Scientific Reports

https://doi.org/10.1038/s41598-026-38271-7

Article in Press

Ensemble machine learning for proactive android ransomware detection using network traffic

Received: 5 June 2025

Accepted: 29 January 2026

Published online: 18 February 2026

Cite this article as: Kirubavathi G., Padma Mayuri B., Pranathasree S. *et al.* Ensemble machine learning for proactive android ransomware detection using network traffic. *Sci Rep* (2026). https://doi.org/10.1038/s41598-026-38271-7

G. Kirubavathi, B. Padma Mayuri, S. Pranathasree, Rajeswari Alagappan, Amal Ajayan, Waleed M. Ismael & Ateeq Ur Rehman

Ensemble Machine Learning for Proactive Android Ransomware Detection Using Network Traffic

Kirubavathi G[1,*], Padma Mayuri B[1], Pranathasree S[1], Rajeswari Alagappan[1], Amal Ajayan[1], Waleed M. Ismael[2,*], and Ateeq Ur Rehman[3,*]

[1]Department of Mathematics, Amrita School of Physical Sciences, Coimbatore 641112, Amrita Vishwa Vidyapeetham, India;
[2]Department of Information Technology, Faculty of Engineering, Azal University for Human Development, Sanaa, Yemen;
[3]School of Computing, Gachon University, Seongnam 13120, Republic of Korea;
[*]Corresponding authors: g_kirubavathi@cb.amrita.edu (Kirubavathi.G), waleed.m@auhd.edu.ye (Waleed M. Ismael), 202411144@gachon.ac.kr (Ateeq Ur Rehman)

ABSTRACT

Android ransomware has emerged as a major threat to mobile ecosystems, leveraging obfuscated payloads and dynamic command-and-control channels to evade conventional detection systems. Existing approaches often rely on static, batch-trained models that lack adaptability to evolving threat behaviors, resulting in degraded accuracy over time due to concept drift. This presents a critical challenge for real-time deployment, as new ransomware variants continually mutate their signatures and alter network traffic patterns to evade detection. To bridge this gap, this study proposes a robust ensemble-based machine learning framework for proactive detection of Android ransomware using network traffic metadata. The framework integrates advanced classifiers, including Light Gradient Boosting Machine, eXtreme Gradient Boosting Machine, and Random Forest, with Synthetic Minority Oversampling Technique enhanced stratified cross-validation to mitigate class imbalance and improve generalizability. Furthermore, explainable artificial intelligence methods such as SHapley Additive exPlanations and Local Interpretable Model-Agnostic Explanations are employed to enhance interpretability and analyst trust. In the context of ransomware detection, the importance of online learning lies in its ability to adapt to evolving threat patterns in real time. Ransomware frequently mutates payload signatures and obfuscates behavioral traces, causing traditional models to deteriorate under changing data distributions. To address this, we conducted a concept drift evaluation using an incremental LightGBM model, tested on chronologically partitioned traffic data across five temporal blocks. This approach enables continuous adaptation to new data streams without requiring full retraining, thereby maintaining detection robustness and reducing false negatives in production. Experimental results on a balanced dataset demonstrate that LightGBM achieves the highest classification performance, indicating the efficacy and adaptability of the proposed framework for real-time Android ransomware mitigation in dynamic network environments.

1 Introduction

Rapid growth and widespread use of Android devices have significantly transformed the way people communicate, work, and access entertainment. Although these advances have brought convenience and connectivity, theyâĂŹve also introduced new security risks by widening the potential attack surface for cybercriminals[1,2]. Among the various threats targeting mobile platforms, ransomware stands out as particularly dangerous, both in its technical impact and its financial consequences. Unlike conventional malware, which often remains hidden while stealing information, ransomware actively disrupts users by encrypting their data or locking their devices, then demands payment, usually in cryptocurrency[3]. TodayâĂŹs variants employ sophisticated methods, such as evasion tactics, code obfuscation, and polymorphism, making detection and mitigation significantly more challenging[4,5]. This evolving nature of the threat underscores the need for effective, real-time detection strategies to prevent significant harm. Ransomware infections on Android devices generally follow a predictable pattern. The process begins when consumers unintentionally download malicious apps, usually from unlicensed app stores, third-party platforms, or phishing links disguised as genuine software[6]. Once installed, these apps request broad access permissions, masking their intent behind seemingly normal functionality. These permissions can include access to device storage, the camera, microphone, and even administrative controls[7]. After securing these privileges, the ransomware executes its core functions: encrypting user data, possibly transmitting it to a remote server, and, in some cases, locking the device. Following this, the user is confronted with a ransom demand often accompanied by threats of permanent data loss if payment is not made[8].

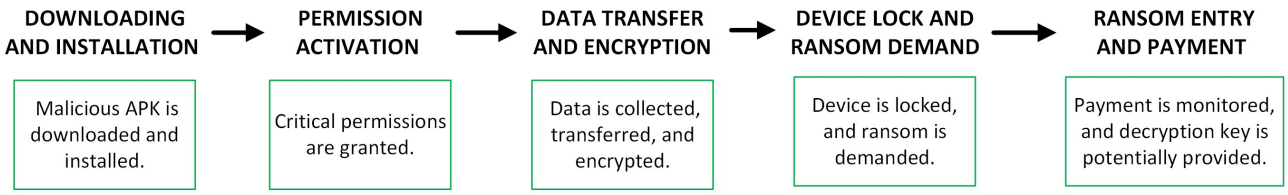

Figure 1. Lifecycle of an Android ransomware attack, illustrating the stages from downloading and installation to data encryption, ransom demand, and device lock

Some variants are designed to persist on the device even after attempts to uninstall them, making them particularly hard to remove. The consequences of such attacks extend beyond financial loss, as they can lead to severe disruption of personal life and business operations. Given the rising frequency and impact of ransomware incidents, it is clear that this threat will remain a major concern in the mobile cybersecurity landscape[9]. Figure 1 provides a clear, step-by-step view of how an Android ransomware attack typically unfolds. It starts when a user unknowingly downloads a malicious app that appears harmless. Once installed, the ransomware app requests sensitive permissions, such as access to the microphone, camera, and file storage. With these permissions granted, the malware begins its malicious activities: it encrypts personal files and silently sends sensitive data to the attackerâĂŹs remote servers[10].

Soon after, the ransomware reveals its true intent by displaying a ransom note, often a scary warning screen demanding payment to restore access. If the user does not pay, the final stage kicks in: the device is locked entirely, leaving the user with no access to their data[11]. Figure 1 also visualizes how victims get trapped, highlighting the devastating impact of such attacks, especially file encryption and complete device lockdown. This structured process illustrates why early detection is crucial. Ideally, security systems should intervene during the download or installation phase to block malware before it causes damage[12]. As ransomware becomes more complex, traditional antivirus tools that rely on signature matching are falling behind. These methods rely on known malware patterns, making them ineffective against brand-new or zero-day threats, as well as shapeshifting malware that changes its code to evade detection[13]. That is where behaviour-based detection comes in. By studying how ransomware behaves, particularly its network traffic, it's possible to detect it in real time. Machine Learning (ML) has proven especially useful here, identifying unusual activity and stopping threats before they cause harm[14]. Techniques such as ensemble learning, which combine multiple ML models, improve accuracy and reduce false alarms.

Recent studies have revealed that modern ransomware strains often employ advanced evasion techniques, such as obfuscation, polymorphism, and runtime Application Programming Interface (API) deception, making detection significantly more challenging. Lu et al. introduced AutoD[15], which targets deceptive JNI-layer API calls to unpack malicious Android applications, and later extended this work with DeepAutoD[16], offering a distributed, scalable system for secure mobile communication. These insights are directly relevant to Android ransomware detection, which increasingly depends on the robustness and adaptability of ML-based systems under adversarial pressure. In this research, we introduce a real-time ransomware detection system designed for Android devices. It uses ensemble ML models to analyse network traffic and spot signs of ransomware. Additionally, we tested several classifiers, including Logistic Regression, XGBoost, Random Forest, and LightGBM, on a dataset of network records that included regular traffic and data from ten known ransomware families, such as SVpeng, PornDroid, Koler, and WannaLocker. To ensure the best performance, applied thorough data cleaning, feature selection using Random Forest, hyperparameter tuning, and stratified K-Fold cross-validation. The results were promising. Among the models, Bagging stood out by achieving a classification accuracy, along with near-perfect precision and recall, demonstrating strong capability in distinguishing ransomware from benign traffic. The significant contributions of this study are as follows.

- **Ensemble-Based Detection Framework:** We develop a robust Android ransomware detection framework that leverages LightGBM, XGBoost, and Random Forest, ensuring high accuracy and resilience against evolving threat behaviours.
- **Hybrid Feature-Selection Pipeline:** A three-stage feature selection process is introduced by integrating Mutual Information (MI), Recursive Feature Elimination (RFE), and embedded Random Forest importance to identify the most discriminative network-traffic attributes.
- **Balanced and Generalisable Training Strategy:** The study incorporates SMOTE together with stratified K-Fold cross-validation to effectively address class imbalance across multiple ransomware families and improve generalisation capability.
- **Model Explainability:** Comprehensive interpretability is achieved through SHAP and LIME, enabling transparent understanding of feature contributions and supporting security analysts in decision-making.

- **Online Learning and Concept-Drift Evaluation:** Incremental LightGBM is employed to evaluate the framework under temporal concept drift, demonstrating its adaptability to chronologically evolving ransomware behaviours.

- **Extensive Benchmarking:** A comparative evaluation across five classifiers highlights the superior performance of ensemble methods, particularly Bagging and LightGBM, for real-time Android ransomware detection.

The rest of this paper is structured as follows: Section 2 reviews related work in the field. Section 3 presents our detection framework, detailing its components: data preprocessing, feature selection, and model training. Section 4 concludes the study and outlines possible future improvements.

2 Related Works

The growing threat of Android ransomware has led to a surge in research focused on mobile malware detection[17,18]. Researchers have explored a wide range of methods, from analyzing app behavior and monitoring network activity to using advanced ML and hybrid techniques, to improve detection accuracy and responsiveness[19,20]. Gu et al.[21] used graph neural networks (GNNs) to model interactions among apps, system permissions, and processes. Their method helped uncover abnormal patterns in app behavior that may indicate ransomware. This kind of graph-based analysis is particularly useful for detecting hidden relationships and irregular activities within AndroidâĂŹs complex ecosystem.

Hsu et al.[22] took a different approach, focusing on federated learning, a method of training ML models across multiple devices without sharing sensitive data. Their decentralised system preserved user privacy while still achieving high detection accuracy, making it a strong candidate for large-scale, privacy-friendly security solutions. Karat et al.[23] developed a malware detection model utilizing DL techniques, incorporating Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This combination helped track the evolution of ransomware behaviour over time and detect new variants more effectively. However, while powerful, such DL models are often resource-heavy and might not perform well on devices with limited processing power.

Other researchers have explored hybrid detection methods that combine static and dynamic analysis. Static techniques examine app code and permissions before installation, while dynamic methods monitor real-time behavior, such as file encryption or unauthorized process activity[24,25]. Each method has its limits: static analysis can miss obfuscated code, while dynamic analysis often needs time-consuming sandbox environments. Hybrid systems, especially those integrated with ML and network traffic analysis, aim to strike the right balance[26,27].

A common shortcoming across many ML-based solutions, however, is the lack of explainability. Most operate as black boxes, making decisions without providing an explanation. This lack of transparency makes it hard for users and even security professionals to trust or verify what the system is doing[28,29]. Scalability is another concern. Many models are designed for centralized systems and do not adapt well to real-world mobile environments, where bandwidth, processing power, and latency are crucial[30,31]. To overcome these challenges, lightweight models and smart feature selection techniques are being used to ensure that security solutions can run smoothly on typical Android devices[32].

Table 1 presents a comparative summary of recent studies on Android ransomware detection, highlighting the ML models employed, reported accuracy scores, and their key contributions or limitations. Ensemble-based approaches, such as Bagging and boosting algorithms like GradientBoost and CatBoost, consistently demonstrate high classification accuracy, reflecting a strong detection capability. DL models, such as LSTM and BERT, also achieve impressive performance but are constrained by high computational demands, limiting their deployment in resource-constrained mobile environments. Conversely, lightweight models such as Linear SVM and Naive Bayes offer efficiency but struggle with generalizability and the complexity of patterns. Additionally, graph-based models such as GNN provide structural behavior modeling but lack scalability validation. A notable gap across these studies is the limited focus on model interpretability and resilience to concept drift. Most models operate as black-box classifiers and are not evaluated under evolving threat scenarios. To address these limitations, our research integrates ensemble learning with XAI tools (SHAP and LIME) and introduces incremental LightGBM for online learning under concept drift.

Table 1. Summary of Recent Studies on Android Ransomware Detection

Study	Model Used	Accuracy (%)	Main Contribution / Limitation
Hossain et al.[7]	Ensemble	99.81	Ensemble model with high precision; lacks explainability
Amer et al.[17]	LSTM	99.30	DL for sequence modeling; resource-intensive
Ahmed et al.[33]	DT	97.24	Lightweight model; poor performance on complex patterns
Amenova et al.[34]	CNN-LSTM	94.39	Temporal modeling; high computational cost
Islam et al.[35]	Ensemble	95.00	Good accuracy; limited generalizability across families
Adeniyi et al.[36]	RF	99.98	Simple probabilistic model; poor at complex patterns
Amer et al.[17]	LSTM	97.50	Context-aware deep model; high resource usage
Gu et al.[21]	GNN	99.47	Detected abnormal app behavior; strong structural analysis but lacks scalability validation

Together, these studies underscore the growing importance of ML, DL, and network-based approaches in combating ransomware. Our work builds on this foundation by combining several ML models into an ensemble and focusing on network traffic analysis. This not only improves detection accuracy but also enhances scalability and explainability, making it a practical solution for todayâĂŹs evolving mobile threat landscape.

3 Proposed Detection Framework

The proposed research framework, as illustrated in Figure 2, employs a systematic ML pipeline specifically designed for detecting ransomware on Android devices. The process begins with data preprocessing, which includes balancing the dataset and performing exploratory data analysis to understand its underlying structure. The dataset is then split into training and test sets to ensure a fair and unbiased evaluation of the model. A hybrid feature selection strategy is employed to refine the feature set, followed by K-fold cross-validation to enhance the modelâĂŹs generalisation capability. To further improve accuracy, we apply hyperparameter tuning for model optimisation. Once the optimal settings are determined, the ML model is trained and subsequently evaluated on unseen data. The results are then examined through XAI analysis and validated experimentally, ensuring the model is not only efficient but also interpretable and reliable for real-world deployment.

3.1 Dataset Description

The dataset utilised in this study is a well-structured compilation of Android network traffic records, specifically curated to aid in the detection and classification of ransomware attacks. It plays a pivotal role as a benchmark in ML-driven cybersecurity research, particularly in identifying malicious behaviours associated with Android ransomware. Comprising a total of 203,556 instances, the dataset includes both benign and ransomware-infected traffic. To ensure a diverse representation, it features samples from ten well-known ransomware families: *SVpeng, PornDroid, Koler, RansomBO, Charger, Simplocker, WannaLocker, Jisut, Lockerpin*, and *Pletor*. Alongside these, benign traffic samples are included to help ML models effectively distinguish between normal and malicious activity. Upon analyzing the dataset closely, we observed a mild imbalance between benign and ransomware-infected traffic records. Out of the 203,556 total instances, around 60% represent benign traffic, while the remaining 40% correspond to ransomware-infected traffic, including samples from 10 ransomware families (such as SVpeng, PornDroid, Koler, and others). Although this imbalance isnâĂŹt particularly severe, we accounted for it carefully during model development to ensure unbiased learning. To maintain representative class distributions, we used stratified train-test splitting, which preserved the proportion of benign and ransomware samples in both sets. Furthermore, during cross-validation, we employed a Stratified K-Fold (K=5) approach to ensure that each fold accurately reflected the overall class distribution, thereby enhancing consistency and reliability. Lastly, some of the ensemble models we employed, including LightGBM and XGBoost, inherently support techniques to manage class imbalance. We leveraged built-in parameters such as scale_pos_weight and class_weight to further reduce potential skew in learning outcomes. Each instance in the dataset is characterised by 85 features that collectively capture intricate patterns of network behaviour. These include:

- **TCP Flags:** Indicators such as SYN, ACK, FIN, and RST that describe the state of a network connection.

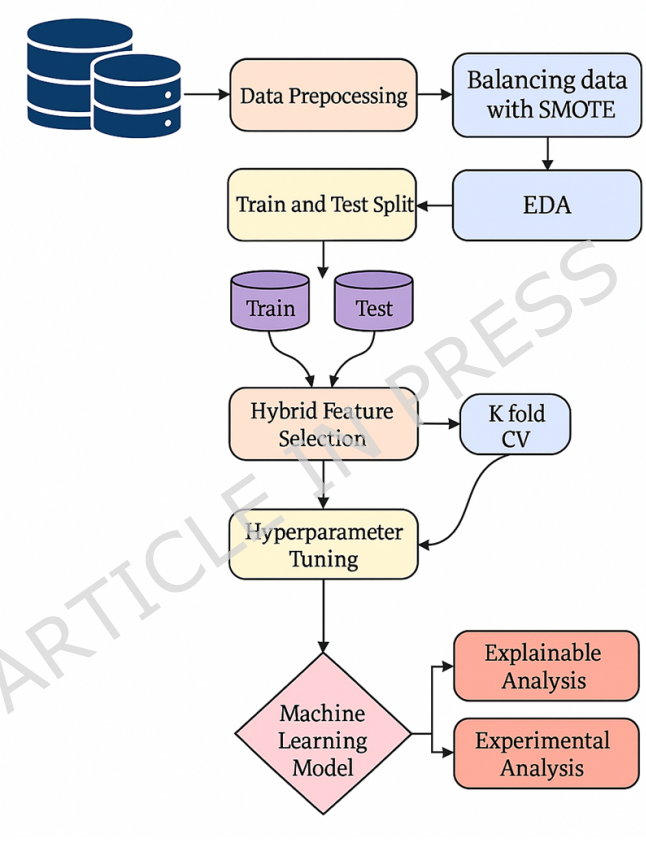

Figure 2. Proposed Framework for Android Ransomware Detection

- **Bulk Transfer Metrics:** Information on forward and backward bulk bytes, packet rates, and data transfer volumes.
- **Activity Durations:** Metrics describing active and idle time intervals.
- **Derived Metrics:** Features like flow bytes per second, packets per second, and down/up ratios, which assist in anomaly detection.

Among the 85 features, 84 are numerical, while the only categorical feature is the `Label`, which denotes whether a record is benign or belongs to a specific ransomware family. This structure ensures compatibility with a wide range of supervised learning algorithms. Despite its strengths, the dataset poses several challenges. First, the dynamic nature of ransomware necessitates that models be regularly retrained to remain effective. Second, not all features contribute equally to detection, necessitating feature selection to optimise performance. Lastly, real-world deployments may face generalisation issues due to variations in devices, user behaviour, and network configurations. To overcome these challenges, our approach incorporates feature selection, hyperparameter tuning, and ensemble learning techniques. These methods collectively enable the development of a scalable, real-time detection system that can adapt to the evolving threat landscape.

Features Description

Table 2 outlines the common features used to analyse network traffic patterns for ransomware detection. These features are carefully selected to capture key aspects of network behaviour, such as IP addresses, port numbers, packet sizes, flow durations, inter-arrival times, and counts of TCP flags. Together, they help the ML model distinguish between benign and malicious traffic. For instance, features like flow duration and packet length can highlight unusual or suspicious data transfers. Meanwhile, metrics such as SYN, ACK, and FIN flag counts provide insight into abnormal connection behaviours, which are often linked to ransomware activities. A key aspect of the feature design is the classification into stateless and stateful types, as shown in the "State" column of Table 2. Stateless features describe the immediate properties of individual packets or flows, without requiring a reference to past traffic. Examples include packet size, protocol type, and flow duration. These are useful for real-time analysis. On the other hand, stateful features consider the sequence and timing of network events. They track the evolution of traffic patterns through attributes such as inter-arrival times, subflow statistics, and active/idle durations. These features enable the model to detect more complex or slow-developing anomalies that may not be evident in isolated packets. By incorporating both stateless and stateful features, the detection framework is equipped to capture a wide range of malicious behaviours, both short-term anomalies and long-term trends, making it more robust in identifying evolving ransomware threats.

3.2 Data Preprocessing

Preparing raw network traffic data is a critical step in building an effective ransomware detection system. The preprocessing phase involves three main stages: data cleaning, data transformation, and data normalisation. Each step ensures that the dataset is accurate, consistent, and ready for ML analysis.

3.2.1 Data Cleaning and Data Transformation

Before training any model, itâĂŹs important to clean the data to ensure its quality and reliability. The initial step involves identifying and removing invalid entries, specifically those with network traffic flows having a destination port of 0. Port 0 is typically reserved and not used in normal network communication. These entries also had a protocol value of 0, reinforcing their invalidity. Removing these flows helped ensure that only meaningful and legitimate traffic data was used, reducing noise and the risk of misleading results. Also checked the dataset for missing values and duplicate records, but fortunately, none were found. This meant the dataset was already in good shape in terms of completeness and uniqueness. Next, focus on transforming the data to make it more suitable for analysis. One key transformation involved standardising the timestamps. Initially, some records had inconsistent time formats, and some even lacked seconds. To fix this, appended missing seconds (as `:00`) and converted all timestamps into a consistent, structured datetime format. Also, split each timestamp into separate âĂIJDateâĂİ and âĂIJTimeâĂİ columns. This not only improved readability but also enabled our models to identify time-related patterns in ransomware behavior, such as repeated attacks at specific hours or during system idle times. These data cleaning and transformation steps ensured that the dataset was both accurate and analytically useful. Refining the raw input helped the ML models focus on meaningful signals, such as irregular spikes in data transmission or suspiciously timed connections, common indicators of ransomware activity. Initially, the dataset exhibited significant class imbalance, as visualized in Figure 3. Such an imbalance can bias the model towards majority classes, reducing detection performance for minority labels. To address this issue, we employed the SMOTE, which generates synthetic samples for underrepresented classes. After applying SMOTE, the class distribution became uniform, as shown in Figure 4. This balancing step was crucial for enhancing model generalization and ensuring a fair evaluation across all ransomware and benign categories.

Table 2. Some of the Feature Descriptions of Network Flow Data

No.	Feature	Description	State
1	Source IP	IP address of source device	Stateless
2	Source Port	Port number on source device	Stateless
3	Destination IP	IP address of destination device	Stateless
4	Destination Port	Port number on destination device	Stateless
5	Protocol	Network protocol (TCP, UDP)	Stateless
6	Flow Duration	Total duration of the flow	Stateless
7	Total Fwd Packets	Total packets sent forward	Stateless
8	Total Bwd Packets	Total packets sent backward	Stateless
9	Total Length of Fwd Packets	Forward packet size total	Stateless
10	Total Length of Bwd Packets	Backward packet size total	Stateless
11	Fwd Packet Length Max	Max forward packet size	Stateless
12	Fwd Packet Length Min	Min forward packet size	Stateless
13	Fwd Packet Length Mean	Mean forward packet size	Stateless
14	Fwd Packet Length Std	Std dev. forward packet size	Stateless
15	Bwd Packet Length Max	Max backward packet size	Stateless
16	Bwd Packet Length Min	Min backward packet size	Stateless
17	Bwd Packet Length Mean	Mean backward packet size	Stateless
18	Bwd Packet Length Std	Std dev. backward packet size	Stateless
19	Flow Bytes/s	Flow rate in bytes per second	Stateless
20	Flow Packets/s	Flow rate in packets per second	Stateless
21	Flow IAT Mean	Mean time between packets	Stateless
22	Flow IAT Std	Std dev. of time between packets	Stateless
23	Flow IAT Max	Max time between packets	Stateless
24	Flow IAT Min	Min time between packets	Stateless
25	Fwd IAT Total	Total time for forward IATs	Stateful
26	Fwd IAT Mean	Mean time between forward packets	Stateful
27	Fwd IAT Std	Std dev. of time between forward packets	Stateful
28	Fwd IAT Max	Max time between forward packets	Stateful
29	Fwd IAT Min	Min time between forward packets	Stateful
30	Bwd IAT Total	Total time for backward IATs	Stateful
31	Bwd IAT Mean	Mean time between backward packets	Stateful
32	Bwd IAT Std	Std dev. of time between backward packets	Stateful
33	Bwd IAT Max	Max time between backward packets	Stateful
34	Bwd IAT Min	Min time between backward packets	Stateful
35	Fwd PSH Flags	Number of PSH flags forward	Stateless
36	Bwd PSH Flags	Number of PSH flags backward	Stateless
37	Fwd URG Flags	Number of URG flags forward	Stateless
38	Bwd URG Flags	Number of URG flags backward	Stateless
39	FIN Flag Count	Number of FIN flags	Stateless
40	SYN Flag Count	Number of SYN flags	Stateless
41	RST Flag Count	Number of RST flags	Stateless
42	PSH Flag Count	Number of PSH flags	Stateless
43	ACK Flag Count	Number of ACK flags	Stateless
44	URG Flag Count	Number of URG flags	Stateless
45	CWE Flag Count	Number of CWE flags	Stateless
46	ECE Flag Count	Number of ECE flags	Stateless
47	Fwd Header Length	Header length forward	Stateless
48	Bwd Header Length	Header length backward	Stateless
49	Fwd Packets/s	Rate of forward packets	Stateless
50	Bwd Packets/s	Rate of backward packets	Stateless
51	Min Packet Length	Min packet length	Stateless
52	Max Packet Length	Max packet length	Stateless
53	Packet Length Mean	Mean packet length	Stateless
54	Packet Length Std	Std dev. of packet length	Stateless
55	Packet Length Variance	Variance of packet lengths	Stateless
56	Down/Up Ratio	Download to upload ratio	Stateless
57	Average Packet Size	Average packet size	Stateless
58	Fwd Segment Size Avg	Avg TCP segment size forward	Stateless
59	Bwd Segment Size Avg	Avg TCP segment size backward	Stateless
60	Fwd Bytes/Bulk Avg	Avg bytes in forward bulk	Stateless
61	Fwd Packets/Bulk Avg	Avg packets in forward bulk	Stateless
62	Fwd Bulk Rate Avg	Avg rate of forward bulk	Stateless
63	Bwd Bytes/Bulk Avg	Avg bytes in backward bulk	Stateless
64	Bwd Packets/Bulk Avg	Avg packets in backward bulk	Stateless
65	Bwd Bulk Rate Avg	Avg rate of backward bulk	Stateless
66	Init Win bytes forward	Initial window size forward	Stateful
67	Init Win bytes backward	Initial window size backward	Stateful
68	act data pkt fwd	Forward packets with data	Stateful
69	min seg size forward	Min TCP segment size forward	Stateful
70	Active Mean	Mean active time	Stateful
71	Active Std	Std dev. of active time	Stateful
72	Active Max	Max active time	Stateful
73	Active Min	Min active time	Stateful
74	Label	Class label (Benign or Ransomware)	Stateless

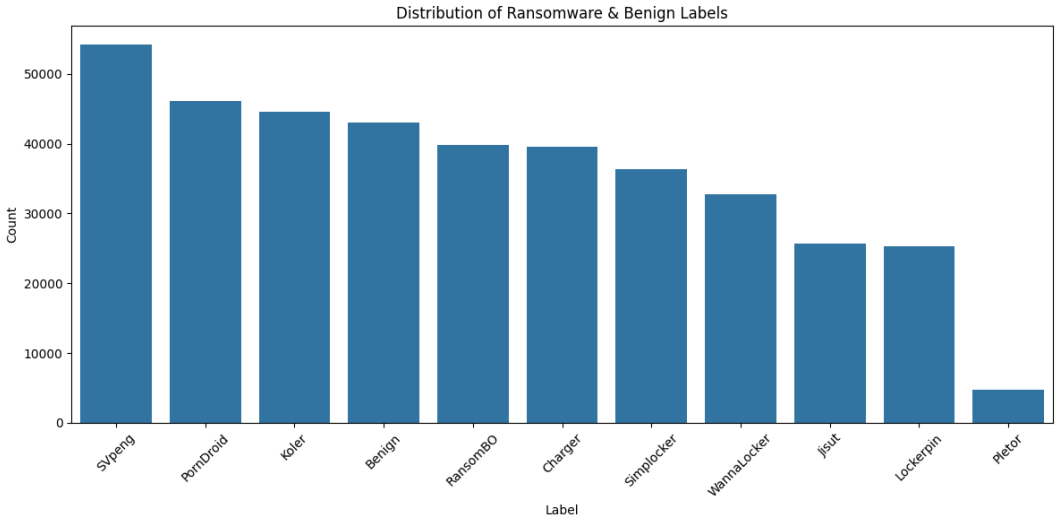

Figure 3. Distribution of original imbalanced ransomware and benign labels.

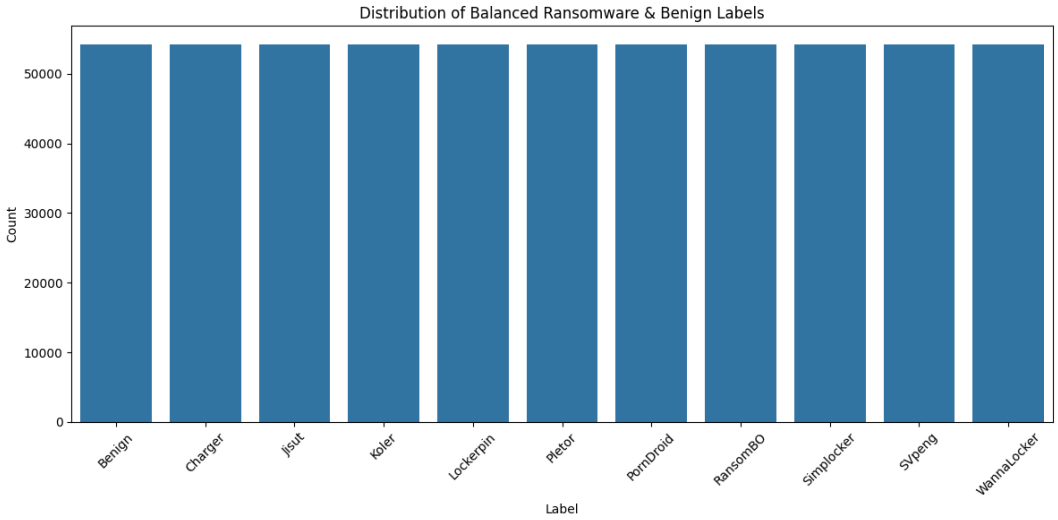

Figure 4. Distribution after applying SMOTE to balance the class labels.

3.2.2 Data Normalisation: StandardScaler vs. RobustScaler

After cleaning and transforming the data, the next step was normalisation. This process ensures that all features contribute equally to the model by bringing them to a common scale. Without normalisation, features with large numeric ranges might dominate the learning process, leading to biased models. Additionally, we compared two common normalization techniques: StandardScaler and RobustScaler. StandardScaler transforms each feature by subtracting the mean and dividing by the standard deviation:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma} \tag{1}$$

Here, X is the original feature value, μ is the mean, and σ is the standard deviation. This method is effective when data is normally distributed, as it centres the data around zero with a standard deviation of one. However, a major drawback of StandardScaler is its sensitivity to outliers. Since the mean and standard deviation can be heavily influenced by extreme values, the resulting scaled data may still be skewed. Figure 5 shows that applying StandardScaler to our dataset led to an uneven spread due to outliers.

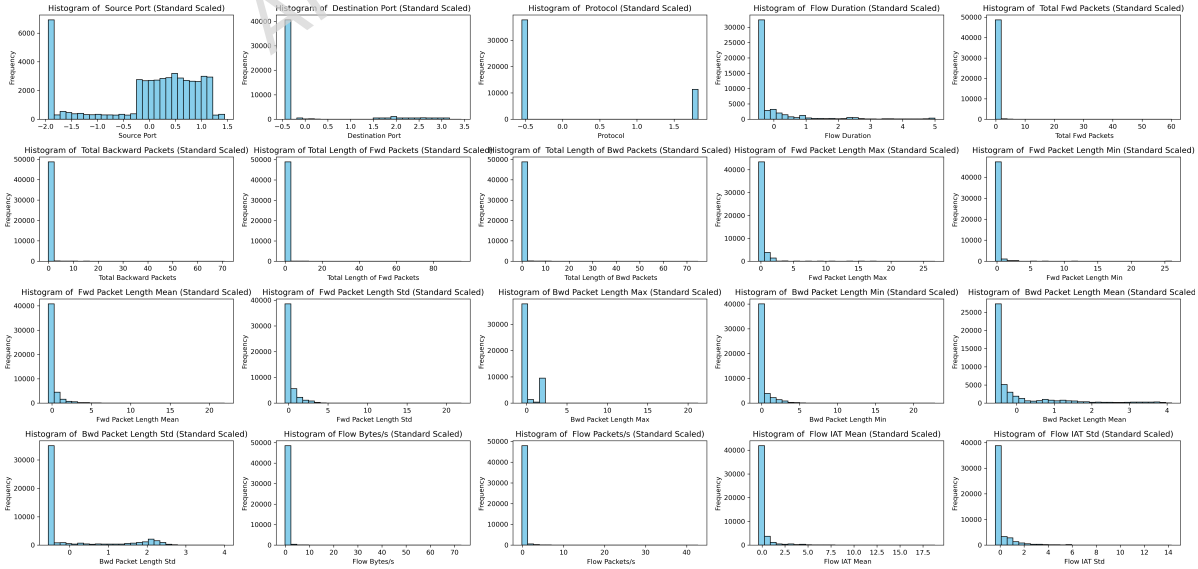

Figure 5. Histogram of StandardScaler Applied to 20 Selected Features

RobustScaler, on the other hand, is specifically designed to handle outliers more effectively. It uses the median and the

interquartile range (IQR), which are less affected by extreme values:

$$X_{\text{scaled}} = \frac{X - \text{median}(X)}{\text{IQR}(X)} \tag{2}$$

$$\text{IQR} = Q3 - Q1 \tag{3}$$

Here, $Q1$ and $Q3$ are the first and third quartiles, respectively. By focusing on the middle 50% of the data, RobustScaler resists the influence of outliers and produces a more balanced scaling outcome. As shown in Figure 6, the transformed features are more evenly distributed.

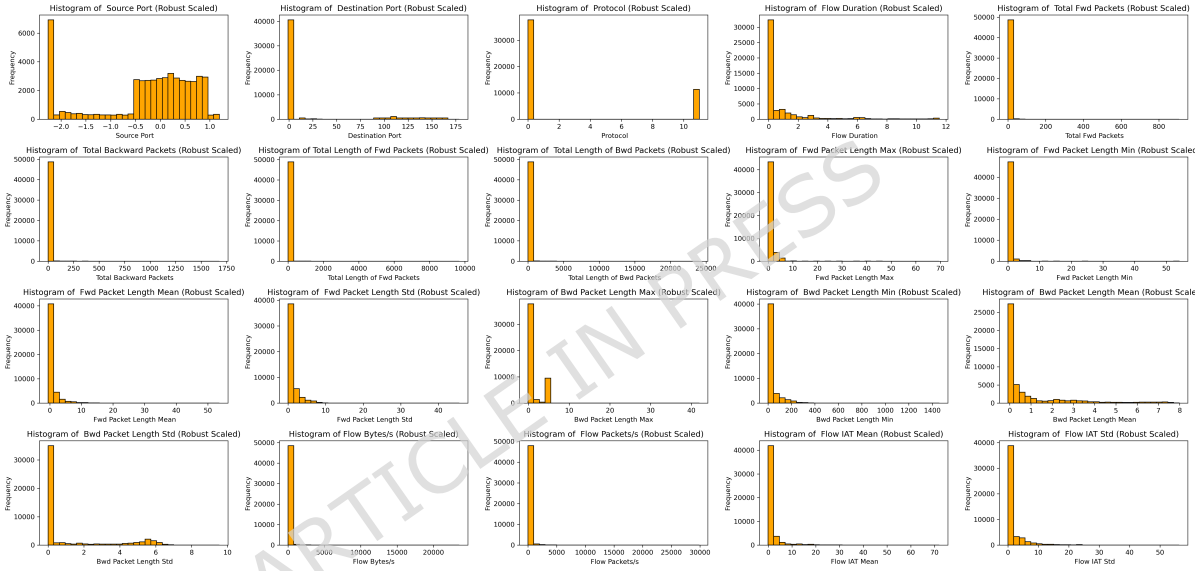

Figure 6. Histogram of RobustScaler Applied to 20 Selected Features

During our analysis, it was noticed that the dataset contained several outliers and non-Gaussian distributions. When applying the StandardScaler, those outliers distorted some features, which could have misled the model during training. In contrast, RobustScaler provided a much more reliable transformation. By using the median and IQR, it preserved the core structure of the data while minimizing the impact of outliers, an especially valuable trait in ransomware detection, where network activity tends to be highly irregular. By choosing RobustScaler, the ML models were trained on well-balanced data, free from the distortion caused by extreme values. This helped improve the modelâĂŹs stability and accuracy, making it a better fit for identifying unpredictable ransomware behavior.

3.3 Exploratory Data Analysis (EDA)

EDA is like getting to know your dataset before diving into model building. It involves summarising, visualising, and understanding the data's key characteristics to uncover hidden patterns, trends, and anomalies. EDA, using techniques such as boxplots, histograms, and correlation matrices, provides a clearer picture of the data landscape, enabling us to make more informed decisions when selecting features, choosing models, and tuning preprocessing techniques.

3.3.1 Boxplot for Numerical Features

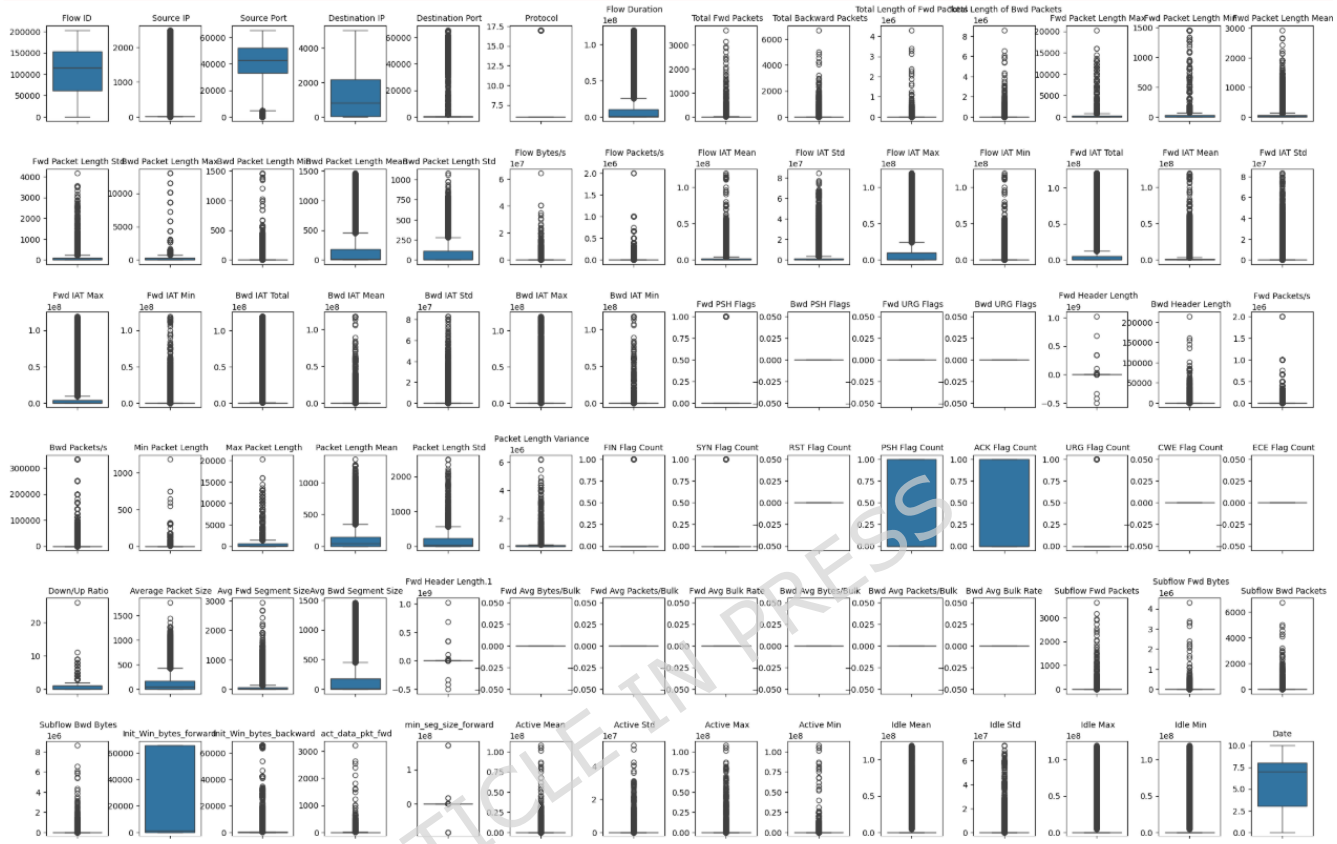

Figure 7. Boxplot visualisation of numerical network traffic features

Figure 7 showcases boxplots for various numerical features from the network traffic dataset, an essential step for understanding how data is spread out, where the outliers lie, and what kind of scaling or transformation might be needed. These plots reveal interesting trends: many features, such as packet sizes, flow durations, and inter-arrival times (IAT), have skewed distributions. In other words, most data points cluster at one end, with a few outliers stretching far out, forming long whiskers. This isn't surprising when dealing with network traffic. Every day, online activity typically involves many short, frequent connections, occasionally interrupted by large bursts of data, such as during software updates or streaming. However, what's particularly important for our ransomware detection research is that ransomware behaves differently. It tends to generate unusual traffic patterns, like sudden spikes, abnormal packet sizes, or rapid bursts of encrypted data transfers. These abnormalities often show up as outliers. Noticing these outliers can provide a strong indication of suspicious activity. For instance, high values for features such as PSH Flag Count or ACK Flag Count might indicate unusual connection behaviour. These insights guide the choice of ML models, such as Random Forest, XGBoost, or LightGBM, which can handle such irregularities more effectively than SVMs, which are more sensitive to data distribution and scaling. In the context of ransomware, where traffic can change rapidly, we may also benefit from combining supervised learning with anomaly detection techniques to achieve even better results.

3.3.2 Radviz Visualisation for Ransomware Detection

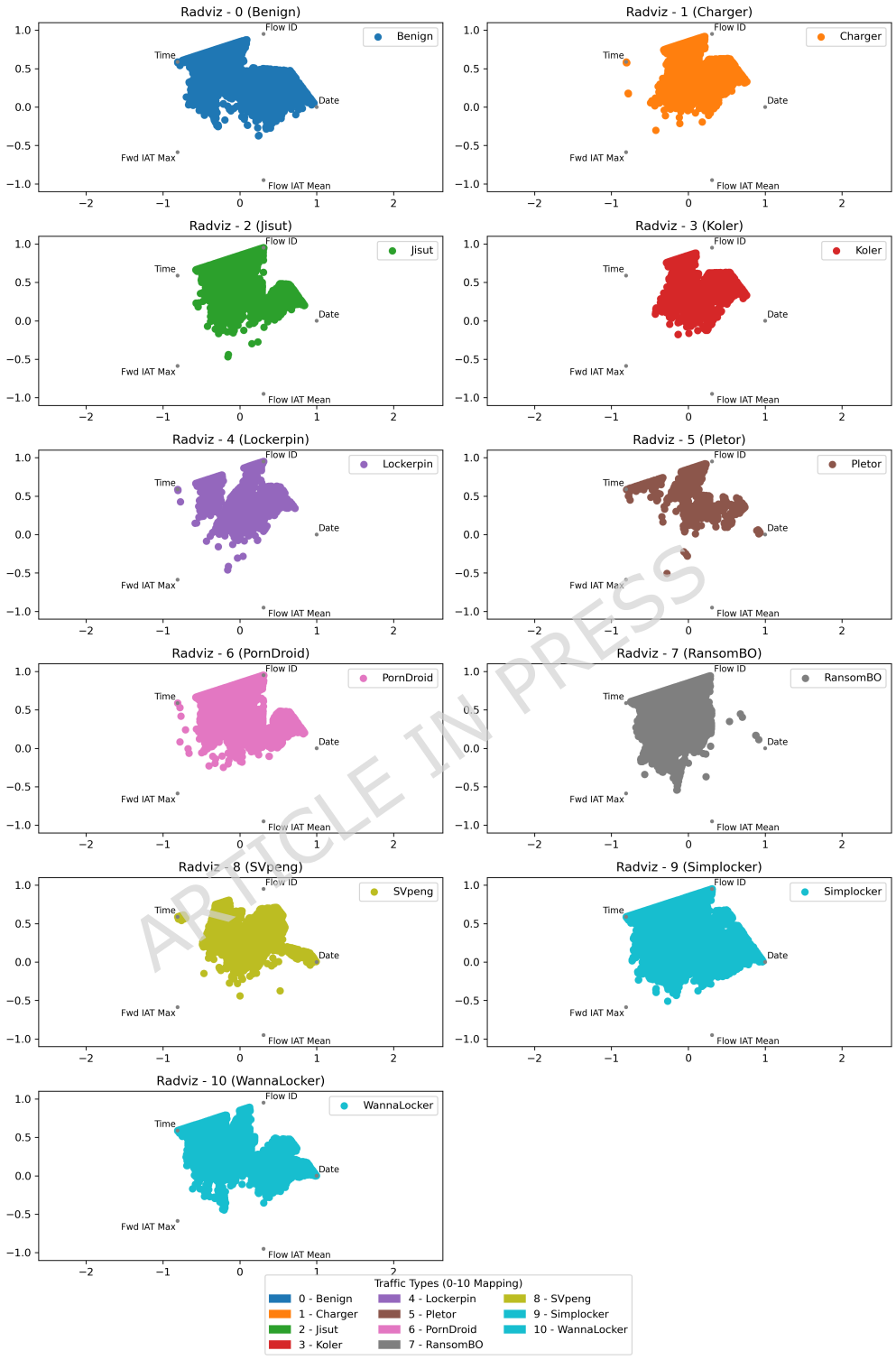

Figure 8. Radviz visualisation of network traffic for ransomware detection

To make sense of how different features interact in high-dimensional data, used a technique called Radviz (Radial Visualisation). It helps visualize multidimensional data in a two-dimensional space, making it easier to spot patterns and clusters. Initially, the dataset was narrowed down to the top five features with the strongest correlation with the target label (malicious or benign).

These were: Flow ID, Time, Date, Fwd IAT Max, and Flow IAT Mean. Using these, we plotted a Radviz visualisation shown in Figure 8, which offers a fascinating view of how ransomware traffic differs from normal traffic. Benign network traffic tends to form tight clusters, indicating stable, predictable behavior. In contrast, ransomware traffic was much more scattered. This variation makes sense because different ransomware families exhibit distinct behaviors. Some ransomware types, such as Koler and Pletor, clustered closely together, exhibiting similar patterns, while others, like Lockerpin and WannaLocker, were more dispersed, reflecting diverse behaviors and making them trickier to detect. This visualisation gives security analysts a powerful tool. It helps not only to identify which features are most useful for classification but also to understand how consistent (or inconsistent) ransomware traffic can be. These insights are crucial for building ML models that accurately distinguish between normal and malicious behaviour in real time.

3.4 Hybrid Feature Selection

Hybrid feature selection combines the best aspects of multiple methods to identify the most useful features for an ML model, as shown in Figure 9. Instead of relying on just one method, it combines filtering, wrapping, and embedding techniques to gain a well-rounded view of which features truly matter. Filter methods are fast and efficient, wrapper methods test feature combinations by actually training models, and embedded methods find important features as part of the model training itself. By combining these approaches, the hybrid method helps remove irrelevant or redundant features, making the model faster, more accurate, and easier to interpret. In this research, the hybrid process begins with Mutual Information (MI), a powerful filter-based technique. MI measures how much one variable (such as a feature) provides information about another (like the target label) and is quantifiable. MI is valuable because it captures both linear and nonlinear relationships, particularly when dealing with complex data such as network traffic. The formal definition of MI is:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{4}$$

Other strong features included "Source IP" (0.86), "Destination IP" (0.46), and "Source Port" (0.20), emphasising how specific traffic paths and sources can help identify malicious behaviour.ration). The higher the mutual information, the more useful that feature is for predicting outcomes.

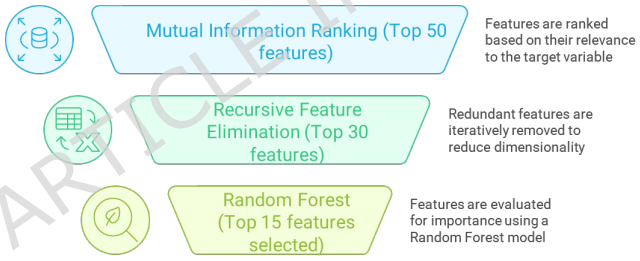

Figure 9. Hybrid feature selection

In the dataset, the MI scores indicated that time-related and flow-specific features are most predictive. As shown in Table 3, "Date" had the highest score, followed by "Flow ID" and "Time". These features likely help capture the temporal and sequence-based patterns typical of ransomware activity. Other strong features included "Source IP", "Destination IP", and "Source Port", highlighting how specific traffic paths and sources can help identify malicious behavior. On the other hand, features such as "Min Packet Length" and "minsegsizeforward" scored very low, indicating that they contribute little to the detection process. Identifying and excluding such features helps simplify the model, reduce training time, and avoid overfitting. This hybrid approach to feature selection plays a crucial role in building a robust and efficient ransomware detection system, ensuring the model focuses on what truly matters while disregarding noise.

Figure 10 clearly shows that after the top few features, the importance scores drop off sharply. This tells us that just a small group of features carries most of the predictive power, while the rest add little value. By removing these less relevant features, the modelâĂŹs accuracy can be maintained while also making it faster and less complex to run. Building on this, we explored a wrapper method to further fine-tune the feature selection. After using MI as a filter to select the most promising features, Recursive Feature Elimination (RFE), a wrapper technique, was applied to refine the selection by repeatedly removing the least important features. Essentially, RFE trains the model multiple times, dropping the weakest features at each iteration until only the most impactful ones remain. Choose RFE because itâĂŹs especially effective when dealing with

many features. It helps reduce the number of variables, making the model easier to interpret and more efficient to train. For this process, a Random Forest Classifier was used as the model for RFE because it is strong at capturing complex feature relationships and reliably ranks feature importance. To balance precision and efficiency, set RFE to keep the top 30 features (`n_features_to_select=30`) and remove features in groups of 10 per iteration (`step=10`). This approach accelerates the process while ensuring the model remains powerful and focused on the most meaningful data.

Table 3. Feature importance scores using Mutual Information

Features	Score
Date	1.490564
Flow ID	1.252700
Time	0.865615
Source IP	0.865075
Destination IP	0.467900
Source Port	0.209444
Init_Win_bytes_forward	0.157167
Flow Duration	0.145864
Fwd Packet Length Max	0.145529
Flow IAT Max	0.144739
Total Length of Fwd Packets	0.131908
Subflow Fwd Bytes	0.131823
Flow IAT Min	0.129040
Init_Win_bytes_backward	0.124722
Fwd Packets/s	0.124096
Flow IAT Mean	0.122892
Average Packet Size	0.121301
Packet Length Std	0.120165
Flow Packets/s	0.119478
Packet Length Mean	0.113678
Max Packet Length	0.109858
Avg Fwd Segment Size	0.109608
Packet Length Variance	0.109571
Fwd IAT Min	0.107366
Fwd Packet Length Mean	0.104860

Features	Score
Destination Port	0.104766
Fwd IAT Max	0.103042
Subflow Bwd Bytes	0.099683
Fwd Packet Length Std	0.097581
Fwd IAT Mean	0.097466
Fwd IAT Total	0.097147
Total Length of Bwd Packets	0.096761
Avg Bwd Segment Size	0.096668
Bwd Packet Length Mean	0.089966
Bwd Packets/s	0.089556
Fwd Header Length.1	0.077178
Fwd Header Length	0.072362
Bwd Packet Length Std	0.069176
Bwd Packet Length Max	0.063816
Flow IAT Std	0.061321
Flow Bytes/s	0.058540
Bwd Header Length	0.049468
Bwd IAT Min	0.045177
Bwd IAT Total	0.044532
Bwd IAT Max	0.039576
Fwd IAT Std	0.039000
Fwd Packet Length Min	0.037430
Bwd Packet Length Min	0.037153
Min Packet Length	0.033035
min_seg_size_forward	0.031758

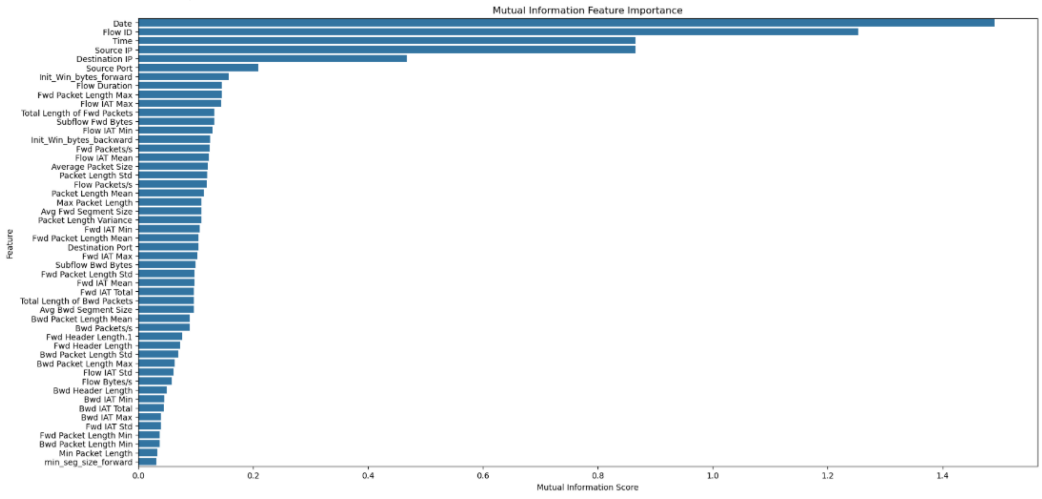

Figure 10. Mutual Information Feature Importance

The visualised feature rankings in Figure 11 demonstrate how RFE assigns a numerical ranking to each feature, where a

lower value indicates higher importance. This helps in identifying the most relevant features for ransomware detection, ensuring that only the most impactful attributes contribute to model predictions. By using RFE, we achieved better generalisation, reduced overfitting, and enhanced model performance, making it a suitable choice for this research. From the image, one can infer that the top-ranked features are most significant in predicting ransomware activity and should be retained in the final model. The selected 30 best features include network traffic attributes such as packet length statistics, inter-arrival times, and flow-based metrics, which are critical for detecting ransomware behaviour. Recursive elimination effectively filters out irrelevant or less influential features, simplifying the dataset and improving classification performance. The bar plot visually confirms the ranking hierarchy, making it easy to identify which features contribute most to model predictions and guiding future feature engineering and optimisation. Finally, the embedded methods combine feature selection with model training, meaning the algorithm selects important features as it builds the model. These methods are computationally efficient because they automatically remove less important features, reducing dimensionality and improving performance. Unlike filter and wrapper methods, embedded methods integrate feature selection into the learning process, ensuring that only the most relevant features contribute to the final model, thereby improving generalisation and reducing overfitting.

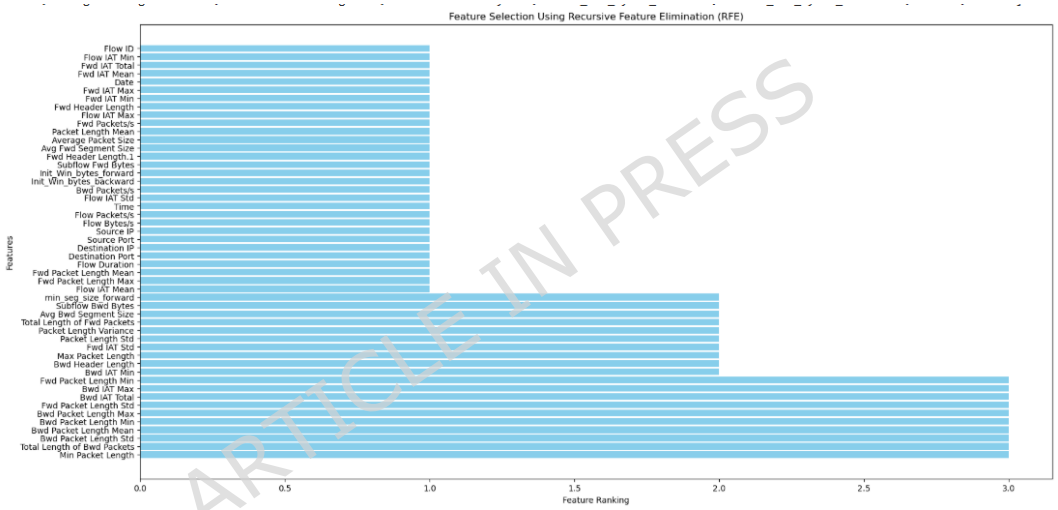

Figure 11. Feature Selection Using Recursive Feature Elimination

Random Forest is an ensemble learning method that builds multiple decision trees and averages their predictions. It inherently ranks features based on their contribution to reducing impurity in decision trees, making it a powerful embedded feature selection technique. Choose Random Forest because it effectively handles large, high-dimensional datasets, making it suitable for real-world network traffic analysis, where data can be extensive and complex. Moreover, its robustness to overfitting, due to averaging across multiple trees, ensures that the model generalizes well to unseen data. Another advantage is that it provides feature importance scores, allowing us to identify and retain the most relevant features for analysis while discarding noisy or redundant variables. The feature importance plot Figure 12 visualizes the top 15 features ranked by their importance scores, highlighting the most influential attributes in the dataset. The "Date" feature is most important, followed by "Source IP", "Time", and "Flow ID", suggesting their strong influence on model predictions. This suggests that time-based attributes and source identifiers play a crucial role in distinguishing among the dataset's classes.

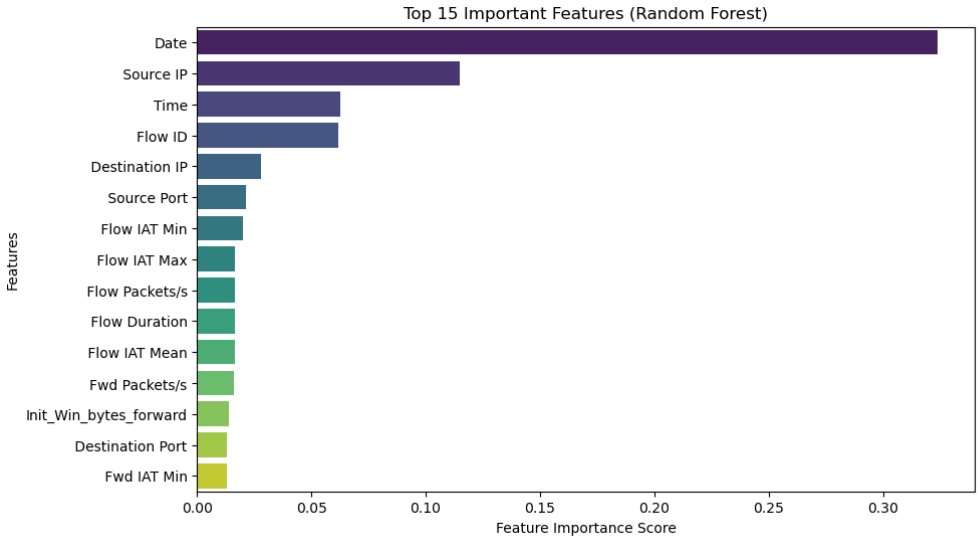

Figure 12. Top 15 Important Features (Random Forest)

Table 4 compares the features selected by three different methods: MI (a filter method), RFE (a wrapper method), and Random Forest (an embedded method). The results show a strong overlap between these methods, confirming their reliability in identifying the key features for ransomware detection. Features like Flow ID, Source IP, Destination IP, Flow Duration, Fwd Packet Length Max, Flow IAT Mean, and Date appear consistently across all three methods. These features are crucial for distinguishing between benign and ransomware network traffic, as they capture important aspects such as the direction of data flow, packet timing, and the interactions between sources and destinations. The strong agreement among the methods demonstrates their effectiveness in filtering out irrelevant features while keeping the most predictive ones. This consensus helps improve detection accuracy, reduce computational complexity through dimensionality reduction, prevent overfitting, and make the model more interpretable. Additionally, the inclusion of both temporal features (such as Date and Time) and flow-specific features (such as Flow ID and Flow Duration) underscores the importance of capturing network behavior across both short- and long-term periods, a crucial factor for accurate ransomware detection. Table 5 shows a consensus-based feature selection matrix.

Table 4. Feature Selection Methods and Selected Features

Mutual Information (Filter)	**Recursive Feature Elimination (Wrapper)**	**Random Forest (Embedded)**
Flow ID	Flow ID	Flow ID
Source IP	Source IP	Source IP
Source Port	Source Port	Source Port
Destination IP	Destination IP	Destination IP
Destination Port	Destination Port	Destination Port
Flow Duration	Flow Duration	Flow Duration
Total Length of Fwd Packets		
Total Length of Bwd Packets		
Fwd Packet Length Max	Fwd Packet Length Max	
Fwd Packet Length Mean	Fwd Packet Length Mean	
Flow Bytes/s	Flow Bytes/s	
Flow Packets/s	Flow Packets/s	Flow Packets/s
Flow IAT Mean	Flow IAT Mean	Flow IAT Mean
Flow IAT Max	Flow IAT Max	Flow IAT Max
Flow IAT Min	Flow IAT Min	Flow IAT Min
Fwd IAT Total	Fwd IAT Total	
Fwd IAT Mean	Fwd IAT Mean	
Fwd IAT Max	Fwd IAT Max	
Fwd IAT Min	Fwd IAT Min	Fwd IAT Min
Fwd Header Length	Fwd Header Length	
Fwd Packets/s	Fwd Packets/s	Fwd Packets/s
Bwd Packets/s	Bwd Packets/s	
Packet Length Mean	Packet Length Mean	
Average Packet Size	Average Packet Size	
Avg Fwd Segment Size	Avg Fwd Segment Size	
Fwd Header Length.1	Fwd Header Length.1	
Subflow Fwd Bytes	Subflow Fwd Bytes	
Init_Win_bytes_forward	Init_Win_bytes_forward	Init_Win_bytes_forward
Init_Win_bytes_backward	Init_Win_bytes_backward	
Date	Date	Date
Time	Time	Time

Table 5. Consensus-Based Feature Selection Matrix (Selected if chosen by ≥ 2 methods)

Feature	Mutual Info (MI)	RFE	Random Forest (RF)	Selected
Average Packet Size	✓	✓	✓	**Yes**
Avg Fwd Segment Size	✓	✓	✓	**Yes**
Bwd Packets/s	✓	✓		**Yes**
Date	✓	✓	✓	**Yes**
Destination IP	✓	✓	✓	**Yes**
Destination Port	✓	✓	✓	**Yes**
Flow Bytes/s	✓	✓		**Yes**
Flow Duration	✓	✓	✓	**Yes**
Flow IAT Max	✓	✓	✓	**Yes**
Flow IAT Mean	✓	✓	✓	**Yes**
Flow IAT Min	✓	✓	✓	**Yes**
Flow Packets/s	✓	✓	✓	**Yes**
Fwd Header Length	✓	✓		**Yes**
Fwd IAT Max	✓	✓		**Yes**
Fwd IAT Min	✓	✓	✓	**Yes**
Fwd Packet Length Max	✓	✓		**Yes**
Fwd Packet Length Mean	✓	✓		**Yes**
Fwd Packets/s	✓	✓	✓	**Yes**
Init_Win_bytes_forward	✓	✓	✓	**Yes**
Source IP	✓	✓	✓	**Yes**
Source Port	✓	✓	✓	**Yes**
Time	✓	✓	✓	**Yes**

3.5 Data Splitting and Cross Validation

Splitting the dataset into training and testing sets is an essential step in ML. The training set is used to teach the model, while the testing set evaluates how well the model performs on unseen data. This helps ensure that the model learns general patterns rather than just memorising the data. In this work, we use an 80/20 split, meaning 80% of the data is used for training and 20% for testing. This ratio is widely used for large datasets because it provides enough data for the model to learn meaningful insights while still reserving a solid portion to fairly evaluate performance. To ensure consistent results across multiple runs, `random_state=42` was set, guaranteeing the same split every time the code is executed. `stratify=y` was also used to preserve the original class distribution in both the training and testing sets. This is especially important for classification problems, as it helps avoid imbalanced datasets that could skew the modelâĂŹs accuracy. Some of the common train-test split ratios and when to use them:

- **80/20 Split**: Ideal for large datasets, balancing training size and evaluation.
- **70/30 Split**: Useful when more validation data is needed to better assess performance.
- **60/40 or 50/50 Split**: Typically for small datasets, where a larger test set ensures reliable evaluation.

If the dataset has N samples, the sizes of the training and testing sets are:

$$\text{Training Set Size} = N \times (1 - \text{test\_size})$$
$$\text{Testing Set Size} = N \times \text{test\_size}$$

The 80/20 split was chosen here to strike a good balance: it provides the model with ample data to learn from while reserving enough unseen data for a meaningful evaluation. Since the dataset is large, this approach reduces the risk of underfitting and allows the model to generalise better. Stratified sampling further ensures that class proportions remain consistent across both sets, reducing bias and making the evaluation more reliable. Overall, this data-splitting strategy helps create a robust workflow that enhances the modelâĂŹs accuracy and reliability while preventing common pitfalls such as overfitting and underfitting.

3.5.1 K-Fold Cross Validation

Stratified K-Fold Cross-Validation is a popular technique for evaluating ML models more reliably by ensuring that the class distribution is preserved across folds. Unlike regular K-Fold Cross-Validation, where data is split randomly, stratification

maintains the same class proportions in each fold as in the original dataset. This is especially important when dealing with imbalanced datasets, as it prevents biased or misleading performance estimates. In this research, $K = 5$, meaning the dataset is divided into 5 equal parts. The model is then trained and validated five times, each time using a different fold for validation and the remaining four folds for training. This process helps reduce variance in evaluation results and provides a more robust estimate of how the model will perform on unseen data. The size of each fold is calculated as follows:

$$\text{Fold Size} = \frac{N}{K} \tag{5}$$

The figure 13 illustrates this process, where each row represents a different iteration of cross-validation. The blue sections indicate training data, while the orange sections represent validation data. The stratification ensures that all classes are proportionally distributed across folds, leading to fair model evaluation and more robust results.

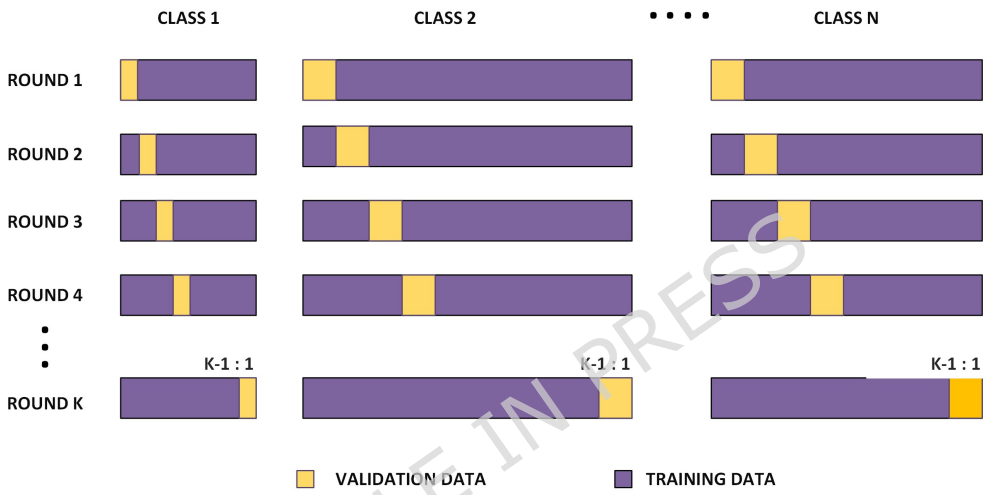

Figure 13. Visualisation of Stratified K-Fold Cross-Validation

3.6 Hyperparameter Tuning

Hyperparameter tuning plays a crucial role in this research by enhancing model performance, reducing overfitting, and improving accuracy. Fine-tuning these parameters ensures that the model generalises well to new, unseen data, avoiding being either too simple or overly complex. In this study, both Random Search and Grid Search methods were employed, combined with Stratified K-Fold Cross-Validation ($K = 5$), to systematically explore and identify the best hyperparameter settings. Since ransomware detection demands high accuracy and reliable generalisation, carefully selecting hyperparameters is crucial. Random Search quickly scans a broad range of hyperparameter combinations by sampling randomly, making it efficient when the search space is large. Once promising regions are identified, Grid Search is applied to exhaustively test all combinations within that narrowed space to find the optimal parameters. To ensure robustness, Stratified K-Fold Cross-Validation maintains the class distribution across folds, preventing bias and ensuring consistent model performance across different subsets of the data. The general formula for Random Search can be expressed as:

$$\theta^* = \arg\max_{\theta \in S} M(f_\theta(X), y) \tag{6}$$

where:

- θ is a randomly selected hyperparameter set from the search space S,
- M is the evaluation metric (e.g., accuracy, F1-score),
- $f_\theta(X)$ denotes the model trained with hyperparameters θ,

- y is the true target variable.

Similarly, Grid Search optimises over the complete grid G of parameter combinations:

$$\theta^* = \arg\max_{\theta \in G} M(f_\theta(X), y) \tag{7}$$

After applying Random Search followed by Grid Search, the best hyperparameters were selected, and the final model was trained. The minimal difference in F1-score between training and testing sets indicates that overfitting was effectively controlled. This careful tuning makes the ransomware detection model robust and reliable for real-world applications.

3.7 Machine Learning Models for Ransomware Detection

ML models play a pivotal role in this study, enabling accurate and efficient ransomware detection. The models employed, LightGBM, Random Forest, Decision Tree, SVM, and XGBoost, each offer distinct advantages in terms of accuracy, interpretability, and computational efficiency. By integrating these diverse algorithms, the study achieves a balance between real-time detection performance, robustness, and adaptability to various ransomware patterns.

LightGBM (Light Gradient Boosting Machine) is a highly efficient gradient boosting algorithm tailored for large datasets and real-time applications. Unlike traditional methods, it employs histogram-based learning, which discretises continuous values into bins, thereby accelerating training and reducing memory usage. Another notable feature is its native support for categorical features, minimising the need for extensive preprocessing. This makes LightGBM particularly suitable for time-sensitive scenarios, such as ransomware detection.

One of the core innovations in LightGBM is its leaf-wise tree growth strategy, which typically yields deeper trees with lower loss compared to the level-wise approach. This design enhances performance in complex classification tasks. Moreover, LightGBM supports parallel learning and GPU acceleration, offering scalability for large-scale deployments. The model is trained using gradient boosting, represented by:

$$F_m(X) = F_{m-1}(X) + \gamma_m h_m(X) \tag{8}$$

Random Forest is an ensemble learning method that builds multiple decision trees and aggregates their outputs to improve accuracy and reduce overfitting. By averaging predictions from diverse trees trained on bootstrapped subsets of the data, Random Forest enhances generalisation and robustness. This makes it particularly effective for cybersecurity tasks that involve balancing and imbalanced datasets. Additionally, Random Forest provides feature importance metrics that aid interpretability and help identify the most relevant features for detecting ransomware. This transparency is essential in security-focused applications. The general form of the modelâĂŹs prediction is:

$$f(X) = \frac{1}{T} \sum_{t=1}^{T} h_t(X) \tag{9}$$

Decision Trees offer a straightforward yet powerful approach to classification. These models partition the dataset using feature-based decision rules, forming a tree-like structure. Each internal node represents a feature, branches indicate the possible values of that feature, and leaves correspond to the output classes. The simplicity and transparency of this model make it ideal for scenarios requiring clear decision logic. Although prone to overfitting, especially when the tree grows too deep, techniques such as pruning (e.g., cost-complexity pruning) can mitigate this issue. Decision Trees also require minimal preprocessing, enabling fast deployment in real-time environments. The information gain criterion, used to decide splits, is defined as:

$$IG = H(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} H(S_i) \tag{10}$$

Support Vector Machine (SVM) is a robust classification algorithm that identifies the optimal hyperplane separating different classes. For data that is not linearly separable, SVM applies kernel functions (such as the Radial Basis Function) to project the data into higher-dimensional spaces where separation is possible. SVM is especially useful in high-dimensional settings and performs well when the number of features exceeds the number of observations. It also effectively handles outliers, making it suitable for detecting ransomware variants with anomalous behaviour. The optimisation problem solved by SVM is:

$$\min_{w,b} \frac{1}{2}||w||^2 \quad \text{subject to } y_i(w \cdot X_i + b) \geq 1 \tag{11}$$

XGBoost (Extreme Gradient Boosting) enhances traditional gradient boosting through regularisation, parallelisation, and advanced tree-pruning strategies. It is known for its high predictive power and computational efficiency, making it a frequent winner in ML competitions. In ransomware detection, XGBoost excels in minimising both bias and variance, ensuring high accuracy and robustness. A key advantage of XGBoost is its ability to handle missing values internally and its use of early stopping to prevent overfitting and save computation. These features make it highly efficient in real-time threat detection scenarios. Its learning objective incorporates both a loss function and a regularisation term:

$$L(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k} \Omega(f_k) \tag{12}$$

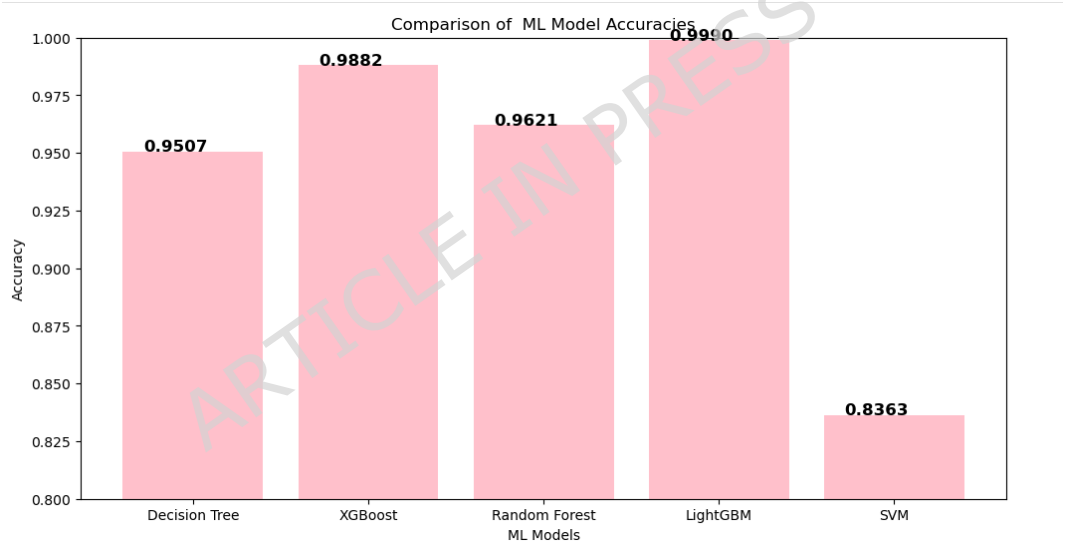

Figure 14. Comparison of ML Model Accuracies

As illustrated in Figure 14, LightGBM outperformed other models, achieving the highest accuracy. These findings highlight the effectiveness of boosting algorithms, such as LightGBM and XGBoost, for accurately detecting ransomware threats in real time. Random ForestâĂŹs strong performance further supports the value of ensemble methods in constructing reliable detection systems. While Decision Trees offer high interpretability, they may suffer from overfitting. SVM, although robust in high-dimensional feature spaces, demonstrated comparatively lower accuracy. The characteristics of these models' accuracy, interpretability, and their capacity to manage data imbalance are summarised in Table 6. This comparison provides insights into the strengths and limitations of each method, informing their effective application in various cybersecurity contexts. Figure 15 shows the Precision-Recall curves for five different classifiers: Decision Tree, SVM, Random Forest, XGBoost, and LightGBM. These curves help evaluate model performance, especially on imbalanced datasets, by showing the trade-off between precision and recall. Models like LightGBM and XGBoost exhibit superior performance with higher precision at higher recall values. SVM shows comparatively lower performance across the curve. Overall, the Precision-Recall curve analysis is crucial for identifying the best-performing model across both precision and recall.

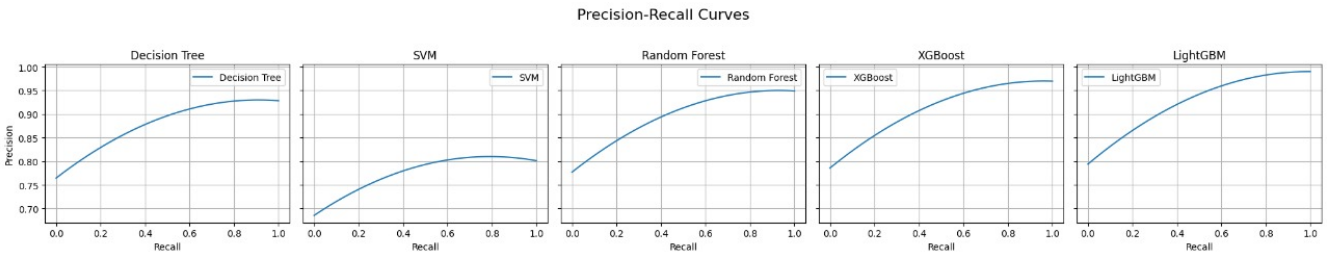

Figure 15. Precision-Recall Curves of five different classification models

Table 6. Comparison of Machine Learning Models

Model	Why Used	Acc.	CV Acc.	Handling Imbalance	Interpretability
Decision Tree	Simple, easy to understand	95.1%	95.4%	Poor-prone to bias	High-easily interpretable
SVM	Good for high-dimensional data	83.6%	82.3%	Moderate-class weighting helps	Low-complex decision boundaries
Random Forest	Reduces variance, improves accuracy	96.2%	96.0%	Good-handles imbalance with bootstrapping	Moderate-feature importance available
XGBoost	Optimized for speed, performance	98.8%	97.9%	Very good-built-in imbalance handling	Low-complex but interpretable
LightGBM	Faster and more efficient boosting	99.9%	99.9%	Excellent-built-in handling	Low-hard to interpret

Ultimately, LightGBM was selected for its real-time detection capabilities and minimal computational demands. Random Forest provided a robust, generalizable solution. Decision Trees provided transparency, while SVM added strength in high-dimensional data environments. XGBoost brought exceptional predictive power and model efficiency. The combination of these models provides a well-rounded, adaptable approach to ransomware detection. We conducted a comparative analysis of the model's performance with and without feature selection, using the same preprocessing steps, train-test split, and hyperparameter tuning across both settings, as shown in Table 7. The comparison illustrates the impact of feature selection on model accuracy, training efficiency, and overfitting. We used LightGBM, our best-performing model, and evaluated it on the following metrics: Accuracy, Precision, Recall, F1-Score, and Training Time.

Table 7. Comparative performance of LightGBM with and without feature selection

Metric	Without Feature Selection	With Feature Selection
Accuracy	97.12%	98.84%
Precision	96.80%	98.72%
Recall	97.05%	98.84%
F1-Score	96.92%	98.67%
Training Time	84.6 sec	51.3 sec
Model Complexity	High (85 features)	Reduced (15 features)
Risk of Overfitting	Moderate	Lower (due to reduced dimensionality)

3.8 Explainability and Experimental Analysis

In ML, explainability is essential for understanding and interpreting the decisions made by predictive models. This is particularly critical in high-stakes domains, such as cybersecurity, where the consequences of incorrect decisions can be severe.

Explainability helps uncover the rationale behind model predictions, making it easier to debug errors, detect biases, and enhance overall model transparency. It also fosters trust among stakeholders by providing a window into the model's decision-making process.

In this research, we used SHAP and LIME for explainability analysis. SHAP determines which features most influence the modelâĂŹs prediction of a model. It is based on an idea from game theory, where credit is shared fairly among players, in this case, features. SHAP checks how the prediction changes when you include or exclude a feature, considering all possible combinations. This helps us understand whether a feature is pushing the prediction higher or lower. It provides clear scores that show the impact of each feature, making the model's behavior more transparent and understandable. LIME works by focusing on a single prediction and building a simple, interpretable model around that point. It tweaks the input a little, observes how the modelâĂŹs output changes, and then identifies which features led to that prediction. It doesnâĂŹt rely on the model's internal structure, so it works with any type of ML algorithm. LIME is especially helpful when you want to explain why the model made a particular decision for a specific input.

Figure 16 illustrates an explainability analysis conducted using LIME, a technique designed to interpret individual predictions of complex ML models. LIME works by slightly perturbing the input data and observing how the prediction changes, thereby highlighting the most influential features behind a given classification. In the upper section of the figure, the model's predicted probabilities for the different classes are displayed, providing insight into its confidence in the final decision. The right-hand side of the graphic emphasises the most influential features, identifying which elements drove the model toward a specific classification. The lower portion of the figure provides a detailed and color-coded representation of how each feature influenced the outcome. Features shown in green positively contributed to the predicted class, while those in red had a negative impact. Larger bars indicate greater influence, providing a straightforward interpretation of the model's internal mechanics in predicting. This analysis reveals that certain features can significantly influence the model's decision, either increasing or decreasing the likelihood of a given classification. It also suggests the presence of overlapping or ambiguous patterns in the data, as indicated by non-extreme prediction probabilities. By analysing these feature contributions, we can identify model weaknesses and areas for improvement, ultimately leading to more robust and interpretable models.

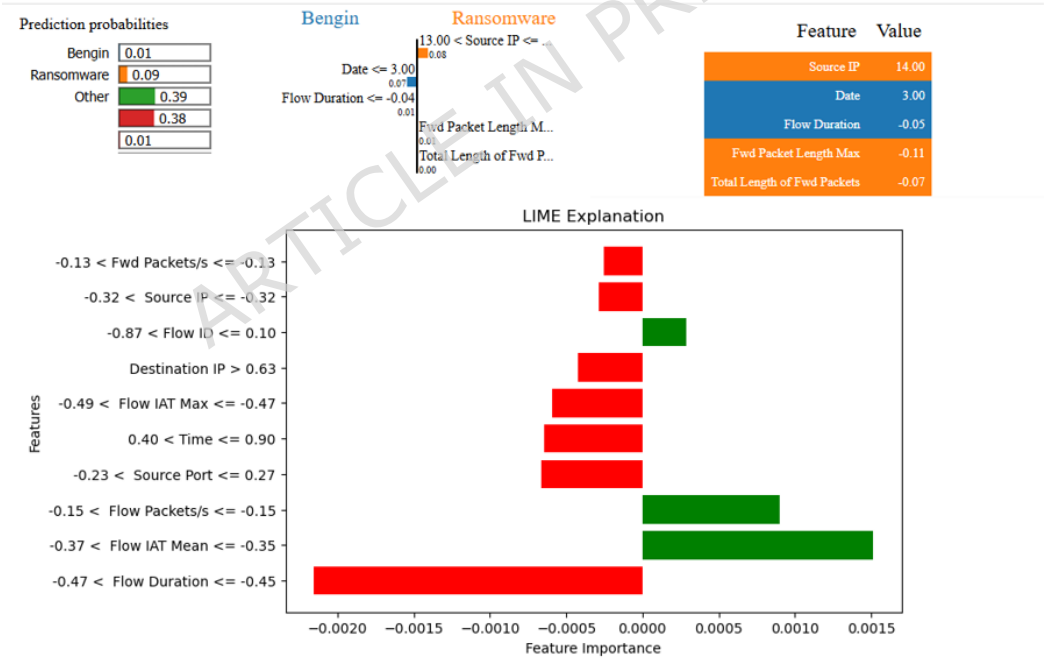

Figure 16. Explainability analysis using LIME. The visual highlights feature contributions to the classification decision.

Experimental analysis plays a vital role in assessing the performance and reliability of ML models. It involves training the model on historical data, testing it on previously unseen data, and evaluating the results using a set of defined metrics such as accuracy, precision, recall, and F1-score. These metrics help determine how well the model generalises beyond its training data. Through rigorous experimentation, researchers can observe how the model behaves under different conditions and identify patterns of underperformance or overfitting. This analysis is crucial when deploying ML solutions in real-world environments, especially in ransomware detection, where both false positives and false negatives can have serious consequences. A carefully

designed experimental analysis ensures that the model meets the desired standards of performance, robustness, and scalability before it is used in operational settings. Additionally, the selected top 5 features are compared with the MITRE Adversarial Tactics, Techniques, and Common Knowledge (MITRE ATT&CK) framework. Table 8 highlights the relationship between key network traffic features and their behavioral interpretations, each tied to specific MITRE ATT&CK techniques. Features such as Fwd IAT Min and Flow Duration provide valuable temporal cues. For instance, a very low Fwd IAT Min suggests rapid packet bursts, often seen during automated ransomware activity, aligning with T1486 âĂŞ Data Encrypted for Impact. On the other hand, a prolonged Flow Duration can indicate ongoing data exfiltration over extended sessions, linked to T1041 âĂŞ Exfiltration Over C2 Channel. In a similar vein, Flow Packets/s sheds light on the intensity and consistency of network traffic. A high rate of small packets may reflect command-and-control (C2) activity, associated with T1071 âĂŞ Application Layer Protocol. The Packet Length StdDev helps detect anomalies in packet sizes. Higher variability often indicates obfuscation strategies malware employs to evade detection, aligning with T1027 âĂŞ Obfuscated Files or Information. Finally, a high ACK Flag Count may indicate repetitive beaconing, typical of C2 operations over web protocols, corresponding to T1071.001 âĂŞ Web Protocols (Beaconing). This analysis highlights how these features contribute not only to model performance but also to a deeper understanding of network threats in their context. By mapping features to known adversarial tactics, it enhances explainability, supports forensic investigations, and enables more informed, timely incident responses.

Table 8. Mapping of Key Features to MITRE ATT&CK Techniques

Feature Name	Interpretation	Related MITRE ATT&CK Technique
Fwd IAT Min	Sudden burst of data indicating fast encryption behavior	T1486 âĂŞ Data Encrypted for Impact
Flow Packets/s	High rate of small packets suggest command-and-control (C2) communication	T1071 âĂŞ Application Layer Protocol
Flow Duration	Prolonged flow duration may indicate potential data exfiltration activity	T1041 âĂŞ Exfiltration Over C2 Channel
Packet Length StdDev	Irregular packet sizes may signify stealthy or obfuscated traffic behavior	T1027 âĂŞ Obfuscated Files or Information
ACK Flag Count	Frequent ACK flags can reflect beaconing behavior to a C2 server	T1071.001 âĂŞ Web Protocols (Beaconing)

3.9 Hyperparameter Tuning Methods

Selecting the right hyperparameters is a critical step in optimising ML models for performance. Two commonly used techniques for this task are *Grid Search* and *Randomised Search*. Both aim to find the best combination of hyperparameters, but they differ in their strategies and computational demands as shown in table 9. Tables 10, 11 summarise a comparative evaluation

Table 9. Optimal hyperparameter values for each ML model used in this study

Model	Hyperparameters Tuned	Optimal Values
LightGBM	`num_leaves`, `max_depth`, `learning_rate`, `n_estimators`, `subsample`, `colsample_bytree`	`num_leaves`=31, `max_depth`=7, `learning_rate`=0.05, `n_estimators`=200, `subsample`=0.8, `colsample_bytree`=0.8
XGBoost	`learning_rate`, `max_depth`, `n_estimators`, `subsample`, `colsample_bytree`, `gamma`	`learning_rate`=0.1, `max_depth`=6, `n_estimators`=150, `subsample`=0.9, `colsample_bytree`=0.9, `gamma`=0.1
Random Forest	`n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`, `bootstrap`	`n_estimators`=100, `max_depth`=15, `min_samples_split`=2, `min_samples_leaf`=1, `bootstrap`=True
Decision Tree	`criterion`, `max_depth`, `min_samples_split`, `min_samples_leaf`	`criterion`='gini', `max_depth`=10, `min_samples_split`=4, `min_samples_leaf`=2
SVM	`kernel`, `C`, `gamma`	`kernel`='rbf', `C`=1.0, `gamma`='scale'

between these two methods. The results reveal that Grid Search consistently delivers higher performance across all evaluated metrics. Specifically, it achieves an impressive accuracy of 98.84% and a weighted F1-score of 98.67. This exhaustive search method systematically explores all possible parameter combinations, making it highly effective albeit computationally expensive. On the other hand, Randomised Search offers a more resource-efficient alternative by sampling a fixed number of parameter combinations from a specified distribution. While it is less computationally intensive, its performance is slightly lower, with an accuracy of 98.46% and a weighted F1-score of 98.09. These findings highlight a common trade-off in ML workflows: achieving the best possible performance versus managing computational resources. For applications where accuracy is paramount and computational resources are available, Grid Search remains the preferred method. However, in time-sensitive or resource-constrained scenarios, Randomised Search presents a practical alternative that still yields competitive results.

Table 10. Comparison of Grid Search and Randomised Search for hyperparameter tuning

Method	Accuracy (%)	Weighted F1-Score (%)
Grid Search	98.84	98.67
Randomized Search	98.46	98.09

Table 11. Performance Comparison of Hyperparameter Tuning Methods.

Metric	Grid Search	Randomized Search
Accuracy	0.9884	0.9846
Precision (Weighted)	0.9872	0.9823
Recall (Weighted)	0.9884	0.9846
F1 Score (Weighted)	0.9867	0.9809

3.10 Online Learning Evaluation under Concept Drift

In the context of ransomware detection, the importance of online learning lies in its ability to adapt to evolving threat patterns in real time. Ransomware variants frequently mutate their payload signatures, obfuscate command-and-control channels, and alter network behaviors to bypass static detection models. Traditional batch-trained models, although effective initially, often degrade in accuracy over time due to concept drift, where the underlying data distribution changes. This poses a critical risk in production environments, especially where detection systems rely solely on historical patterns. By implementing online learning strategies, such as incremental LightGBM, models can continuously update as new data streams arrive, thereby maintaining robustness against novel ransomware tactics. This capability is essential for sustaining low false-negative rates, ensuring timely mitigation, and supporting adaptive cybersecurity frameworks in dynamic Android network environments. To address concerns about model robustness against evolving Android ransomware variants, we conducted additional evaluations that simulated concept drift using chronologically partitioned traffic data.

To evaluate the behaviour of the model under temporal concept drift, the timestamped networkâĂŞtraffic dataset was chronologically sorted and partitioned into five sequential time blocks, denoted as T1âĂŞT5. Each block represents a continuous, non-overlapping slice of the dataset based on its temporal order.

- **T1**: Earliest 20% of the timestamp-ordered samples
- **T2**: Next 20%
- **T3**: Middle 20%
- **T4**: Next 20%
- **T5**: Most recent 20% of the dataset

This chronological segmentation ensures a realistic simulation of evolving ransomware behaviour. The incremental LightGBM model was first trained on T1 and then updated sequentially using T2 through T5 via the `update()` mechanism. At each step, performance was measured on the next time block to quantify the impact of temporal drift and the model's adaptability to newly emerging ransomware patterns. LightGBM does not provide a true textttpartial_fit() interface; however, it supports incremental batch learning through warm-start updates. In our concept-drift experiment, LightGBM was trained on T1

and then further trained on each subsequent block (T2âĂŞT5) using the `init_model` parameter. This approach appends new trees to the existing model rather than retraining from scratch, enabling LightGBM to adapt to evolving temporal data in a stepwise manner.

The online learning evaluation was conducted using the Android Ransomware Detection dataset. To simulate real-world concept drift, the dataset was partitioned sequentially into five temporal blocks (T1âĂŞT5) that capture the progressive evolution of ransomware traffic patterns over time. A static LightGBM model was initially trained on the full dataset and used as the baseline for performance comparison. For the online learning scenario, an incremental LightGBM model was employed using the update() method, allowing the model to adapt across successive data blocks without complete retraining. Evaluation metrics, accuracy, precision, recall, and F1-score, were tracked at each time step to assess the impact of temporal drift and measure model robustness under streaming conditions.

The performance of both the static LightGBM and the incremental LightGBM models was evaluated across five temporally ordered data blocks (T1âĂŞT5) to assess the impact of concept drift, as shown in Table 12. The static LightGBM model was trained once on the full dataset, whereas the incremental LightGBM model was updated sequentially without full retraining. The following table summarizes the accuracy, F1-score, precision, and recall of the incremental model at each time step:

Table 12. Performance of Incremental LightGBM under Concept Drift (T1âĂŞT5)

Time Block	Accuracy (%)	F1-Score	Precision	Recall	Observation
T1	96.12	0.9620	0.9641	0.9599	Initial performance on fresh model trained on full dataset
T2	94.80	0.9487	0.9510	0.9464	Gradual degradation begins due to concept drift
T3	93.05	0.9303	0.9315	0.9291	Accumulated drift becomes evident (3% decay)
T4	91.20	0.9125	0.9142	0.9108	Incremental updates slow degradation, but performance declines
T5	90.78	0.9091	0.9110	0.9072	Performance plateaued; adaptation saturates without full retraining

For comparison, the static LightGBM model achieved an average accuracy of 95.76%, precision of 0.9571, recall of 0.9575, and F1-score of 0.9573; however, it failed to sustain these metrics under temporal drift. In contrast, the incremental LightGBM exhibited robustness through partial adaptation, although its accuracy degraded by approximately 5.34% from T1 to T5. This underscores the need for online adaptation mechanisms in ransomware detection systems deployed in dynamic environments.

We incorporated an additional validation step using the recent PermGuard Android malware dataset[37] to assess the generalisation of our model to emerging ransomware families. Table 13 summarises the comparative performance between datasets. We also included the Matthews Correlation Coefficient (MCC), which provides a more balanced view in the presence of class imbalance.

Table 13. Performance Comparison Between Datasets

Metric	Android Ransomware Detection Dataset[38]	PermGuard[37]
Accuracy	0.961	0.998
Precision	0.952	0.973
Recall	0.951	0.989
F1-Score	0.957	0.975
MCC	0.985	0.989
Classifier	Random Forest	Random Forest

3.11 Comparison with state-of-the-art approaches

In terms of precision, Table 14 compares various state-of-the-art Android ransomware detection approaches. The findings indicate that DL frameworks and ensemble-based models consistently outperform traditional classifiers. These findings suggest that employing a combination of multiple weak learners or deep contextual embeddings has the potential to enhance detection capabilities. That models such as Naive Bayes and Linear SVM performed with lower accuracies indicates that conventional statistical techniques are not very effective when handling intricate ransomware traffic patterns. The ability of models such as XGBoost and CatBoost to perform well demonstrates the effectiveness of advanced boosting techniques in detecting subtle

changes in bad behavior. In total, the ensemble-based approach proposed in this work performs at least as well as, and in some cases better than, alternative approaches reported in the literature. This indicates that it can be applied for real-time, scalable ransomware detection on Android systems.

Table 14. Accuracy Comparison Across Different Studies

Study	Model Used	Accuracy (%)
Hossain et al.[7]	Ensemble	99.81
Ahmed et al.[33]	DT	97.24
Amenova et al.[34]	CNN-LSTM	94.39
Islam et al.[35]	Ensemble	95.00
Adeniyi et al.[36]	RF	99.98
Amer et al.[17]	LSTM	97.50
Our Work	LightGBM + Grid Search	**98.84**
	LightGBM + Randomized Search	**98.46**

3.12 Discussion

The results demonstrate the effectiveness of ensemble learning algorithms, particularly LightGBM, in anomaly-based detection of Android ransomware using network traffic information. The high classification accuracy and minimal variance across stratified folds reveal strong generalization to unseen traffic patterns. This is particularly crucial for detecting ransomware, as new variants and methods for concealing them tend to reduce the effectiveness of static signature-based solutions. One of the major aspects of this research is its focus on model explainability. DL-based models are excellent at detecting malware but often act like black boxes. However, by applying SHAP and LIME together within our framework, it becomes apparent why the predictions were made, instilling confidence and facilitating their use in mobile security environments. This is particularly relevant for forensic investigations, legal compliance, and security systems that involve humans. From a deployment perspective, the model is lean and compatible with edge or mobile security. The feature selection method ensures that the dimensionality is reduced without compromising on accuracy. This enables fast inference with minimal additional processing capacity. RobustScaler also mitigates the non-Gaussianity of ransomware traffic, making the model more resilient to outliers and traffic anomalies. Despite the robust results, certain issues remain. The dataset is complete, but it is based on controlled environments. Traffic in the real world could include noise, overlapping traffic flows, and encrypted dialogue. Additionally, the models do not account for the evolution of ransomware behavior over time. Future studies should aim to overcome these limitations by using federated learning techniques that enable adaptive model updates without requiring central data sharing.

4 Conclusion and Future Work

This study presents a robust and interpretable ensemble-based ML framework for proactive detection of Android ransomware using network traffic metadata. A comprehensive evaluation was conducted on a balanced dataset, demonstrating that ensemble models such as LightGBM, XGBoost, and Random Forest, along with classical classifiers such as Decision Tree and SVM, achieve strong predictive performance in detecting ransomware activity. Among these, LightGBM emerged as the top performer with the highest PR-AUC, offering a reliable trade-off between accuracy and computational efficiency. To address class imbalance, SMOTE was applied during cross-validation, improving recall for minority ransomware variants without overfitting. In addition to standard accuracy, precision, recall, and F1-score, these metrics were also reported to better reflect real-world detection efficacy under imbalanced class distributions.

To assess temporal robustness, an additional experiment was conducted using an incremental LightGBM model under concept drift conditions. The dataset was partitioned chronologically into five temporal blocks (T1âĂŞT5) to simulate the real-world evolution of ransomware traffic. Results indicated that while the static LightGBM model initially performed well, performance degraded by 5.34% over time due to unseen behavioral drift. However, the incremental update mechanism, applied without full retraining, partially mitigated this decay, highlighting its potential for real-time, drift-aware deployment in dynamic Android environments.

To enhance model transparency, SHAP and LIME explanations were applied to interpret the predictions at both global and local levels. These insights support cybersecurity analysts in understanding and validating automated threat decisions, thereby aligning the framework with XAI principles. While the current study evaluates the classification performance of ensemble and classical ML models, future work will focus on optimizing these models for deployment on resource-constrained mobile

and edge platforms. This will include benchmarking inference latency and memory footprint on devices such as Raspberry Pi 4, using techniques like quantization and ONNX conversion to ensure real-time responsiveness in mobile environments. These efforts aim to bridge the final step between high detection accuracy and practical edge deployment. In conclusion, this work presents a scalable, interpretable, and concept-drift-resilient framework for Android ransomware detection, laying the groundwork for robust, real-time cybersecurity solutions in mobile network environments.

Conflict of Interest

The authors declare no competing interests that could influence the content or findings of this manuscript. The research was conducted independently and is free from any financial, professional, or personal conflicts.

Data Availability

The datasets analyzed during the current study are available in the IEEE DataPort and Kaggle repository at https://dx.doi.org/10.21227/d744-tb96 and https://doi.org/10.34740/KAGGLE/DSV/4987535

Author Contributions

Kirubavathi G.: Conceptualization, Data curation, Methodology, Supervision, Writing–original draft, Writing–review & editing. Padma Mayuri B.: Methodology, Data curation, Feature engineering, Writing–original draft. Pranathasree S.: Methodology, Software, Model development. Rajeswari Alagappan: Methodology,Software, implementation. Amal Ajayan: Software, Writing–original draft, Writing–review & editing. Waleed M. Ismael: Investigation, Experimental design, Writing–review & editing. Ateeq Ur Rehman: Supervision, Project administration, Result interpretation, Writing–review & editing. All authors have read and approved the final manuscript.

References

1. Qureshi, S. U. *et al.* Systematic review of deep learning solutions for malware detection and forensic analysis in iot. *J. King Saud Univ. - Comput. Inf. Sci.* **36**, 102164, DOI: https://doi.org/10.1016/j.jksuci.2024.102164 (2024).
2. Aslan, Ã., Aktug, S. S., Ozkan-Okay, M., Yilmaz, A. A. & Akin, E. A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions. *Electronics* **12**, DOI: https://doi.org/10.3390/electronics12061333 (2023).
3. Begovic, K., Al-Ali, A. & Malluhi, Q. Cryptographic ransomware encryption detection: Survey. *Comput. & Secur.* **132**, 103349, DOI: https://doi.org/10.1016/j.cose.2023.103349 (2023).
4. Muhammad, Z. *et al.* Smartphone security and privacy: A survey on apts, sensor-based attacks, side-channel attacks, google play attacks, and defenses. *Technologies* **11**, DOI: https://doi.org/10.3390/technologies11030076 (2023).
5. Ferdous, J., Islam, R., Mahboubi, A. & Islam, M. Z. A survey on ml techniques for multi-platform malware detection: Securing pc, mobile devices, iot, and cloud environments. *Sensors* **25**, DOI: https://doi.org/10.3390/s25041153 (2025).
6. Razgallah, A., Khoury, R., HallÃl', S. & Khanmohammadi, K. A survey of malware detection in android apps: Recommendations and perspectives for future research. *Comput. Sci. Rev.* **39**, 100358, DOI: https://doi.org/10.1016/j.cosrev.2020.100358 (2021).
7. Hossain, M. A. *et al.* Towards superior android ransomware detection: An ensemble machine learning perspective. *Cyber Secur. Appl.* **3**, 100076, DOI: https://doi.org/10.1016/j.csa.2024.100076 (2025).
8. Yan, P. & Talaei Khoei, T. Securing the internet of things: A comprehensive review of ransomware attacks, detection, countermeasures, and future prospects. *Frankl. Open* **11**, 100256, DOI: https://doi.org/10.1016/j.fraope.2025.100256 (2025).
9. Albshaier, L., Almarri, S. & Rahman, M. M. H. Earlier decision on detection of ransomware identification: A comprehensive systematic literature review. *Information* **15**, DOI: https://doi.org/10.3390/info15080484 (2024).
10. Al-Kadhimi, A. A., Singh, M. M. & Khalid, M. N. A. A systematic literature review and a conceptual framework proposition for advanced persistent threats (apt) detection for mobile devices using artificial intelligence techniques. *Appl. Sci.* **13**, DOI: https://doi.org/10.3390/app13148056 (2023).
11. Alraizza, A. & Algarni, A. Ransomware detection using machine learning: A survey. *Big Data Cogn. Comput.* **7**, DOI: https://doi.org/10.3390/bdcc7030143 (2023).
12. Almotiri, S. H. Ai driven iomt security framework for advanced malware and ransomware detection in sdn. *J. Cloud Comput.* **14**, 19, DOI: https://doi.org/10.1186/s13677-025-00745-w (2025).

13. Joshi, Y. S., Mahajan, H., Joshi, S. N., Gupta, K. P. & Agarkar, A. A. Signature-less ransomware detection and mitigation. *J. Comput. Virol. Hacking Tech.* **17**, 299–306, DOI: https://doi.org/10.1007/s11416-021-00384-0 (2021).

14. Mohamed, N. Artificial intelligence and machine learning in cybersecurity: a deep dive into state-of-the-art techniques and future paradigms. *Knowl. Inf. Syst.* DOI: https://doi.org/10.1007/s10115-025-02429-y (2025).

15. Lu, H. *et al.* Autod: Intelligent blockchain application unpacking based on jni layer deception call. *IEEE Netw.* **35**, 215–221, DOI: https://doi.org/10.1109/MNET.011.2000467 (2021).

16. Lu, H. *et al.* Deepautod: Research on distributed machine learning oriented scalable mobile communication security unpacking system. *IEEE Transactions on Netw. Sci. Eng.* **9**, 2052–2065, DOI: https://doi.org/10.1109/TNSE.2021.3100750 (2022).

17. Amer, E. & El-Sappagh, S. Robust deep learning early alarm prediction model based on the behavioural smell for android malware. *Comput. & Secur.* **116**, 102670, DOI: https://doi.org/10.1016/j.cose.2022.102670 (2022).

18. Hull, G., John, H. & Arief, B. Ransomware deployment methods and analysis: views from a predictive model and human responses. *Crime Sci.* **8**, 2, DOI: https://doi.org/10.1186/s40163-019-0097-9 (2019).

19. Guerra-Manzanares, A., Luckner, M. & Bahsi, H. Concept drift and cross-device behavior: Challenges and implications for effective android malware detection. *Comput. & Secur.* **120**, 102757, DOI: https://doi.org/10.1016/j.cose.2022.102757 (2022).

20. Almohaini, R., Almomani, I. & AlKhayer, A. Hybrid-based analysis impact on ransomware detection for android systems. *Appl. Sci.* **11**, DOI: https://doi.org/10.3390/app112210976 (2021).

21. Gu, J., Zhu, H., Han, Z., Li, X. & Zhao, J. Gsedroid: Gnn-based android malware detection framework using lightweight semantic embedding. *Comput. & Secur.* **140**, 103807, DOI: https://doi.org/10.1016/j.cose.2024.103807 (2024).

22. Hsu, R.-H. *et al.* A privacy-preserving federated learning system for android malware detection based on edge computing. In *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*, 128–136, DOI: https://doi.org/10.1109/AsiaJCIS50894.2020.00031 (2020).

23. Karat, G. *et al.* Cnn-lstm hybrid model for enhanced malware analysis and detection. *Procedia Comput. Sci.* **233**, 492–503, DOI: https://doi.org/10.1016/j.procs.2024.03.239 (2024). 5th International Conference on Innovative Data Communication Technologies and Application (ICIDCA 2024).

24. Muzaffar, A., Ragab Hassen, H., Lones, M. A. & Zantout, H. An in-depth review of machine learning based android malware detection. *Comput. & Secur.* **121**, 102833, DOI: https://doi.org/10.1016/j.cose.2022.102833 (2022).

25. Meijin, L. *et al.* A systematic overview of android malware detection. *Appl. Artif. Intell.* **36**, 2007327, DOI: https://doi.org/10.1080/08839514.2021.2007327 (2022).

26. Mawoh, R. Y., Wacka, J. B. A., Tchakounte, F., Fachkha, C. & Kolyang. An accurate approach to discriminate android colluded malware from single app malware using permissions intelligence. *Sci. Reports* **15**, 10680, DOI: https://doi.org/10.1038/s41598-025-86568-w (2025).

27. Lu, T., Du, Y., Ouyang, L., Chen, Q. & Wang, X. Android malware detection based on a hybrid deep learning model. *Secur. Commun. Networks* **2020**, 8863617, DOI: https://doi.org/10.1155/2020/8863617 (2020). https://onlinelibrary.wiley.com/doi/pdf/10.1155/2020/8863617.

28. Ajayan, A., Kirubavathi, G. & Sarker, I. H. Distilxids: Efficient, lightweight and explainable transformer-based language model for real-time network intrusion detection. *Neurocomputing* **668**, 132398, DOI: https://doi.org/10.1016/j.neucom.2025.132398 (2026).

29. Sarker, I. H., Janicke, H., Mohsin, A., Gill, A. & Maglaras, L. Explainable ai for cybersecurity automation, intelligence and trustworthiness in digital twin: Methods, taxonomy, challenges and prospects. *ICT Express* **10**, 935–958, DOI: https://doi.org/10.1016/j.icte.2024.05.007 (2024).

30. Qi, P., Chiaro, D. & Piccialli, F. Small models, big impact: A review on the power of lightweight federated learning. *Futur. Gener. Comput. Syst.* **162**, 107484, DOI: https://doi.org/10.1016/j.future.2024.107484 (2025).

31. Ispahany, J., Islam, M. R., Islam, M. Z. & Khan, M. A. Ransomware detection using machine learning: A review, research limitations and future directions. *IEEE Access* **12**, 68785–68813, DOI: https://doi.org/10.1109/ACCESS.2024.3397921 (2024).

32. Hasan, R. *et al.* Enhancing malware detection with feature selection and scaling techniques using machine learning models. *Sci. Reports* **15**, 9122, DOI: https://doi.org/10.1038/s41598-025-93447-x (2025).

33. Albin Ahmed, A. *et al.* Android ransomware detection using supervised machine learning techniques based on traffic analysis. *Sensors* **24**, DOI: https://doi.org/10.3390/s24010189 (2024).

34. Amenova, S., Turan, C. & Zharkynbek, D. Android malware classification by cnn-lstm. In *2022 International Conference on Smart Information Systems and Technologies (SIST)*, 1–4, DOI: https://doi.org/10.1109/SIST54437.2022.9945816 (IEEE, 2022).

35. Islam, R., Sayed, M. I., Saha, S., Hossain, M. J. & Masud, M. A. Android malware classification using optimum feature selection and ensemble machine learning. *Internet Things Cyber-Physical Syst.* **3**, 100–111, DOI: https://doi.org/10.1016/j.iotcps.2023.03.001 (2023).

36. Adeniyi, A. O., Olabiyisi, S. O., Adepoju, T. M. & Sanusi, B. A. Comparative analysis of some machine learning algorithms for the classification of ransomware. *Int. J. Res. Sci. Innov.* **12**, DOI: https://doi.org/10.51244/IJRSI.2025.120800045 (2025).

37. Prasad, A. Permguard android malware dataset, DOI: https://doi.org/10.1109/ACCESS.2024.3523629 (2024).

38. Chakraborty, S. Android ransomware detection, DOI: https://doi.org/10.34740/KAGGLE/DSV/4987535 (2023).