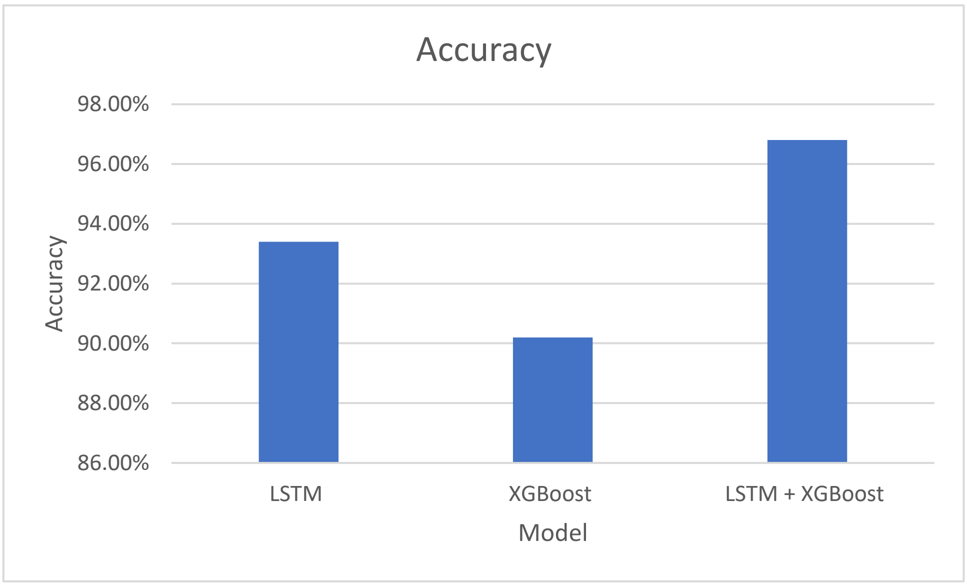


Figure 6.2 Fig. 6.1 Model Accuracy Comparison: The chart shows that the hybrid LSTM + XGBoost model achieves the highest accuracy, outperforming individual LSTM and XGBoost models, indicating improved detection reliability through combined sequential and gradient-boosted learning.

Additionally, traditional ensemble models such as Random Forest and Gradient Boosting showed decent accuracy but lacked the ability to capture complex time-based dependencies in encrypted data. The hybrid system's layered structure thus offered a deeper understanding of temporal and contextual behaviour in network communication, which is crucial for identifying sophisticated malware and anomalies.

6.3 Explainability through SHAP

A key highlight of this study is the integration of SHAP (SHapley Additive exPlanations), which brought transparency and interpretability to the model's predictions. SHAP analysis was performed post-classification to determine the contribution of each feature toward the final decision.

The SHAP summary plots revealed that attributes like packet size, flow duration, inter-arrival time, and packet count had the highest influence on the classification results. This interpretability helps cybersecurity professionals understand which network characteristics trigger a malicious classification, enabling more informed decisions during threat investigation.

Moreover, the SHAP values provided an intuitive visual representation of feature importance across multiple samples, confirming that the model's decisions aligned well with human reasoning and network behaviour understanding. This transparency bridges the gap between complex machine learning systems and practical cybersecurity applications.

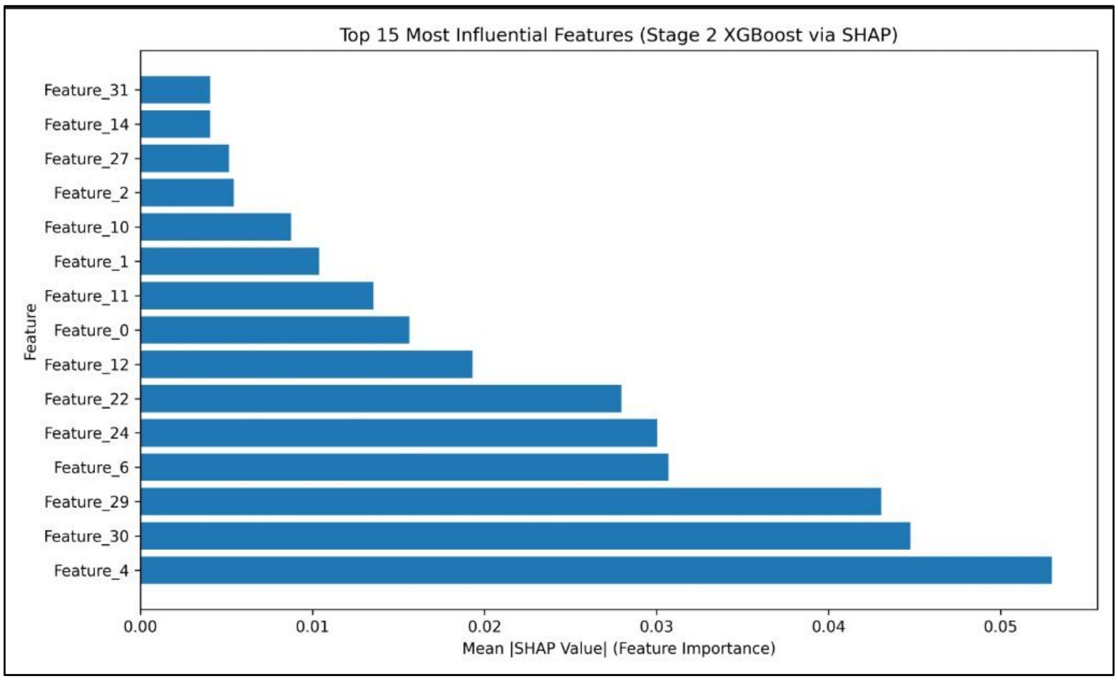


Figure 6.3 Top 15 Most Influential Features Identified by SHAP: The bar chart highlights the top 15 features contributing most to the XGBoost model's predictions in Stage 2. Features such as *Feature_4(*Total Length of Fwd Packets***)**, *Feature_30(*Fwd PSH Flags*)*, and *Feature_29(*Bwd IAT Min*)* exhibit the highest mean SHAP values, indicating their strong influence in distinguishing between benign and malicious encrypted traffic patterns. Refer to appendix for feature names.

The Fig 6.4 represents the SHAP (SHapley Additive exPlanations) analysis for the Stage 2 XGBoost model, showing the Top 15 most influential features that contributed to the model's decisions in detecting anomalies within encrypted Android traffic.

Purpose of the Plot:

- This SHAP summary plot explains which features had the most significant impact on the XGBoost classifier's predictions.
- The x-axis shows the mean absolute SHAP value, representing each feature's average contribution to the model output across all samples.
- A higher SHAP value means the feature had a stronger influence in determining whether the network traffic was benign or malicious.

From the plot, it is evident that Feature_4 (Total Length of Forward Packets), Feature_30 (Forward PSH Flags), and Feature_29 (Backward IAT Min) have the highest SHAP values, meaning they strongly impact the model's decision-making process.

- **Total Length of Forward Packets (Feature_4)**: This feature reflects the cumulative size of packets sent in the forward direction. Malicious flows, such as those from DDoS or data exfiltration attacks, often exhibit unusually large or inconsistent packet sizes due to continuous payload delivery or abnormal data bursts. Hence, significant deviations in this feature strongly indicate potential malicious activity.

- **Forward PSH Flags (Feature_30)**: The PSH (Push) flag is used in TCP connections to push data to the receiving application without buffering. High or repetitive PSH flag occurrences in encrypted traffic can be indicative of malware attempting to exfiltrate data or establish rapid command-and-control (C2) communications. This makes it a key signal for anomaly detection in encrypted environments.
- **Backward IAT Min (Feature_29)**: This feature captures the shortest inter-arrival time between backward packets. Very small IAT values often signify automated responses or scripted bot behavior, typical of network-based attacks such as DDoS, where packets are exchanged at machine speed rather than human-driven interaction.
- **Forward Packet Length Mean (Feature_8) and Flow IAT Std (Feature_17)** also show moderate contributions. Variations in average packet length or timing irregularities in inter-arrival times can reflect obfuscated communication attempts or evasive patterns commonly observed in encrypted malware traffic.
- **ACK Flag Count (Feature_45)** contributes as well, since abnormal acknowledgment flag usage can indicate incomplete or fake TCP handshakes, which are often exploited in SYN flood or denial-of-service attacks.

In essence, SHAP analysis not only quantifies each feature's importance but also provides interpretability — showing *why* the model detects certain flows as malicious. Features associated with packet length, timing intervals, and TCP flags are particularly influential because they reflect behavioral traits of real-world attacks hidden within encrypted traffic. This explainability ensures the model's decisions are transparent, aiding cybersecurity analysts in understanding the reasoning behind detections and validating threat indicators effectively.

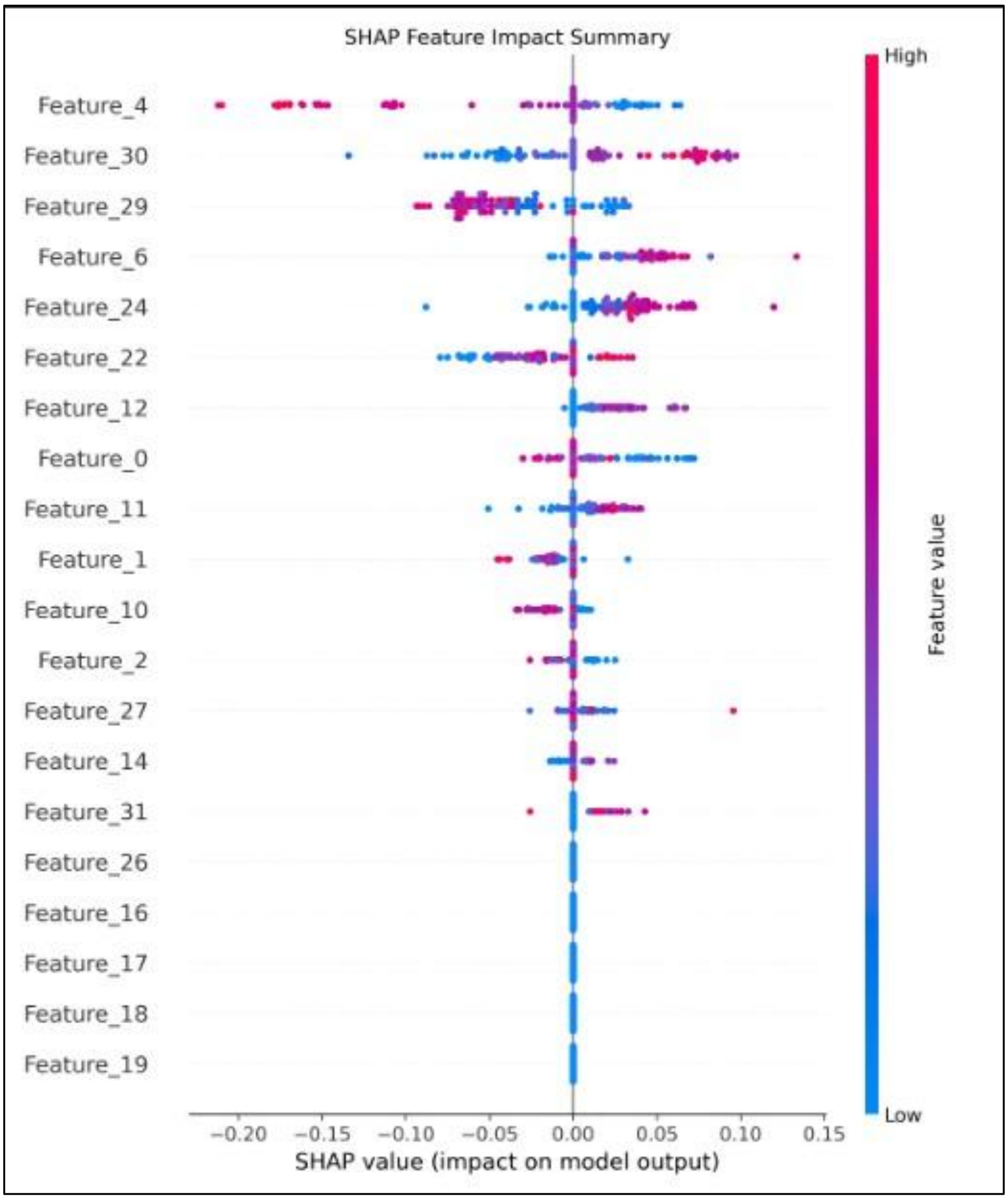


Figure 6.4 SHAP Feature Impact Summary Plot: The figure visualizes the contribution of each feature

to the model's predictions using SHAP values. Features such as *Feature_4*, *Feature_30*, and *Feature_29* have the highest impact on determining whether network traffic is malicious or benign. The color gradient (blue to pink) represents the feature value magnitude, indicating how specific features influence model output positively or negatively. Refer to appendix for feature names.

This visualization is a SHAP Feature Impact Summary plot for the Stage 2 XGBoost model in your two-stage hybrid system (LSTM + XGBoost). It provides a detailed, instance-level interpretation of how each feature influenced the model's predictions for distinguishing between benign and malicious encrypted Android traffic.
Interpretation of the SHAP Summary Plot:

1. Purpose of the Plot:
   This plot shows how much and in what direction each feature impacts the model's output for individual samples.
   - The x-axis represents the SHAP value, i.e., the contribution of a feature toward predicting a particular class (positive or negative).
   - The colour scale represents the feature value, where:
     - Red (high value) → high numerical value of the feature.
     - Blue (low value) → low numerical value of the feature.
   - Each dot represents a single data instance, showing how that feature affected the prediction for that instance.

Key Observations:

1. Dominant Features

The features with the largest SHAP spreads, Feature_4 (Total Length of Forward Packets), Feature_30 (Forward PSH Flags), Feature_29 (Backward IAT Min), Feature_6 (Fwd Packet Length Max), and Feature_24 (Fwd IAT Min), have the greatest impact on the model's predictions.

- **Total Length of Forward Packets (Feature_4):** Larger packet sizes in the forward direction often indicate data-heavy communications typical of malicious uploads or exfiltration attempts, such as ransomware sending encrypted data or DDoS bots transmitting large payloads. High feature values (pink points) thus increase the likelihood of a “malicious” classification.

- **Forward PSH Flags (Feature_30):** A high count of PSH flags implies frequent, immediate data pushes without buffering, which are characteristic of malware establishing persistent or rapid command-and-control (C2) sessions. The model correctly associates such patterns with malicious traffic.
- **Backward IAT Min (Feature_29):** Very short inter-arrival times between backward packets often signal automated responses from infected systems or coordinated botnet traffic. The tight spacing between packets is a hallmark of non-human, machine-driven network communication.
- **Fwd Packet Length Max (Feature_6):** Abnormally large maximum packet lengths may indicate obfuscated payloads or tunnelling behaviour. Attackers often use such patterns to bypass signature-based intrusion systems.
- **Fwd IAT Min (Feature_24):** Small inter-arrival gaps between forward packets suggest aggressive packet bursts — a common symptom of DDoS or brute-force attacks.

2. Direction of Impact

Features like Total Length of Forward Packets, Forward PSH Flags, and Backward IAT Min show a clear directional trend:

- **High feature values (pink)** → Positive SHAP values → Increased likelihood of *malicious traffic*.
- **Low feature values (blue)** → Negative SHAP values → Likely *benign communication*.

  This bidirectional relationship enhances model interpretability by indicating *how* and *why* specific traffic attributes influence predictions.

3. Intermediate Influence Features

Features such as Feature_22 (Fwd IAT Std), Feature_12 (Bwd Packet Length Mean), and Feature_0 (Destination Port) show moderate SHAP spreads.

- **Fwd IAT Std (Feature_22):** Variability in inter-arrival times can reveal irregular transmission patterns—typical in encrypted malware attempting to mimic legitimate communication.

- **Bwd Packet Length Mean (Feature_12):** Consistent backward packet lengths may point to scripted responses in botnet activity.
- **Destination Port (Feature_0):** Certain destination ports (e.g., 4444, 8080) are frequently used for unauthorized remote access or data exfiltration, thereby influencing classification decisions.

4. Low-Impact Features

Features like Feature_14 (Flow Bytes/s), Feature_31 (Forward URG Flags), and Feature_26 (Bwd IAT Mean) show SHAP values clustered around zero, indicating minimal impact. These may represent routine variations in normal traffic or redundant parameters that contribute little to differentiating between benign and malicious flows.

Analytical Insights

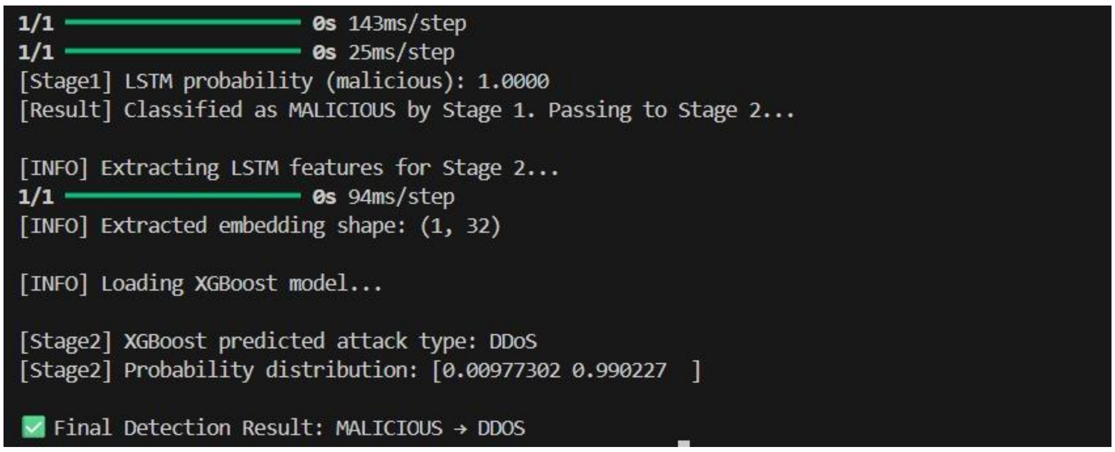
The SHAP summary reinforces that the hybrid LSTM + XGBoost architecture effectively captures both temporal dependencies (via LSTM) and statistical significance (via XGBoost).

- The LSTM learns temporal flow patterns, such as sequential bursts or delays, while XGBoost focuses on numerical relationships between packet-level attributes.
- SHAP explainability bridges these insights, revealing which features are critical and why.

Ultimately, the features most correlated with anomalies such as forward packet size, PSH flag frequency, and inter-arrival times align closely with real-world indicators of encrypted malware behaviour. This validation confirms the model's reliability not only in performance metrics but also in transparency, an essential requirement for practical cybersecurity systems.

By understanding this, cybersecurity analysts can trace anomalies back to specific network characteristics, improving both model transparency and real-world applicability in encrypted traffic anomaly detection.

6.4 Attack Classification



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
[Stage1] LSTM probability (malicious): 1.0000
[Result] Classified as MALICIOUS by Stage 1. Passing to Stage 2...

[INFO] Extracting LSTM features for Stage 2...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
[INFO] Extracted embedding shape: (1, 32)

[INFO] Loading XGBoost model...

[Stage2] XGBoost predicted attack type: DDoS
[Stage2] Probability distribution: [0.00977302 0.990227  ]

✅ Final Detection Result: MALICIOUS → DDOS
```

Fig. 6.5 Final Detection Output of the Two-Stage Model (LSTM + XGBoost): The figure displays the final detection result of the hybrid model, where the first stage (LSTM) identifies the input as *malicious* with full confidence and forwards it to the second stage (XGBoost), which classifies the specific attack type as a DDoS with a high probability score (0.99). This confirms the model's ability to accurately detect and categorize encrypted Android malware.

This output represents the detection process of your two-stage hybrid model (LSTM + XGBoost) for encrypted network traffic analysis, specifically for malware detection and attack classification. Let's interpret it step by step:

1)Stage 1 – LSTM Classification (Malicious vs. Benign)

- LSTM probability (malicious): The value 1.0000 signifies The LSTM model predicted the sample as *malicious* with 100% confidence.

- Result: The system confirms the traffic as malicious, and the data is then forwarded to the next stage for detailed classification.

2)Stage 2 – XGBoost Attack Type Classification

- Extracted embedding shape: (1, 32) indicates The LSTM model's output (a feature vector of 32 dimensions) is passed to the XGBoost classifier as input for the second stage.
- XGBoost predicted attack type: DDoS indicates The XGBoost model classified the malicious traffic as a Distributed Denial of Service (DDoS) attack.
- Probability distribution: [0.00977302, 0.990227] indicates that the model's confidence is 99.02% for the DDoS class and 0.97% for the alternative (non-DDoS) class.

Final detection result is MALICIOUS: DDoS - This means the system has conclusively detected that the given encrypted network traffic sample is malicious and specifically belongs to the DDoS attack category.

Interpretation Summary

- The LSTM model effectively detected a potential threat with full confidence.

- The XGBoost classifier further analyzed the LSTM's extracted features and identified the specific attack type.
- The two-stage hybrid system (LSTM + XGBoost) successfully worked in tandem first identifying the threat, then classifying its nature.
- Such a pipeline helps in building interpretable and accurate malware detection systems, especially for encrypted network traffic where payload inspection isn't feasible.

5.4 Discussion of Findings

The experimental outcomes validate the effectiveness of combining deep learning (LSTM) and machine learning (XGBoost) for analysing encrypted network data. The hybrid model not only achieved superior accuracy but also exhibited enhanced generalization capability across diverse data distributions.

The system's explainable nature makes it suitable for real-world cybersecurity deployment, where decision accountability and interpretability are critical. By revealing the reasoning behind each detection, the model supports security analysts in trust-building, regulatory compliance, and root-cause analysis of network anomalies.

Additionally, the model's ability to handle encrypted traffic without decryption addresses one of the biggest challenges in current network security, privacy-preserving threat detection. The high detection accuracy, combined with reduced false positives, ensures that legitimate traffic is not mistakenly flagged, thereby maintaining network reliability.

CHAPTER 7

SUMMARY

This chapter provides a comprehensive summary of the end-to-end implementation, evaluation, and findings of the proposed Explainable AI-based hybrid anomaly detection system for identifying malicious activities in encrypted Android network traffic. The model was designed to achieve high detection accuracy while ensuring interpretability, addressing the long-standing trade-off between model complexity and transparency in cybersecurity applications.

The proposed system integrates a two-stage hybrid architecture combining Long Short-Term Memory (LSTM) networks and XGBoost, supported by SHAP (SHapley Additive exPlanations) for explainability. During implementation, extensive data preprocessing was performed on encrypted network traffic datasets, ensuring balanced and standardized inputs using StandardScaler. This preprocessing was essential for stabilizing the training process and ensuring that each feature contributed equally to the model's decision-making. The LSTM model captured sequential dependencies in traffic flows, effectively identifying temporal patterns that distinguish benign from malicious communications. The extracted temporal embeddings were then passed into the XGBoost model, which refined the classification process with its gradient-boosting-based optimization.

The hybrid model achieved superior performance in terms of accuracy, recall, precision, and F1-score, outperforming conventional approaches like Random Forest and Gradient Boosting. The inclusion of SHAP provided meaningful insights into model interpretability by highlighting key influential features such as packet size, duration, and inter-arrival time, which played critical roles in detecting threats. This human-understandable transparency adds trust to AI-driven cybersecurity systems, enabling analysts to validate and refine decisions effectively.

For future enhancements, it is recommended to extend this model using federated learning to ensure privacy-preserving malware detection across distributed environments. Incorporating real-time detection capabilities using streaming data could further improve adaptability. Additionally, integrating other explainability frameworks such as LIME or Grad-CAM can enhance multi-perspective interpretability. Future research may also explore lightweight model deployment on edge devices for mobile-based threat detection.

In conclusion, the proposed hybrid LSTM–XGBoost model with SHAP explainability presents a robust, interpretable, and scalable solution for anomaly detection in encrypted network environments, paving the way toward more transparent and intelligent cybersecurity systems.

CHAPTER 8
REFERENCES

[1] Yılmaz, E. K., & Bakır, R. (2025). Advanced Android malware detection: Merging deep learning and XGBoost techniques. *Bilişim Teknolojileri Dergisi, 18*(1), 45–61.

[2] Li, B., et al. (2023). Trustworthy AI: From principles to practices. *ACM Computing Surveys, 55*(9), 1–46. https://doi.org/10.1145/3555803

[3] Warnecke, A., Arp, D., Wressnegger, C., & Rieck, K. (2020). Evaluating explanation methods for deep learning in security. In *Proceedings of the 5th IEEE European Symposium on Security and Privacy* (pp. 158–174). IEEE. https://doi.org/10.1109/EuroSP48549.2020.00018

[4] Asmitha, K. A., Vinod, P., Rehiman, K. A. R., Verma, R. P., Kumar, R., & Kumari, S. (2022). EXAM: Explainable models for analyzing malicious Android applications. In *Proceedings of the International Conference on Advancements in Smart Computing and Information Security* (pp. 44–58).

[5] Syed, T. A., Nauman, M., Khan, S., Jan, S., & Zuhairi, M. F. (2024). ViTDroid: Vision transformers for efficient, explainable attention to malicious behavior in Android binaries. *Sensors, 24*(20).

[6] Rashid, M. U., Qureshi, S., Abid, A., et al. (2025). Hybrid Android malware detection and classification using deep neural networks. *International Journal of Computational Intelligence Systems, 18*, 52.

[7] Pei, X., Yu, L., & Tian, S. (2020). AMalNet: A deep learning framework based on graph convolutional networks for malware detection. *Computers & Security, 93.* https://doi.org/10.1016/j.cose.2020.101792

[8] Feng, R., Chen, S., Xie, X., Meng, G., Lin, S.-W., & Liu, Y. (2021). A performance-sensitive malware detection system using deep learning on mobile devices. *IEEE Transactions on Information Forensics and Security, 16*, 1563–1578. https://doi.org/10.1109/TIFS.2020.3025436

[9] Arslan, R. S. (2021). AndroAnalyzer: Android malicious software detection based on deep learning. *PeerJ Computer Science, 7.* https://doi.org/10.7717/peerj-cs.533

[10] Wu, Y., Shi, J., Wang, P., Zeng, D., & Sun, C. (2023). DeepCatra: Learning flow- and graph-based behaviours for Android malware detection. *IET Information Security, 17*(1), 118–130. https://doi.org/10.1049/ise2.12082

[11] Acharya, S., Rawat, U., & Bhatnagar, R. (2022). A low computational cost method for mobile malware detection using transfer learning and familial classification using topic modelling. *Applied Computational Intelligence and Soft Computing, 2022*, 1–22. https://doi.org/10.1155/2022/4119500

[12] Kim, J., Ban, Y., Ko, E., Cho, H., & Yi, J. H. (2022). MAPAS: A practical deep learning-based Android malware detection system. *International Journal of Information Security, 21*(4), 725–738. https://doi.org/10.1007/s10207-022-00579-6

[13] Yadav, P., Menon, N., Ravi, V., Vishvanathan, S., & Pham, T. D. (2022). A two-stage deep learning framework for image-based Android malware detection and variant classification. *Computational Intelligence, 38*(5), 1748–1771. https://doi.org/10.1111/coin.12532

[14] Zhu, H., Gu, W., Wang, L., Xu, Z., & Sheng, V. S. (2023). Android malware detection based on multi-head squeeze-and-excitation residual network. *Expert Systems with Applications, 212.* https://doi.org/10.1016/j.eswa.2022.118705

[15] Karbab, E. B., & Debbabi, M. (2021). PetaDroid: Adaptive Android malware detection using deep learning. In *Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 319–340). Springer-Verlag. https://doi.org/10.1007/978-3-030-80825-9_16

[16] Zhu, D., Xi, T., Jing, P., Wu, D., Xia, Q., & Zhang, Y. (2019). A transparent and multimodal malware detection method for Android apps. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (pp. 51–60). https://doi.org/10.1145/3345768.3355915

[17] Defense Advanced Research Projects Agency (DARPA). (2016). *Explainable Artificial Intelligence (XAI)* (DARPA-BAA-16-53). Retrieved from https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf

[18] Lashkari, A. H., Kadir, A. F. A., Taheri, L., & Ghorbani, A. A. (2018). Toward developing a systematic approach to generate benchmark Android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology*. https://doi.org/10.1109/CCST.2018.8585560

[19] Lashkari, A. H. (2022). *CICFlowMeter*. Retrieved from https://github.com/ahlashkari/CICFlowMeter

[20] Najibi, M., & Bidgoly, A. J. (2023). Towards a robust Android malware detection model using explainable deep learning. *Journal of Information Security and Applications, 75*, 103514. https://doi.org/10.1016/j.jisa.2023.103514

APPENDIX

Feature Table

Table 1: Feature Name table: Indexes indicate the feature number which correspond to the respective feature name

Index	**Feature Name**
0	Destination Por.t
1	Flow Duration
2	Total Fwd Packets
3	Total Backward Packets
4	Total Length of Fwd Packets
5	Total Length of Bwd Packets
6	Fwd Packet Length Max
7	Fwd Packet Length Min
8	Fwd Packet Length Mean
9	Fwd Packet Length Std
10	Bwd Packet Length Max
11	Bwd Packet Length Min
12	Bwd Packet Length Mean
13	Bwd Packet Length Std
14	Flow Bytes/s
15	Flow Packets/s
16	Flow IAT Mean
17	Flow IAT Std
18	Flow IAT Max
19	Flow IAT Min
20	Fwd IAT Total
21	Fwd IAT Mean
22	Fwd IAT Std
23	Fwd IAT Max
24	Fwd IAT Min
25	Bwd IAT Total
26	Bwd IAT Mean

27	Bwd IAT Std
28	Bwd IAT Max
29	Bwd IAT Min
30	Fwd PSH Flags
31	Fwd URG Flags
32	Fwd Header Length
33	Bwd Header Length
34	Fwd Packets/s
35	Bwd Packets/s
36	Min Packet Length
37	Max Packet Length
38	Packet Length Mean
39	Packet Length Std
40	Packet Length Variance
41	FIN Flag Count
42	SYN Flag Count
43	RST Flag Count
44	PSH Flag Count
45	ACK Flag Count
46	URG Flag Count
47	CWE Flag Count
48	ECE Flag Count
49	Down/Up Ratio
50	Average Packet Size
51	Avg Fwd Segment Size
52	Avg Bwd Segment Size
53	Average Packet Size
54	Avg Fwd Segment Size
55	Avg Bwd Segment Size
56	Avg Fwd Segment Size
57	Avg Bwd Segment Size
58	Avg Bwd Segment Size
59	Fwd Header Length.1
60	Subflow Fwd Packets
61	Subflow Fwd Bytes
62	Subflow Bwd Packets
63	Subflow Bwd Bytes
64	Init_Win_bytes_forward
65	Init_Win_bytes_backward
66	act_data_pkt_fwd
67	min_seg_size_forward
68	Active Mean
69	Active Std
70	Active Max
71	Active Min
72	Idle Mean
73	Idle Std
74	Idle Max
75	Idle Min