Android Malware Detection using Network Traffic

Noureldine Adib, Hani Ragab Hassen, Ali Muzaffar, Michael A. Lones

***Abstract*—This study investigates and reviews network traffic-based Android malware detection, focusing on analysing various network traffic representations used in training machine learning models. We re-implement and compare various combinations of the different network traffic representations, including image and text representations in addition to the traditional matrix/table representation, with different types of machine learning models to determine which models work best with particular Android network representations. Our work makes a substantial contribution by using a balanced, up-to-date dataset of Android malware and benign application traffic, which is currently the largest publicly available dataset for Android malware detection. We identify the best-performing feature representations, feature selection algorithms, and models for Android malware detection.**

***Index Terms*—Android malware, Network traffic captures, Network Traffic Representation, Machine Learning**

CONTENTS

I. INTRODUCTION

ANDROID is an open-source mobile operating System (OS) developed by Google [8]. As of 2022, Android holds a dominant share of the mobile OS market with 69.64%, surpassing that of other platforms such as iOS (29.9%) and Samsung (0.37%) [9]. This vast array of Android devices and users has sparked an ecosystem of applications in Android stores, with users downloading an estimated 26.9 billion apps from the Google Play Store in the first quarter of 2023 [10]. Users spent a further estimated 167 billion US dollars on Android apps in 2022 [11].

However, this immense popularity also brings challenges, particularly in security. Malware threats cover a broad range, all the way from irritating adware popups to sophisticated security risks that are capable of data breaches and financial theft. In 2022, an estimated 5.5 billion malware attacks were detected worldwide [12], resulting in billions of US dollars in loss [13]. According to a report by Kaspersky, an estimated 87% to 95% of Android phones have at least one critical vulnerability on them [14].

The need for a robust and effective solution to safeguard users in light of the risks and threats posed by malware is more critical than ever. The rapid expansion of the Android ecosystem is closely coupled with the expansion of malware

threats and attack frequency, making traditional security measures incapable of handling it alone. To fill this gap, many researchers have proposed solutions based around machine learning (ML). Although promising, there are a number of issues which limit practical deployment [15], [16].

One of the surprising observations from the literature is that features derived specifically from network traffic often lead to the most accurate ML models for malware detection. An important question we seek to answer in this paper is *why* network features are effective in supporting the training of ML models for Android malware detection, since it is not immediately clear that this should be the case. We approach this by reviewing and reimplementing previous studies that built ML models from network features. In each case, we reevaluate within a contemporary malware environment, and go beyond many of the original studies by attempting to understand how each ML model uses network features to reach a classification decision. Specifically:

- We review previous approaches to ML-based Android malware detection using network traffic features.
- We curate and share[1] a balanced, up-to-date dataset of Android malware and benign application traffic.
- Using this dataset, we re-implement and reevaluate past Android anti-malware approaches that use different network traffic representations.
- We explore combinations of network traffic representations and ML models to determine the most effective approaches.
- We report the best-performing feature representations, feature selection techniques and models.
- Through explainable AI techniques, we investigate feature usage within the ML models.

Most significantly, our results show that network features are often quite brittle. That is, although they appear to support the training of accurate models, in practice these models have a tendency to overfit their training data, and may not generalise to realistic usage situations.

This paper is organised as follows. (add text here!)

II. Background Concepts

Before critically reviewing the literature on network traffic representation and classification, we begin with a brief introduction to the underlying network traffic architecture and the nature and types of malware.

A. Network Traffic Architecture

Understanding the architecture of network traffic is essential for detecting and classifying malware. This section provides an overview of the principal components and structures of network traffic and how they relate to malware detection.

Packets are the basic units of data exchange on the network. Each packet can be characterized by five main attributes: source Internet Protocol (IP) address, destination IP address, source port, destination port, and protocol [5]. Each packet contains a header, comprising meta-data and routing information, and a payload, i.e., the data it transmits over the network.

The TCP/IP network architecture model comprises four distinct layers that form the backbone of data communication over a network [17], [18]. These are:

Application The application layer defines the network applications [17]. This layer encompasses protocols such as Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), and Domain Name System (DNS). The purpose of this layer is to build something usable for applications on top of the underlying network services.

Transport The transport layer is responsible for ensuring the end-to-end, i.e., process-to-process, exchange of network traffic [18]. This layer uses protocols such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a connection-oriented protocol that establishes a connection between a client and a server. This connection allows for reliable and error-checked packet delivery. TCP is used for applications that require reliability, such as file transfer. UDP provides a connection-less model with no error-checking of packet delivery. UDP is used for applications that require speed, such as video streaming.

Internet The internet layer is responsible for routing packets [18]. IP is responsible for determining the path the packets take in the network to allow for host-to-host communication. ARP is responsible for connecting IP addresses to physical addresses, often called Media Access Control (MAC) addresses in a Local-Area Network (LAN). ICMP is responsible for reporting errors across the network, such as dropped packets or connection failures.

Data Link The data link layer is responsible for low-level one-hop data transfer and error handling [18].

B. Malware Categories and Families

Malicious software, or *malware*, refers to any software application or program intended to cause harm, damage, or make any unauthorized changes to a computer system, computer network or environment. There are several malware classifications, but common malware types include [19], [20]:

Viruses Self-replicating pieces of code that cause damage to the systems or environment. They typically require user interaction to activate.

Worms Similar to viruses, but do not require user interaction to activate.

Trojans Disguise themselves as legitimate software while secretly causing damage to the system or environment.

Ransomware Encrypts user files, preventing access to them until a ransom is paid.

Adware Spams users with unwanted advertisements.

Spyware Secretly collects user and/or system data in the background.

Rootkits Used to hide the existence of malicious code and files on the victim's host.

Keyloggers Records the keystrokes and/or mouse activity of users on the device.

[1]Upon acceptance.

ID	Description
1	Host
2	Request-URI
3	Request Method
4	User agent

TABLE I: HTTP features used by Wang et al. [1]

ID	Description
1	Number of bytes in uplink traffic flow
2	Number of bytes in downlink traffic flow
3	Number of upstream traffic flow packets
4	Number of downstream traffic flow packet
5	Average number of bytes in uplink traffic flow packets
6	Average number of bytes in downlink traffic flow packets

TABLE II: TCP features used by Wang et al. [1]

Mobile malware Malware specifically designed to infect mobile and tablet devices.

Bots Bots are automated software that performs tasks over a network. Malicious bots can create networks of bots called botnets.

Malware is further classified into families, where each family is based on a common codebase, attack vectors, and behaviors [21]. For instance, Zeus and Citadel are explicitly designed for finance-related attacks, with targets such as account information theft, while Stuxnet and Duqu target industrial control and supervision systems [21].

III. Previous Work

In this section, we review previous approaches to Android malware detection which used ML models based on network traffic. We selected papers based on the following criteria:

1) The approach entirely or partially relies on network traffic for detection.
2) The paper describes the implementation of a machine learning model.
3) The study was conducted within the last ten years.

The reviewed approaches can primarily be organised by representation, i.e., how the network traffic data is presented to the model. The main forms of representation are traditional matrix/tabular forms, text representations, and image representations. We review each of these in the following sections.

A. Matrix/Tabular Representation of Network Traffic

Matrix or tabular representations are the most widely used in the literature. These consist of numeric features which are extracted from network traffic, and which can be listed in a table. Each feature is predefined and has a fixed size. The benefit of this approach is its simplicity and its applicability to a broad range of ML models. It can also be useful from an analytical viewpoint since these features summarise well-understood properties of network traffic. However, by summarising the underlying traffic, this approach does risk losing information.

A seminal 2016 study which used this approach was that of Wang et al. [22]. Their detection model, TrafficAV, analyzed both HTTP and TCP traffic. The set of features for each is shown in tables I and II, respectively: there are 6 features for TCP, and 4 for HTTP. They used a C4.5 decision tree and the Drebin project dataset [23]. On this dataset—which was released in 2014, but remains in common use—they found that HTTP request models had a higher accuracy (99.65%) than TCP flow models (98.16%). In later work [1], they found that higher levels of detection could be achieved by combining HTTP and TCP features. A 2023 reimplementation by authors of this study [24] reevaluated this approach using a large contemporary dataset, largely corroboratating the findings of their original study with accuracies of 97.3% (TCP) and 96.3% (HTTP) using the same features. It is interesting that such a high level of accuracy can be achieved using such a small number of features.

Later studies by other authors have also shown the benefit of using traditional ML models with network-based features. Zulfili et al. reported accuracies of up to 98.4%, and Thangaveloo et al [25] showed that models with network features performed better than those without.

More recent studies have tended to focus on the use of deep learning models. One example of this is a 2022 study by Fallah and Bidgoly, who used LSTMs [2]. Their approach used flow-based network statistics. For each flow, 75 features were extracted, representing a broad range of statistical properties which are presented in Table III. They found that 50 flows ($\sim$1600 packets on average) per application led to an impressive accuracy of 99.96% on the (2017) CICAndMal2017 dataset, with no benefit to using longer flows. This is a significantly higher accuracy than those reported for traditional ML models; however, different datasets were used, so we can not draw any firm conclusions from this. The authors of this study also considered the problem of discriminating malware categories and reported a multi-class accuracy of 99.46%. Interestingly, for ransomware and SMSware categories, the accuracy was found to be higher, perhaps due to the use of obfuscation and encryption by other malware types. In all cases, a two-layer-stacked LSTM model was used.

Shabtai et al. focused on self-updating malware detection using deviations in the network traffic pattern [3]. They consider the task of Android malware detection as an anomaly detection problem rather than the more common classification problem. Their approach is composed of two sets of models: local, client-side models, and global, server-side models. The local models monitor the applications already installed on the user's device and try to identify any deviations in the network pattern. The global models focus on newly installed applications. The list of features used is presented in Table IV. They conducted experiments using various subsets of these, along with different values for anomaly acceptance. They found decision table and decision tree models to be best, achieving accuracies of 87.5% on a small dataset.

B. Text Representation of Network Traffic

An emerging approach in processing network traffic is using text semantics. Information transmitted across networks combines various protocols such as HTTP, DNS, and others. Each protocol has its respective components, such as HTTP headers,

Byte-based features
Average number of bytes (sent/received)
The total number of bytes used for headers (sent/received)
Ratio of number of incoming bytes to number of outgoing bytes
Average number of bytes per second
Packet-based
Total number of packets (sent/received)
Total length of packets (sent/received)
Average number of packets per second
Average number of packets per second (sent/received)
Min, mean, max, and standard deviation (SD) of the size of packet
Min, mean, max, and SD of the size of the packet (sent/received)
Average number of packets (sent/received/bulk
Sub-flow packets (sent/received)
Ratio number of incoming packets to number of outgoing packets
Time-based features
Min, mean, max, and SD time between two packets sent in the (forward/backward) direction
Min, mean, max, and SD time a flow was idle before becoming (active/idle)
Flow-based features
The duration of the flow
Min, mean, max, and SD of the length of a flow
Average number of packets per flow
Average number of packets (sent/received) per flow
Average number of bytes (sent/received) per flow
The average number of bytes in a sub flow in the (forward/backward) direction
Variance of total number of bytes used in the (forward/backward) direction
Number of packets with FIN, SYN, RST, PSH, ACK, URG and CWR
The total number of bytes sent in initial window in the (forward/backward) direction
Avg, max/min segment size observed in the (forward/backward) direction

TABLE III: Fallah and Bidgoly's features [2]

ID	Description
1	Data sent in Bytes
2	Data received in Bytes
3	Data sent in percentage
4	Data received in percentage
5	Network State (Cellular, WiFi or None)
6	Time since last data transmission in seconds
7	Transmission type (Eventual or Continuous)
8	Application state (Foreground or Background)
9	Time in Foreground in seconds
10	Time in Background in seconds
11	Time in Foreground in percentage
12	Time in Background in percentage
13	Time since the application was last active

TABLE IV: Shabtai et al's features [3]

DNS records and TCP flags. This data can be transformed into structured texts. This transcription involves parsing the structures and contents of network packets and representing them in a structured, readable format comparable to sentences and paragraphs. Subsequently, these texts can be processed using Natural Language Processing (NLP) techniques.

Wang et al. did this with HTTP and TCP traffic [4]. Their dataset, which they created by combining VirusShare2017 and applications they obtained from Baidu, was composed of 31706 benign samples and 5258 malicious samples. They mainly focused on HTTP and HTTPS traffic and filtered out NBNS, LLMNR, DHCP, and DNS traffic as they considered these to be unrelated to malware detection. The authors pre-processed HTTP and HTTPS traffic by extracting the headers and storing them as text files. These text files were then filtered to remove common words such as "en-us" and "expires", meaningless low-frequency words such as file extensions ".css" and ".js", and stop words such as "the" and "is". This was followed by N-gram generation, where each sequence of N-words is used as a potential feature to represent its respective HTTP(S) flow. The last step of feature pre-processing was to reduce the feature count using the chi-squared score. Using Support Vector Machines (SVM), they experimented with different N-gram sizes. They achieved a detection rate of 99.15% for malicious flows and a False Positive Rate (FPR) of 0.45% using N-gram size 1 and 600 features, which was measured using 10-fold cross-validation.

C. Image Representation of Network Traffic

Another popular recent approach is the use of image representations. As described by Wang et al. [26], network traffic can be encoded as greyscale images in which each byte of the network capture file (PCAP) represents one pixel. This approach captures the spatial relationships between packets. However, it is sensitive to the image size (usually a 28×28 pixel image), as larger images can capture more detailed information but also increase computational demands and the potential for introducing features that may confound the model.

Wang et al. utilized a CNN classifier that's coupled with their tool USTC-TK2016 which they developed to transform network capture to greyscale images [26]. Their tool can be configured to capture different levels of granularity when pre-processing the network flow. It can be set to capture all layers or only the 7 layers of the OSI model. Additionally, it can capture only TCP flows or sessions, with the difference being that sessions are bidirectional flows. They used a CNN architecture similar to LeNet-5 [27], and reported a binary accuracy of 100% when using all layers and session data.

Xu et al. developed an approach called Hybrid-Falcon that analyses both spatial and temporal aspects of network flow [5]. Their method transforms network traffic flows into 2D images. A network flow is defined as a 5-tuple composed of source IP, destination IP, source port, destination port, and protocol; packets with the same 5-tuple are considered to be in the same flow. The network traffic analysis part of their model comprises two parts: a CNN and a Long Short-Term Memory (LSTM). The CNN is used to translate each image into a feature vector. The LSTM is then used to translate a time series of these feature vectors into a summary feature vector that captures the temporal dimension of the network traffic. This is combined with features extracted from the application's source code and fed into a classifier. The overall accuracy of the model is reported as 97.16% when evaluated on a balanced dataset of about 50,000 applications, though this falls to 95.39% when the code analysis component is removed.

In a more recent approach, Shen et al. [6] also used a model that combines a CNN with an LSTM. However, their model notably differs from Xu et al. in the use of a self-attention layer within the LSTM model, in theory allowing the model to take into account more dependencies when reaching a classification. They report a malware detection accuracy of 99.12% evaluated using 10-fold cross-validation on a relatively small

dataset containing malware from CICAndMal2019. They also found that their model outperformed various other traditional ML and deep learning approaches on this dataset.

Feng et al. described a 2-layer deep neural network model that used both static and network features [7]. Their model's first layer was responsible for static feature extraction, such as permissions, and outputs a feature vector. The second layer was a cascading CNN. The authors also addressed the challenge of unlabelled images by adding an autoencoder before the CNN. Focussing on HTPP, UDP, and TCP traffic, their preprocessing consisted of converting the PCAP files into greyscale images using Wang et al.'s USTC-TK2016 tool. Evaluated using the CICAndMal2017 dataset, they reported a binary accuracy of 99% using a single 80:20 train-test split.

D. Limitations of Existing Studies

Studies in the literature that used matrix/tabular representation all focus on meta-data regarding the flow. That is, they do not use information about the packet payloads, and this consequently limits the information they have access to. However, they appear to achieve high levels of malware discrimination despite this. Whilst there is somewhat of a consensus regarding features used, there is a lack of consideration of protocols other than HTTP and TCP. The studies that did provide an explanation on their choice of protocols attributed their focus on HTTP and TCP to the limited importance of other protocols. This, however, makes the model ineffective in specific attack vectors such as DHCP and ARP poisoning. Finally, the nature of matrix/tabular data requires heavy pre-processing and, as a result, may drop valuable information.

There is currently a lack of literature utilizing text representations. The approach taken by Wang et al. [4] did not account for packet content but instead focused on the headers. While they did experiment with HTTPS traffic, its usage was minimal, with only 23% of the traffic being HTTPS compared to 77% to HTTP traffic. The authors also focused only on HTTP and TCP flow. The data underwent heavy pre-processing and filtering, which can be computationally expensive when scaled. Finally, they filtered out files such as JavaScript, which reduced the chance of catching malware that used attack vectors such as JavaScript Cross-Site Scripting (XSS), JavaScript Injection, or ClickJacking.

In the current literature on image representation, all studies appear to fix the size of the image at 28×28 pixels. This value stems from Wang et al. [26]. The authors chose this value as it covers the connection, included in the header of the packet, and a small amount of the payload. It also allows the model to be computationally less expensive. However, there is a lack of literature exploring alternative values. Furthermore, the literature only interprets grey-scale images, and no literature was found regarding colored images or the use of multiple channels.

However, in this study, we focus on addressing a number of limitations that are specifically linked to evaluation and understanding of ML models. Evaluation practice is a weakness across ML-based malware studies [15], [16]. Specifically, there is a tendency to use small outdated datasets that provide limited insight into how well a solution will work for detecting current malware. We see this in a number of the studies reviewed here. In this study, we address this by reevaluating existing approaches using a large up-to-date dataset and a contemporary Android environment.

There is also a lack of post-hoc analysis in previous studies, with no inspection of the trained models beyond reporting their evaluation metrics. This is potentially problematic, since it is well known that ML models often overfit spurious correlations within datasets, leading to poor performance when applied beyond their original datasets [28]. Most of the studies reviewed here report very high classification accuracies, yet offer no real insight into why network traffic-based features perform so well for malware discrimination. Since there is no obvious explanation for this, there is a need to explore how ML models use these features, and in this study we do this using explanatory AI techniques.

IV. Methodology

In this section, we describe our re-implementation and re-evaluation of approaches reviewed in Section III. We retain the distinction between the three categories of network traffic representation, namely image representation, text representation and matrix/tabular representation. In order to evaluate each of these representations, we follow the methodology shown in Figure 1. First, we select each paper from the reviewed papers and identify both its pre-processing techniques and model used. We then carry out the same pre-processing on our own dataset, and implement an ML model in line with the paper's specification. Finally we train and evaluate the model.

Data Pre-Processing: Each paper has its own unique set of features; therefore, we implement a separate script for each paper that follows the pre-processing steps or algorithms mentioned in the study. If the study mentioned the use of any library or tool, we use the same library or tool to ensure fidelity.

Model Implementation: Each paper implemented its own unique model. We replicate the model's architecture with any hyperparameters mentioned in the study. In the case of information missing from published works, we attempted to obtain it by contacting authors. If the studies mention additional steps, such as hyperparameter optimization, we also implement these.

A. Data Collection

The dataset we used in this study is the Android Malware dataset created by Muzafar et al. at Heriot-Watt University using their tool, DroidDissector [29]. Muzafar et al.'s dataset contains an up-to-date collection of both Benign and Malicious Android APK traffic capture. This dataset contains a total of 130,000 PCAP files. The authors gathered their benign samples by scrapping Android webstores for APK files and then using the VirusTotal API to confirm they are benign. Their malicious samples were taken from VirusShare's repository of malicious APK files.

Authors	Representation	Best Model	Dataset	Highest Accuracy(%)	Classification Type
Authors	Representation	Best Model	Dataset	Highest Accuracy(%)	Classification Type
Chen et al.	Matrix	IDGC	Not Reported	Not Reported	2-Class
Wang et al.	Matrix	C4.5 Tree	Drebin	97.89	2-Class
Fallah & Bidgoly	Matrix	LSTM	CICAndMal2017	96.94	11-Class
Shabtai et al.	Matrix	Decision Table/ REPTree	Authors' own dataset	87.5	2-Class
Wang et al.	Text	SVM	VirusShare 2014-2017	99.15	2-Class
Xu et al.	Image	CNN + LSTM	Authors' own dataset	95	2-Class
Shen et al.	Image	CNN + LSTM + Self Attn.	CICAndMal2019	99.12	2-Class
Feng et al.	Image	CACNN	CICAndMal2017	99.19	2-Class

TABLE V: Overall Literature Review

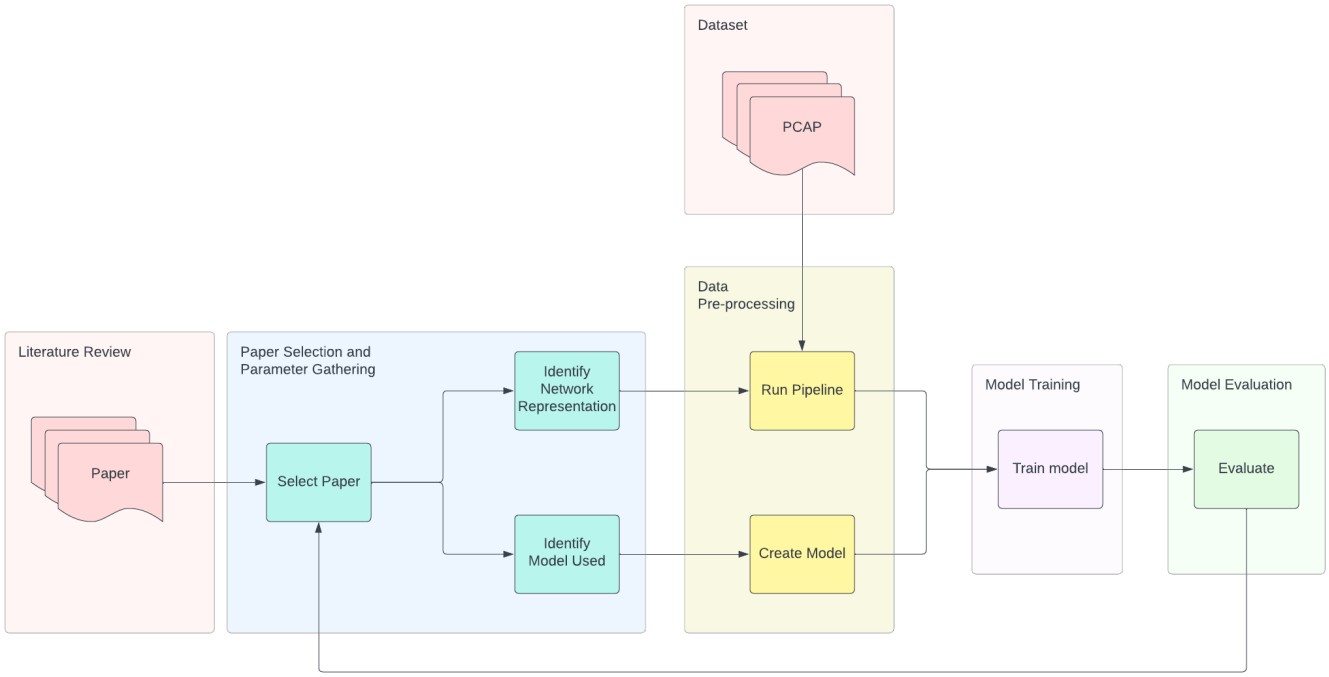


Fig. 1: System Architecture

B. Data Pre-processing

For each of the three network representations, we implemented a pipeline to transform the PCAP files into these representations. Each representation captures different aspects of the network traffic.

Matrix/Tabular Representation : In the matrix/tabular representation, the choice of features varies across papers. Most of the literature uses features relating to packets' size and headers (in bytes), their number and traffic statistics such as time between packets and frequency. Some studies extract features on multiple levels, such as packet-level, flow-level, and the temporal aspect of the flow; for example, Fallah and Bridgoly [2] and Shabtai et al. [3].

Text Representation : In the text representation, each flow is transformed into a vector. This vector is constructed by first extracting all the HTTP Request headers per flow, then these headers are reduced by removing stop words, low-frequency words, and high-frequency words that are not useful for classification. These filtered headers then undergo segmentation. Using these segmented headers, n-grams are generated. The authors then perform feature selection on these n-grams using the Chi-square test. This then results in a bag-of-words. A vector representing a flow n is then calculated by one-hot encoding this flow's n-grams using this bag-of-words. Figure 2 shows this process. This approach can be theoretically adjusted to work with multiple protocols. In practice this would be challenging, since each protocol will have different headers, which might result in the most common protocol dominating the bag-of-words therefore requiring a large feature vector to counteract this.

Image Representation : In the image representation, bytes of the network traffic are transformed into pixels. The size of the image determines how many bytes of the network traffic are represented. Most of the authors who used the image representation adopted the one proposed by Wang et al. [26]. As shown in figure 3, Wang et al. transformed each network flow into a fixed-size greyscale image of 784 bytes (28×28 pixels). They argued that the first 784 bytes of the traffic capture most of the initial connection data, which should be able to best reflect the characteristics of the flow while reducing the size of the images. This byte-to-pixel approach doesn't allow this representation to account for the spatial

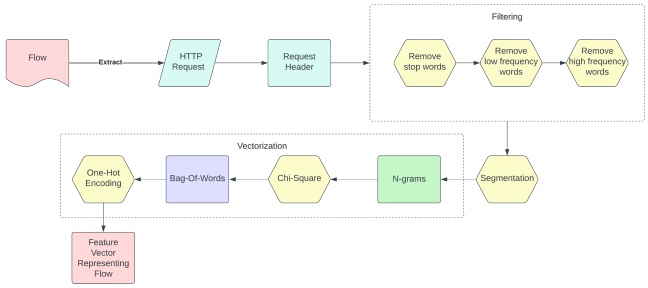


Fig. 2: Text Network Representation

dimension of the data. As seen in the figure 3, the packets are stacked into rows of 28 bytes but the data itself doesn't make sense in a 2D space as it is sequential. Additionally, the temporal aspect is more tricky to accommodate in this representation.

C. Models

ML models in Android malware detection can be divided into two main groups: traditional machine learning models and deep learning models. Traditional ML models have been widely used in Android malware detection. While traditional ML models are able to achieve good results in both detection and classification [15], [24], they struggle to capture the spatial and temporal aspects of network traffic. To address this, deep neural networks, such as Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) models have been implemented and tested in various studies [2], [5]–[7], [26]. The use of CNNs permits incorporation of the spatial aspect of the data; typically the data is represented as an image in these approaches. LSTMs complement this by incorporating temporal aspects of the data, with the data represented as a time series. According to the literature, these deep neural network approaches have achieved results that exceed those of traditional ML models [2], [6].

D. Feature Selection

The papers that proposed deep learning models did not perform feature selection, since these models typically extract their own features [5]–[7]. Papers that used traditional machine learning and matrix/tabular representations all used a predefined list of features, and did not perform further feature selection. In the text-based approach of Wang et al. [4] a Chi-Square test was used to perform feature selection.

E. Performance Metrics

In order to rigorously assessing the performance of network traffic representations and models, this study employs a range of evaluation metrics:

Confusion matrix provides a breakdown of the model's classification performance.

- True Positive (TP): The application correctly predicted as malicious
- False Positive (FP): The application falsely predicted as malicious
- True Negative (TP): The application correctly predicted as benign
- False Negative (TP): The application falsely predicted as benign

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Accuracy is the percentage of correct predictions made by the model.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total predictions}}$$

Precision is the percentage of correctly classified malware among all predictions classified as malware.

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

Recall or sensitivity is the percentage of malware samples detected by the model.

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

F1-Score is the harmonic mean of precision and recall.

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

Area Under the ROC Curve (AUC-ROC) A Receiver Operating Characteristic (ROC) curve plots recall against False Negative Rate (FNR) at different thresholds. AUC measures the area under this curve, providing a measure of the model's performance across thresholds which can be interpreted as the probability of malware producing a stronger response from the model than benign software.

F. Explainability

In an effort to explain the basis of decisions made by trained models and determine why certain approaches perform better than others, we conduct explainability analysis. For models with discrete features, we use Local Interpretable Model-Agnostic Explanations (LIME) [30] for this, since this is a widely used and well-understood method for quantifying the influence of features on model outputs. For CNN models, we generate saliency maps, since these capture the influence of each individual image pixel on the model's output. The analysis in each case is conducted using a selected sample from the training dataset. It should be noted that explainable AI (XAI) methods such as these are only indicative, and are not able to give a complete understanding of how a model works. This is a limitation of all current XAI methods.

V. Matrix/Tabular Representation

The papers that used this representation adopted different data extraction pipelines. We followed the general evaluation protocol presented in Section V-A when re-implementing the papers. We give the implementation-specific details of each

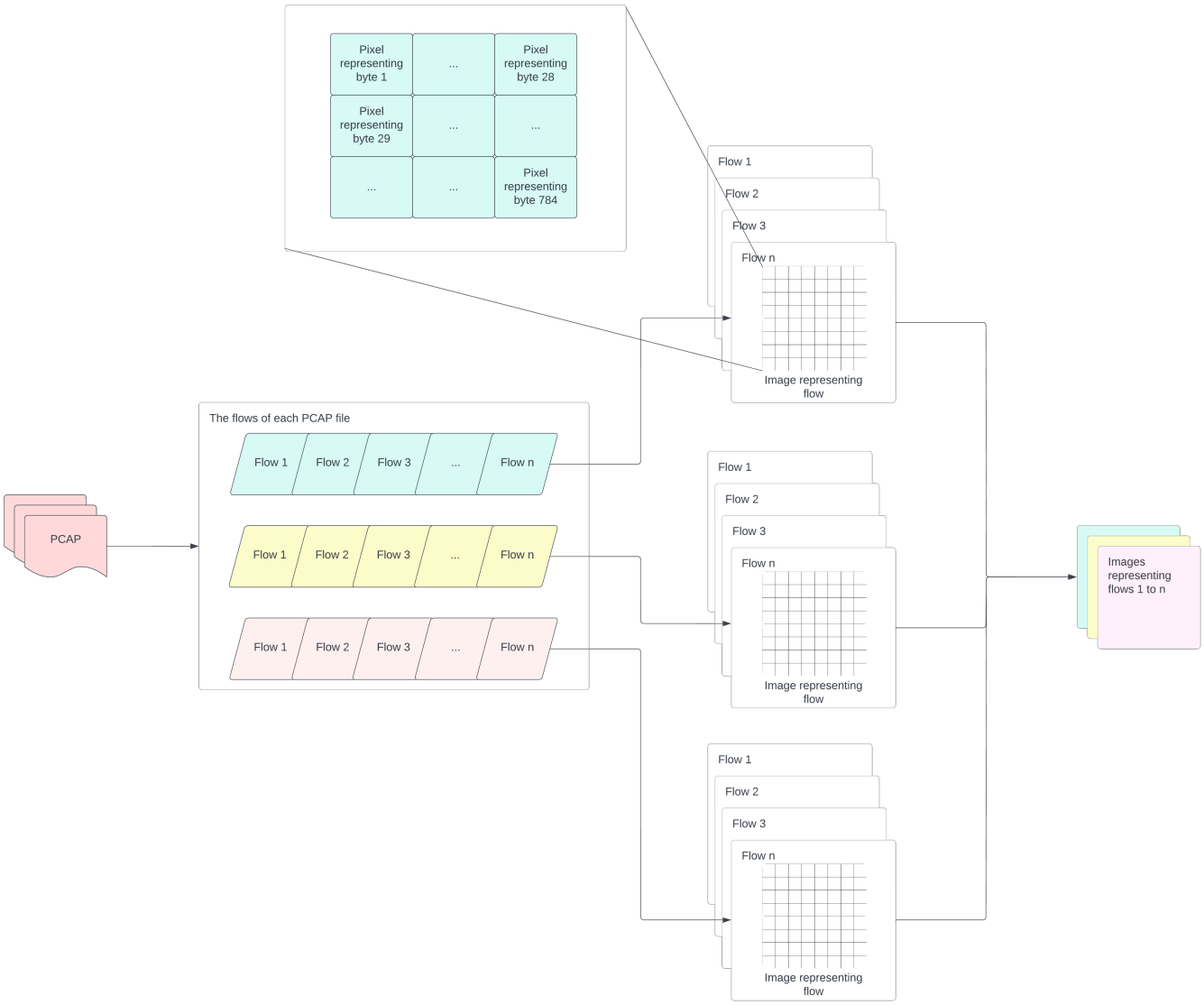


Fig. 3: Image Network Representation

paper and present our implementation results in Section V-B. We then compare the performance of the trained models using statistical tests in Section V-C, and perform an explainability analysis in Section V-D.

A. Evaluation Protocol

We use the evaluation protocol shown in Figure 1 for all papers in all categories. We start by running the data pre-processing pipeline, including the extraction of the features, and potentially feature selection if implemented in the paper. Then we implement the model(s) to the specifications of the paper. We chose to evaluate models using 10-fold cross-validation, since this offers a robust way of measuring performance that is more informative than a simple train-test split. Then, we compare the results of the models using statistical tests. We use the Kruskal-Wallis H test, which is a non-parametric omnibus test used to check whether there are significant differences between the metric distributions of a group of models [31]. If positive, this is followed by Dunn's tests to check if there for significant differences between pairs of models [32]. We use the Bonferroni correction to account for multiple comparisons [33]. This is done in accordance with the guidelines put forward in [28]. Before reimplementing papers, we contacted the paper authors whenever there were any uncertainties regarding implementation details, especially for missing hyperparameters. In cases where the authors did not reply, we attempted to reach sensible design decisions and documented these.

B. Reimplementation and Results

1) Reimplementation of Wang et al. [1]: We created two feature sets following Wang et al. [1], namely, one with TCP features and one with HTTP features. Both the TCP and HTTP sets were split 50-50 between benign and malicious applications, resulting in 9280 TCP samples and 23573 HTTP samples. Wang et al. [1], used a C4.5 decision tree model,

Algorithm 1 Model Evaluation Process

```
for each model in representation do
    Run the data pre-processing pipeline
    Implement model
    if Missing hyper-parameters then
        Contact authors
        if Authors reply then
            Use authors' response
        else if No response then
            Choose hyper-parameters
        end if
    end if
    Evaluate model Using 10-fold cross-validation
    Save results distribution
end for
Run Kruskal-Wallis H test on results distributions
if Significant then
    Run pairwise Dunn's tests with Bonferroni corrections
end if
Save and report results
```

which we reimplemented using WEKA's open source version, J48. The authors did not mention the framework they used; WEKA was chosen since it implements a C4.5 model. The authors also did not provide hyperparameters, so we used WEKA's default settings. The authors trained models using the two feature sets separately and in combination, and we did the same. The authors used a simple train-test split. However, as with the other reimplementations reported here, we evaluated these models using 10-fold cross-validation. The results as presented in Table VII.

The results of our implementation indicate a high level of malware discrimination, but not as high as the accuracies ($\sim 98 - 99\%$) reported in the original study. The difference can likely be attributed to the use of a different and much larger dataset, as well as potential differences in ML packages/framework and model hyperparameters. In general, we do not observe much difference between models trained on HTTP, TCP and both. In our results, TCP performs slightly better than HTTP, which counteracts the conclusions of the original study. However, we do see a slight benefit to combining both, which agrees with the original study.

2) Reimplementation of Fallah and Bidgoly [2]: We reimplemented Fallah and Bidgoly's LSTM model, which attempts to capture both spatial and temporal aspects of network flow, and used the largest set of features out of all the studies. The authors did not provide the exact number of cells per layer, and they did not respond to our request for clarification. After initial experimentation, we found 50 cells to be a suitable value.

The 10-fold cross-validation results are presented in Table VII. This model has the lowest performance of the tabular/matrix-based models we reimplemented, and also less stability between evaluations, as shown by a higher standard deviation. At just under 90%, the accuracy is still reasonable, though is significantly less than the 99.96% reported in the original study. This may be due to using different datasets; their dataset contained only 9244 samples split 50-50 between benign and malicious It may also be influenced by our estimation of the layer size hyperparameter; however, the standard deviation did not noticeably change with different values.

Feature Set 1	Feature Set 2
Average Sent Bytes	Average Sent Bytes
Average Received Bytes	Average Received Bytes
Percentage of Received Bytes	%age of Average Received Bytes
Inner Average Send Interval	Inner Average Send Interval
Inner Average Received Interval	Inner Average Received Interval
Outer Average Send Interval	Outer Average Send Interval
Outer Average Received Interval	Outer Average Received Interval
	Average Sent Data Percentage
	Average Received Interval %age

TABLE VI: Feature sets of Shabtai et al.

3) Reimplementation of Shabtai et al. [3]: This study was notable for considering a range of feature sets. Specifically, the authors identified those shown in Table VI as being the best for building models, so we also used these in our reimplementation. We also followed the authors in using WEKA's DecisionTable and REPTree, which they found to be best performing models. The authors did not provide hyperparameter settings, so we assumed these were not optimised and used the default settings from WEKA.

The 10-fold cross-validation results are presented in Table VII. These are notably higher than the accuracies reported in the original study (87.5%), though we were using much more data to train the models. It is notable that the accuracies are similar to those achieved in Wang et al's study, which may suggest that the exact choice of feature set is not so important.

C. Statistical Comparison

To compare the models, we first conducted the Kruskal-Wallis H test, to look for group-wise differences among the distributions of accuracy collected during 10-fold cross-validation. This gave a positive result, so we followed it up using pairwise Dunn's tests. Table VIII shows the resulting p-values. Assuming a significance level of 95% (i.e. p-value ≤ 0.05), this suggests there are no significant differences between Wang et al's three decision tree models and Shabtai et al's tree-based model, and that this set of models collectively have better performance than Fallah and Bidgoly's LSTM and Shabtai et al's decision table model. Interestingly, this suggests that the choice of model is more important than the choice of features. Notably, HTTP and TCP appear equally predictive, at least for these particular feature sets.

D. Explainability Analysis

We next apply LIME (see Section IV-F) to the trained models, to get some insight into the basis of their decisions. For models with a matrix/tabular representation, these explanations are based on feature values and ranges.

Figure 4 shows the LIME explanations for the models with matrix/tabular feature representations. Across the models, it appears that the most influential factors are the number of

Authors	Model Name	Avg. Accuracy ± Std	Avg. Precision ± Std	Avg. Recall ± Std	Avg. F1-Score ± Std	ROC-AUC
Wang et al.	HTTP	0.961 (± 0.003)	0.961 (± 0.003)	0.961 (± 0.003)	0.961 (± 0.003)	
Wang et al.	TCP	0.966 (± 0.005)	0.966 (± 0.005)	0.966 (± 0.005)	0.966 (± 0.005)	
Wang et al.	HTTP-TCP	0.969 (± 0.003)	0.969 (± 0.003)	0.969 (± 0.003)	0.969 (± 0.003)	
Fallah and Bidgoly	LSTM	0.883 (± 0.025)	0.879 (± 0.034)	0.894 (± 0.026)	0.879 (± 0.033)	
Shabtai et al.	DECISIONTABLE	0.950 (± 0.006)	0.950 (± 0.006)	0.950 (± 0.006)	0.950 (± 0.006)	
Shabtai et al.	REPTREE	0.963 (± 0.008)	0.963 (± 0.008)	0.963 (± 0.008)	0.963 (± 0.008)	

TABLE VII: Combined Results for matrix/tabular representation models

Comparison	Statistic	P-Value	Significant
Sequential Deep Learning vs. Anomaly Detection Decision Table	45.453090	1.000000	No
Sequential Deep Learning vs. Anomaly Detection REPTree	45.453090	0.000803	Yes
Sequential Deep Learning vs. Behaviour Based TCP	45.453090	0.000014	Yes
Sequential Deep Learning vs. Behaviour Based HTTP	45.453090	0.005293	Yes
Sequential Deep Learning vs. Behaviour Based HTTP+TCP Combined	45.453090	0.000000	Yes
Anomaly Detection Decision Table vs. Anomaly Detection REPTree	45.453090	0.228700	No
Anomaly Detection Decision Table vs. Behaviour Based TCP	45.453090	0.014865	Yes
Anomaly Detection Decision Table vs. Behaviour Based HTTP	45.453090	0.750222	No
Anomaly Detection Decision Table vs. Behaviour Based HTTP+TCP Combined	45.453090	0.000639	Yes
Anomaly Detection REPTree vs. Behaviour Based TCP	45.453090	1.000000	No
Anomaly Detection REPTree vs. Behaviour Based HTTP	45.453090	1.000000	No
Anomaly Detection REPTree vs. Behaviour Based HTTP+TCP Combined	45.453090	1.000000	No
Behaviour Based TCP vs. Behaviour Based HTTP	45.453090	1.000000	No
Behaviour Based TCP vs. Behaviour Based HTTP+TCP Combined	45.453090	1.000000	No
Behaviour Based HTTP vs. Behaviour Based HTTP+TCP Combined	45.453090	0.493678	No

TABLE VIII: Statistical comparison of models using matrix/tabular representation

packets or bytes being uploaded, and the host or URI. The more accurate models have relatively few influential factors, suggesting that the decision logic may be quite simple, based on considering data volume and common hosts/URIs. This is perhaps concerning, since identifying features such as hosts are by nature quite brittle.

It is notable that the LSTM model appears to take into account a lot more features when reaching a decision. Given its relatively low performance metrics, and taking the analysis of the decision tree models into account, this may be a sign of overfitting.

There are also some more subtle variations in feature importance in Wang et al's decision tree models, and this may explain their minor differences in average performance. For instance, the importance of upBytes and averageUpPckBytes is inverted between the TCP and combined HTTP+TCP model.

The decision table model of Shabtai et al. is somewhat of an outlier, and appears to base its decisions on different features. Although it does still consider the number of bytes sent, the most influential features are all concerned with intervals. Whilst this may just be a reflection of its slightly lower performance, it may also indicate more than one strategy for detecting malware from network traffic.

VI. Text Representation

We next reimplement and reevaluate a model which uses a text representation V-A. There is only one prior work which used this approach.

A. Implementation Specification and Results

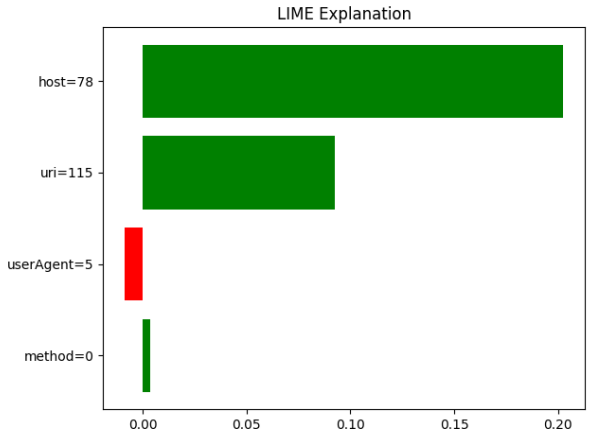
1) Reimplementation of Wang et al. [4]: This study involved extracting HTTP header text and, after processing, using an SVM to classify the data. We reimplemented the SVM model described by the authors using Python's Scikit-learn library. Hyperparameter values were not provided by the original paper or through correspondence, so again, we used default values and specified the kernel to be a linear kernel, which we deduced from the graph drawn by the authors. The pre-processing is explained in detail in section IV-B. For features, we used values which the authors reported as leading to the best performance: an n-gram size of 1 and a final vector size of 600.

The 10-fold cross-validation results are shown in Table IX. The results differ from the original study, where values in excess of 99% were reported. A difference of this magnitude seems unlikely to be due to differences in the data set alone, but may be explained in part by differences in hyperparameter settings. Beyond performance, this model is also potentially problematic due to the significant preprocessing overhead, which may will result in a model that is hard to keep up to date. Additionally, this approach is currently only implemented for HTTP requests. While it can be scaled to accommodate more protocols, this will present its own set of challenges, especially when it comes to the vectorization phase or processing as the distribution of protocols is not equal.

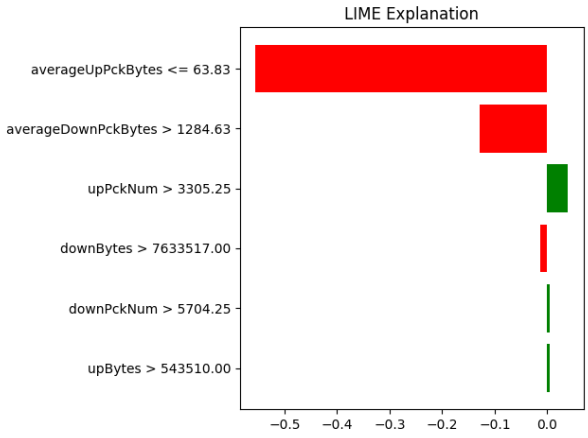
B. Explainability Analysis

To explain the models, we followed the same protocol as in Section III-A. However, for text representation, another step was required to decode the one-hot-encoded data back to their original n-grams.

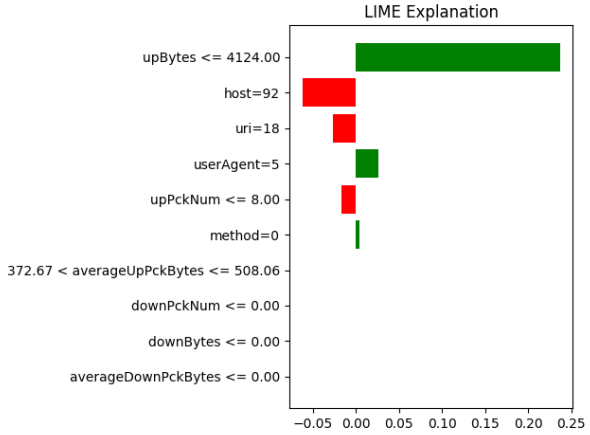
Figure 5 shows the top 25 most influential features. The feature names shown in the image are the 1-grams after undergoing pre-processing. The most influential feature `enq08` is originally from the string `Accept-Language:`



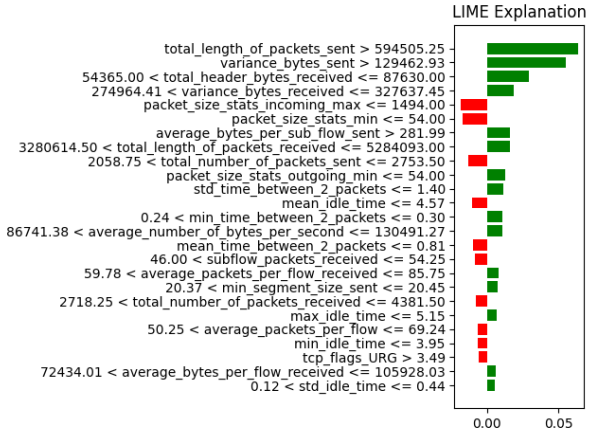
(a) Wang et al. HTTP decision tree



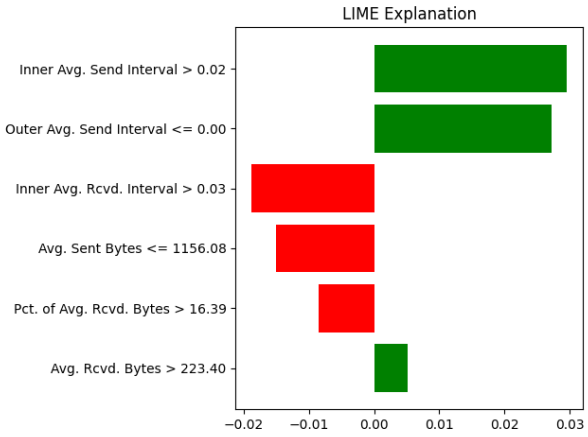
(b) Wang et al. TCP decision tree



(c) Wang et al. HTTP+TCP decision tree



(d) Fallah and Bidgoly LSTM



(e) Shabtai et al. decision table with Feature Set 1

Fig. 4: Explainability analysis of models using matrix/tabular feature representations. In each case, feature values are ranked, top-to-bottom, based on their influence on the model's decision. Green and red indicate that a feature value was influential on a sample being classed as malware or benign, respectively.

Avg. Accuracy ± Std	Avg. Precision ± Std	Avg. Recall ± Std	Avg. F1-Score ± Std
0.742 (± 0.044)	0.783 (± 0.026)	0.742 (± 0.043)	0.730 (± 0.053)

TABLE IX: Cross Validation Results for SVM

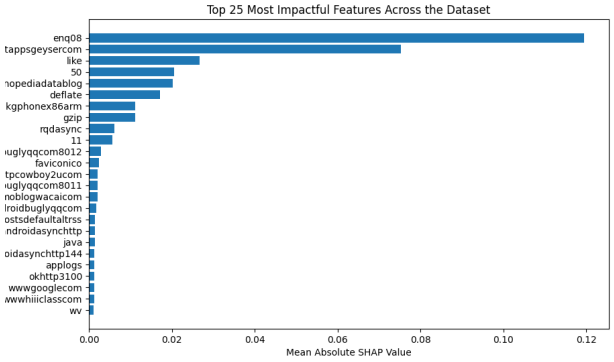


Fig. 5: LIME analysis of Wang et al.'s SVM model

`en;q=0.8` which is the accepted language along with the relative quality factor. Most of the other features are URLs along with keywords such as `gzip` and `faviconico`. These findings are similar to that of the matrix/tabular representation with the HTTP feature set (Section V-D) where the highest impacting features were the host and URI, and again suggests the models may be learning potentially brittle features.

VII. Image Representation

The image representation for the network flow was generated in the same way for all the models. We used the USTC-TK2016 tool of Wang et al. [26]. All the papers used the same image size, 28×28 pixels. After preprocessing all network flows, the dataset had 10476601 samples split between 5112695 benign and 5363906 malicious. We again use the evaluation protocol presented in Section V-A.

A. Implementation Specification and Results

1) Reimplementation of Xu et al. [5]: We reimplemented both the CNN and Bi-LSTM models. For the CNN, the authors provided all the hyperparameters required to reconstruct it using TensorFlow. However, for the Bi-LSTM, the authors did not list the number of cells in the LSTM layer, and we were not able to obtain this through correspondence. We tested with a range of cell values and found the highest accuracy was achieved when using 32 cells. Therefore, we used this value. Xu et al.'s proposed model is composed of a CNN that is used to transform the grey-scale image into a vector representing the image. This vector is then fed into a Bi-LSTM with a dense network for classification. The Bi-LSTM takes in a 32-vector that is representative of the greyscale image. These vectors are processed in batches of 8, representing an image sequence. The Bi-LSTM is connected to a fully connected layer that classifies the output into one of two categories, malicious or benign.

Our 10-fold cross-validation results are presented in Table XI. The results differ from those reported in the original study. This is unlikely to be due to the number of cells in the Bi-LSTM, as we tested with a wide range of values, and the accuracy remained at around 0.5. Furthermore, the choice of batch size 8 made the training relatively slow. The Bi-LSTM is also unstable, as seen by the high standard deviation in both accuracy and loss. However, the CNN layer alone performed very well, suggesting that the Bi-LSTM is not needed.

2) Reimplementation of Shen et al. [6]: We implemented the CNN, LSTM and LSTM with self-attention mechanism. For the CNN and self-attention layer, the authors provided all the hyperparameters required to reconstruct it using TensorFlow. However, for the LSTM, the authors did not list the number of cells in the LSTM layerand this could not be obtained through correspondence. We selected the number 32 for the cells as this is a standard size for the 64 features the LSTM processes.

Shen et al.'s proposed model is composed of a CNN that is used to transform the grey-scale image into a matrix representing the image. This matrix is the last feature map in the CNN. This feature map is then extracted and flattened. This vector is then fed into the LSTM. The LSTM has a self-attention mechanism which is intended to improve the performance of the model. Finally, the self-attention layer is followed by a fully connected layer for classification.

Our 10-fold cross-validation results for the CNN, LSTM and LSTM-self-attention models are all presented in table XI.

The results are very close to those reported by the original study. These differences are likely due to the exact parameters of the LSTM used by the authors. However, the addition of the self-attention layer does not seem to impact performance. We performed a Mann-Whitney test between the LSTM and the LSTM with the self-attention layer and found no significant increase in performance as presented in tables X and **??**.

3) Reimplementation of Feng et al. [7]: This model is composed of a convolutional auto-encoder (CAE) that handles unlabelled images, which is then processed by a CNN with a fully connected layer at the end for classification (CACNN). We reimplemented both components. For the CAE, the authors did not list the exact specifications, and we could not obtain these through correspondence. We designed the following Convolutional Auto-Encoder:

Encoder:

$$
\begin{aligned}
E_1 &= \text{ReLU}\left(\text{conv2d}^{64}_{3\times3}(X_{28\times28})\right)\\
E_2 &= \text{MaxPooling}_{2\times2}(E_1)\\
E_3 &= \text{ReLU}\left(\text{conv2d}^{32}_{3\times3}(E_2)\right)\\
E_4 &= \text{MaxPooling}_{2\times2}(E_3)
\end{aligned}
$$

Comparison	Statistic	P-Value	Significant
LSTM vs. Self-Attention + LSTM	14.296464	1.000000	No

TABLE X: Statistical comparison of Shen et al's LSTM model with and without self-attention

Authors	Model Name	Avg. Accuracy ± Std	Avg. Precision ± Std	Avg. Recall ± Std	Avg. F1-Score ± Std	ROC-AUC
Xu et al.	CNN	0.9987 (± 0.0004)	0.9994 (± 0.0010)	0.9995 (± 0.0009)	0.9994 (± 0.0009)	
Xu et al.	BI-LSTM	0.5120 (± 0.0000)	0.4001 (± 0.1299)	0.5021 (± 0.0053)	0.3670 (± 0.0438)	
Shen et al.	CNN	0.9454 (± 0.0100)	0.9975 (± 0.0036)	0.9975 (± 0.0035)	0.9975 (± 0.0036)	
Shen et al.	LSTM	0.5080 (± 0.0120)	0.3808 (± 0.1433)	0.5099 (± 0.0099)	0.4079 (± 0.0859)	
Shen et al.	LSTM-Self-Attention	0.5020 (± 0.0060)	0.3939 (± 0.1489)	0.5031 (± 0.0031)	0.3465 (± 0.0176)	
Feng et al.	CA-CNN	0.6003 (± 0.0146)	0.5093 (± 0.0660)	0.5074 (± 0.0634)	0.4926 (± 0.0632)	

TABLE XI: Combined results for image representation models

Comparison	Statistic	P-Value	Significant
Self-Attention vs. Hybrid-Falcon	25.806452	0.033255	Yes
Self-Attention vs. 2-layer deep learning	25.806452	0.000001	Yes
Hybrid-Falcon vs. 2-layer deep learning	25.806452	0.033255	Yes

TABLE XII: Statistical comparison of image representation models

Decoder:

$$
\begin{aligned}
D_1 &= \text{ReLU}\left(\text{Conv2DTranspose}^{32}_{3\times3}(E_{4shape})\right)\\
D_2 &= \text{UpSampling2D}_{2\times2}(D_1)\\
D_3 &= \text{ReLU}\left(\text{Conv2DTranspose}^{64}_{3\times3}(D_2)\right)\\
D_4 &= \text{UpSampling2D}_{2\times2}(D_3)\\
Y &= \text{Sigmoid}\left(\text{Conv2DTranspose}^{1}_{3\times3}(D_4)\right)
\end{aligned}
$$

This architecture is based on the details in the paper for the functions used in the Encoding and Decoding part, but, as for the layer, they were chosen based on the performances seen in both Xu et al. [5] and Shen et al. [6].

Results of 10-fold cross-validation are presented in Table XI. The results exceed those reported by the original study achieving an impressive 99.99% accuracy. The difference may be due to the choice of CAE model hyperparameters.

B. Statistical Tests Comparison

To compare the models, we again conducted Kruskal-Wallis H test, followed by Dunn's test [28] using the accuracies collected in the 10-fold cross-validation runs. The results presented in Table XII show that there are statistically significant differences between the models. Feng et al's [7] model has the highest mean accuracy and its results are significant higher than the other two approaches.

C. Explainability Analysis

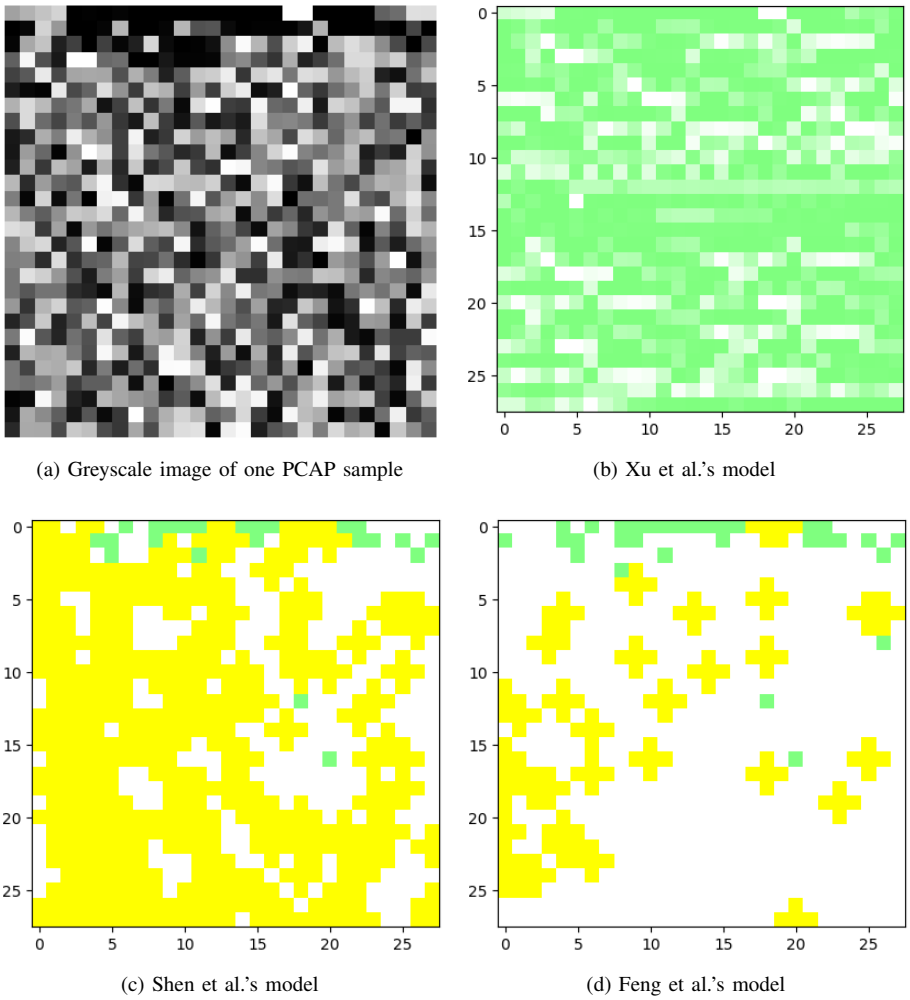
To explain the models, we generated a saliency map for each CNN. For models with multiple stages, for example a CNN followed by an LSTM, we did the explainability analysis only for the CNN part as we are only interested in finding out which parts of the traffic contribute to the final classification. For uniformity of explanations, we used one PCAP sample for all the saliency maps. The corresponding greyscale image is shown in 6, alongside the saliencty maps for each of the three CNN models.

Looking at the saliency map for Xu et al.'s [5] CNN in figure **??**, we can see that it focuses almost exclusively on the black pixels (lack of bytes) and leaving out the white pixels (existence of bytes). These bytes include the bytes of the header as well as the application data. The grey-scale image's first 2-3 rows contain the headers, including both the ETH and TCP headers. Which includes data such as the address and type for the Ethernet. And header length, total length, time-to-live, fragmentation options, flags, port and IP addresses. The rest of the image is mostly application data. Therefore, by not looking at the white bytes, Xu et al.'s CNN missed this information in part. While the accuracy of the CNN is still relatively high, the Bi-LSTM's input will not contain the header information, which, as seen in the III-A, were the most influential metrics in the highest-performing approaches.

Looking at the saliency map for Shen et al.'s [6] CNN in figure **??**, we can see there is much more focus on the white pixels, especially in the header with regional hotspots for the application data. The header, however, has more focus, as indicated by the green highlight, which signifies more focus than its yellow counterpart. While the data is encrypted, there is probably a correlation between these hotspots and the influential features seen in the matrix/tabular section III-A.

Finally, in the saliency map for Feng et al.'s [7] CACNN in figure **??**, we can see that the model still focuses mainly on the headers with an even larger hotspot around the header. This is in addition to the much fewer hotspots in the application data. This ratio of focus on headers to focus on regional app data can explain its relatively higher performance, which exceeds all other models.

It seems that the more the model focuses on regional features and, specifically, the headers, the better it performs. While the application data is encrypted, an educated guess is that it will have similar features to the ones seen in the matrix/tabular section III-A. Furthermore, the most influential features remain the header features which correlate with the findings in the matrix/tabular section. III-A



(a) Greyscale image of one PCAP sample

(b) Xu et al.'s model

(c) Shen et al.'s model

(d) Feng et al.'s model

Fig. 6: Saliency-based explanatory analysis of CNN models

VIII. Discussion

In this section, we aim to discuss the differences between the representations in terms of model performance and feature impact.

A. Model Comparison

In order to compare, we look at the results of the 10-fold Cross-Validation. In this regard, Wang et al.'s [1]'s C4.5 model outperforms all other models. This is also backed by the presence of statistically significant differences between the models as presented in table XIII. All 3 approaches handle TCP flow. Wang et al. [4]'s SVM feature selection would be difficult to implement in a real-world setting viewing the overhead it requires. Wang et al. [1]'s C4.5 Decision Tree feature selection was relatively lightweight and easy to implement and keep up-to-date in a real-world setting. However, both of these approaches suffer inherent data loss due to the feature selection. Feng et al. [7]'s CACNN is a deep learning model and therefore has no feature selection but does require the most resource-consuming data-prepossessing as the transformation of PCAPs to images took. All 3 approaches can be adapted to more protocols other than TCP. However, Feng et al. [7]'s CACANN model is the only model found in recent literature that can handle both labelled, and unlabelled data, which gives it a significant advantage in real-world implementation. It's worth mentioning that on a subset of the dataset (around 30000 images) Feng et al. [7]'s CACANN model achieved a very impressive 99.99% accuracy. However, as the data size increased the accuracy got gradually lower and lower. This is likely due to the model attemptting to memorize the IP addresses but not being to do so in such a large volume of data.

B. Features Comparison

The feature selection applies to the text and matrix/tabular representations. Looking at HTTP features the most influential are the HOST, URI, and language encoding. The HTTP method ranked surprisingly last in the matrix/tabular representation [1] and didn't rank among the top 25 features in the text representation [4]. Looking at the TCP features the most influential features were related to the size, especially the upload size) and frequency of the packets. This matches the behavior of malware that aims to upload the most amount

Comparison	Statistic	P-Value	Significant
2-layer deep learning vs. Text-Semantics	26.514286	0.030107	Yes
2-layer deep learning vs. Behaviour Based HTTP-TCP Combined	26.514286	0.000001	Yes
Text-Semantics vs. Behaviour Based HTTP-TCP Combined	26.514286	0.030107	Yes

TABLE XIII: Statistical tests results for All representations

Model	Avg. Accuracy ± Std	Avg. Precision ± Std	Avg. Recall ± Std	Avg. F1-Score ± Std
C4.5 Decision Tree	0.969 (± 0.003)	0.969 (± 0.003)	0.969 (± 0.003)	0.969 (± 0.003)
SVM	0.7419 (± 0.0436)	0.7834 (± 0.0256)	0.7419 (± 0.0432)	0.7304 (± 0.0531)
CACNN	0.6003 (± 0.0146)	0.5093 (± 0.0660)	0.5074 (± 0.0634)	0.4926 (± 0.0632)

TABLE XIV: Cross Validation Results for All representations

Representation	Authors	Model Name	Avg. Accuracy ± Std	Avg. Precision ± Std	Avg. Recall ± Std	Avg. F1-Score ± Std
Image	Xu et al.	BI-LSTM	0.5120 (± 0.0000)	0.4001 (± 0.1299)	0.5021 (± 0.0053)	0.3670 (± 0.0438)
		CNN	0.9987 (± 0.0004)	0.9994 (± 0.0010)	0.9995 (± 0.0009)	0.9994 (± 0.0009)
	Shen et al.	LSTM	0.5080 (± 0.0120)	0.3808 (± 0.1433)	0.5099 (± 0.0099)	0.4079 (± 0.0859)
		LSTM-Self-Attention	0.5020 (± 0.0060)	0.3939 (± 0.1489)	0.5031 (± 0.0031)	0.3465 (± 0.0176)
		CNN	0.9454 (± 0.0100)	0.9975 (± 0.0036)	0.9975 (± 0.0035)	0.9975 (± 0.0036)
	Feng et al.	CA-CNN	0.6003 (± 0.0146)	0.5093 (± 0.0660)	0.5074 (± 0.0634)	0.4926 (± 0.0632)
Matrix/Tabular	Fallah and Bidgoly	LSTM	0.8833 (± 0.0245)	0.8785 (± 0.0342)	0.8942 (± 0.0260)	0.8791 (± 0.0334)
	Wang et al.	HTTP	0.961 (± 0.003)	0.961 (± 0.003)	0.961 (± 0.003)	0.961 (± 0.003)
		HTTP-TCP	0.969 (± 0.003)	0.969 (± 0.003)	0.969 (± 0.003)	0.969 (± 0.003)
		TCP	0.966 (± 0.005)	0.966 (± 0.005)	0.966 (± 0.005)	0.966 (± 0.005)
	Shabtai et al.	REPTREE	0.950 (± 0.006)	0.950 (± 0.006)	0.950 (± 0.006)	0.950 (± 0.006)
		DECISIONTABLE	0.963 (± 0.008)	0.963 (± 0.008)	0.963 (± 0.008)	0.963 (± 0.008)
Text	Wang et al.	SVM	0.7419 (± 0.0436)	0.7834 (± 0.0256)	0.7419 (± 0.0432)	0.7304 (± 0.0531)

TABLE XV: Consolidated Results for Image, Matrix/Tabular, and Text Representation Models

of data in the least amount of packets **(I remember reading this in a paper, need to cite it here)**. This suggests that a malware developer can just change the flow of the traffic and it will go undetected.

IX. Limitations

Our work has highlighted the differences between network representations as well as the models coupled with them. We have assessed them in a fair and standardized manner. However, this study has a set of limitations.

- This study has only tested these models on a 2-class classification (Benign and Malicious). We cannot make any claims in regard to other types of classification, such as category or family classification.
- This study did not take into account hyperparameter tuning, which could have an effect on the performances of the tested models.
- The explainability analysis was conducted using LIME [30] which is not always 100% accurate and in some cases can sacrifice accuracy for the sake of explainability.

X. Conclusion and Future Work

In this study, we conducted a comparison between three representations. We highlighted the differences between them in terms of data processing, models used and performance. We tested all the approaches using a balanced, up-to-date dataset of Android malware and benign applications traffic that is, at present, the largest publicly available network traffic dataset for Android malware detection. We used both 10-fold Cross-validations as well as statistical tests to compare the approaches [28]. Results show that the C4.5 model proposed by Wang et al.'s [1] coupled with an text representation for the network flow performs the best in Android malware detection.

In the future, we have planned to work on the limitation of the study mentioned in section IX as well as evaluate more representations if there are any and further explore the capabilities and variants of the image representation.

References

[1] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, pp. 15–25, May 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804518304028

[2] S. Fallah and A. J. Bidgoly, "Android malware detection using network traffic based on sequential deep learning models," *Software: Practice and Experience*, vol. 52, no. 9, pp. 1987–2004, Sep. 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/spe.3112

[3] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1–18, Jun. 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404814000285

[4] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting Android Malware Leveraging Text Semantics of Network Flows," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1096–1109, May 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8101555/

[5] P. Xu, C. Eckert, and A. Zarras, "hybrid-Falcon: Hybrid Pattern Malware Detection and Categorization with Network Traffic and Program Code," Jan. 2022, arXiv:2112.10035 [cs]. [Online]. Available: http://arxiv.org/abs/2112.10035

[6] L. Shen, J. Feng, Z. Chen, Z. Sun, D. Liang, H. Li, and Y. Wang, "Self-attention based convolutional-LSTM for android malware detection using network traffics grayscale image," *Applied Intelligence*, vol. 53, no. 1, pp. 683–705, Jan. 2023. [Online]. Available: https://link.springer.com/10.1007/s10489-022-03523-2

[7] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic," *IEEE Access*, vol. 8, pp. 125 786–125 796, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9136685/

[8] G. Google, "Android open source project," 2023. [Online]. Available: https://source.android.com/

[9] StatCounter, "Mobile Operating System Market Share Worldwide," StatCounter Global Stats, 2022, accessed: 16 November 2023. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide/2022

[10] L. Ceci, "Global iOS & Google Play app downloads by quarter 2023," 2023. [Online]. Available: https://www.statista.com/statistics/695094/quarterly-number-of-mobile-app-downloads-store/

[11] ——, "Mobile app spend global 2022," 2022. [Online]. Available: https://www.statista.com/statistics/870642/global-mobile-app-spend-consumer/

[12] T. t. p. g. i. S. a. n. l. f. t. i. g. b. c. o. c. D. t. v. u. cycles and S. C. D. M. u.-t.-D. D. T. R. i. t. Text, "Topic: Malware," 2022. [Online]. Available: https://www.statista.com/topics/8338/malware/

[13] K. Huang, X. Wang, W. Wei, and S. Madnick, "The Devastating Business Impacts of a Cyber Breach," *Harvard Business Review*, May 2023, section: Cybersecurity and digital privacy. [Online]. Available: https://hbr.org/2023/05/the-devastating-business-impacts-of-a-cyber-breach

[14] K. Lab, "Android Mobile Security Threats," Apr. 2023, section: Resource Center. [Online]. Available: https://www.kaspersky.com/resource-center/threats/mobile

[15] A. Muzaffar, H. Ragab Hassen, M. A. Lones, and H. Zantout, "An in-depth review of machine learning based Android malware detection," *Computers & Security*, vol. 121, p. 102833, Oct. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404822002279

[16] A. Guerra-Manzanares, "Android malware detection: mission accomplished? a review of open challenges and future perspectives," *Computers & Security*, p. 103654, 2023.

[17] R. T. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122, Oct. 1989. [Online]. Available: https://www.rfc-editor.org/info/rfc1122

[18] ——, "Requirements for Internet Hosts - Application and Support," RFC 1123, Oct. 1989. [Online]. Available: https://www.rfc-editor.org/info/rfc1123

[19] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, 1st ed. USA: No Starch Press, 2012.

[20] D. Tidmarsh, "The complete guide to types of malware and prevention techniques," Nov 2023. [Online]. Available: https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/types-of-malware/

[21] A. K. Sood and R. Enbody, "Chapter 5 - data exfiltration mechanisms," in *Targeted Cyber Attacks*, A. K. Sood and R. Enbody, Eds. Boston: Syngress, 2014, pp. 77–93. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012800604700005X

[22] S. Wang, Z. Chen, L. Zhang, Q. Yan, B. Yang, L. Peng, and Z. Jia, "Trafficav: An effective and explainable detection of mobile malware behavior using network traffic," in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, 2016, pp. 1–6.

[23] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[24] A. Muzaffar, H. R. Hassen, H. Zantout, and M. A. Lones, "A Comprehensive Investigation of Feature and Model Importance in Android Malware Detection," Jan. 2023, arXiv:2301.12778 [cs]. [Online]. Available: http://arxiv.org/abs/2301.12778

[25] R. Thangaveloo, W. W. Jing, C. K. Leng, and J. Abdullah, "Datdroid: Dynamic analysis technique in android malware detection," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 10, no. 2, pp. 536–541, 2020. [Online]. Available: http://ijaseit.insightsociety.org/index.php?option=com_content&view=article&id=9&Itemid=1&article_id=10238

[26] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International Conference on Information Networking (ICOIN)*. Da Nang, Vietnam: IEEE, 2017, pp. 712–717. [Online]. Available: http://ieeexplore.ieee.org/document/7899588/

[27] Y. Lecun, L. Jackel, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Learning algorithms for classification: A comparison on handwritten digit recognition," *The Statistical Mechanics Perspective*, 07 2000.

[28] M. A. Lones, "How to avoid machine learning pitfalls: a guide for academic researchers," Feb. 2023, arXiv:2108.02497 [cs]. [Online]. Available: http://arxiv.org/abs/2108.02497

[29] A. Muzaffar, H. Ragab Hassen, H. Zantout, and M. A. Lones, "Droid-dissector: A static and dynamic analysis tool for android malware detection," in *Proceedings of the International Conference on Applied Cybersecurity (ACS) 2023*, H. Zantout and H. Ragab Hassen, Eds. Cham: Springer Nature Switzerland, 2023, pp. 3–9.

[30] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.

[31] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952. [Online]. Available: http://www.jstor.org/stable/2280779

[32] O. J. Dunn, "Multiple comparisons using rank sums," *Technometrics*, vol. 6, no. 3, pp. 241–252, 1964. [Online]. Available: http://www.jstor.org/stable/1266041

[33] W. Haynes, "Bonferroni Correction," in *Encyclopedia of Systems Biology*, W. Dubitzky, O. Wolkenhauer, K.-H. Cho, and H. Yokota, Eds. New York, NY: Springer New York, 2013, pp. 154–154. [Online]. Available: https://doi.org/10.1007/978-1-4419-9863-7_1213