Federated Learning Approach for Vision-based Android Malware Family Classification using CNN

Thi Nguyen Chi
University of Information Technology - HCMVNU
Ho Chi Minh city, Vietnam
21522614@gm.uit.edu.vn

Duc Bui Le Trong
University of Information Technology - HCMVNU
Ho Chi Minh city, Vietnam
21520725@gm.uit.edu.vn

PhD. Cam Nguyen Tan
University of Information Technology - HCMVNU
Ho Chi Minh city, Vietnam
camnt@uit.edu.vn

ABSTRACT

In light of the vast number of Android devices and users, ensuring information security on the Android platform has become a critical concern. Leveraging the increasing volume of data and advancements in technology, Machine Learning (ML) is widely employed for the detection and classification of Android malware. However, conventional ML models necessitate large datasets for effective training, posing privacy concerns for users. To address this, our team implements Federated Learning, a decentralized learning method that enables diverse devices to collaborate on a machine learning model without compromising sensitive data on a central server. We apply this approach to the classification of Android malware, addressing information security concerns associated with data storage and training. Our system converts APK files into RGB images, normalizes them, and employs them as inputs for a Convolutional Neural Network (CNN) classification model. OpenFL, an open-source Federated Learning framework, is utilized for implementation. Research findings indicate that Federated Learning is an effective approach for Android malware classification in a distributed environment, ensuring privacy and security for user data.

1 INTRODUCTION

In recent years, Android-based platforms and the number of Android users have been increasing significantly. However, the number of users who is aware of information security is still inconsiderable. It attracts a numerous amount of cyber assaults to take advantage of this opportunity to carry out evasion on users' system since any of Android's architecture layers are vulnerable and have open-source. It is feasible for hackers and attackers to modify the original code to exploit or take control the whole system depending on their malicious intentions [11]. These dynamic and evolving threats necessitates the development of advanced and adaptive techniques for timely detection and classification. Because there were many proposed Android Malware detection systems that can accurately identify the presence of malware [19] [2], we believe that it is more necessary to categorize their variants instead of continuing solving that problem. We can determine exactly the vulnerability of the system and prioritize mitigation techniques based on the additional information about the threat.

The majority of studies in this domain mainly propose static, dynamic and hybrid analysis. The static-based technique tries to extract specific features such as API class, opcodes, permissions from the APK file without executing it. On the other hand, dynamic-based approach captures all the events or actions through logs after executing the APK in actual devices or particular environments. Then, features would be extracted from these logs providing inputs for further detection. A more efficient way is the combination of the two above approaches for this detection called hybrid [11]. Following the static-based method, we try to remake a proposed lightweight system serving the classification of Android Malware variants. APK files would be transformed into RGB images. Results of the transformation will then be normalized using nearest-neighbor interpolation as the inputs of the Convolutional Neural Network (CNN) classifier [5].

Traditional Machine Learning (ML) approaches require a centralized data storage and training process. In cybersecurity, this prototype would be a big challenge to be overcome since having only one singular data point raises the risk of losing that point to attackers' control. Therefore, we prefer to implement a decentralized concept for this study called Federated Learning (FL). This technique enables many different devices to work on a collaborative ML model without uploading their sensitive data related to the training process to the central server. A global model is located on an aggregator node. This node would transfer necessary data to other authorized collaborators and these nodes start to train that global model locally with their local data. There are several motivations of FL for cybersecurity. This consolidating way of collecting and sharing data makes sure to guarantee data confidentiality, availability, and integrity. Each node collects data locally and transfers only achieved local model's parameters rather than sends the raw data which is always targeted by cyber invaders. Besides, FL models is designed for the privacy preservation, so crucial information would not be easily transferred out of the local environment on every node while FL is utilized [3].

This study delves into the utilization of Open Federated Learning (OpenFL) [15], an open-source FL framework known for its versatility and efficiency. This framework was initially developed for healthcare purposes by Intel Labs and the University of Pennsylvania. Nevertheless, as the noticeable development of cybersecurity and the exigency of federated learning method for decentralized data, the OpenFL project is continuously updated to become more diversely applied in many other purposes [15].Specifically, our major works in this study can be summarized as follows:

- **Rebuild a lightweight classifier for RGB images transformed from APK files [5]:** We reuse the general concept of turning APK files into RGB images by mapping permissions, intents, activities, services from the manifest.xml files on the green channel; API calls and opcodes from the classes.dex files on the red channel; and all the suspected behaviour consisting of suspicious API calls, protected strings, permissions on the blue channel. This mapping process relies on the defined dictionary comprised of APK's components and their corresponding value in range (0;255). The resultant RGB images of this modification are resized using nearest-neighbour interpolation to remain their essence as much as possible.

- **Making the training process into an FL workflow:** We loaded the whole source codes of RGB images mapping process and model into the FL workspace located at the aggregator node. The packed workspace is then moved to others authorized collaborators via a mutually-authenticated Transport Layer Security (TLS) network connection which helps to encrypt and decrypt those sensitive data. Collaborators utilize received information for their local training process. After they have finished their training, they starts to send back optimized weights and computed metrics. The aggregator node synthesizes those data and apply them to the global model by utilizing an aggregation algorithm which is ..... in this circumstance. This mechanism repeats for a particular number of iterations defined in the FL plan comprised along side with the model within the FL workspace. For these purposes, we utilize OpenFL [15], which is an efficient framework for FL to train ML model in the decentralized-data concept.

The source code of this study is posted on Github[1]

2 RELATED WORKS

2.1 Android Malware Family Categorization

Traditional machine learning methods for Android Malware analysis include static, dynamic and hybrid techniques [14]. Static approach involve detection and analysis based on features of the application without executing it. However, this method does not have a good performance whenever indentifying certain source code modification behaviours. [12] evaluated their methods on both CICMaldroid2020, Drebin Dataset and achieved a very high F1-Score of 97.8%, 99.6% accuracy respectively on those two datatsets using 1D-CNN[2]. Dynamic technique detect the execution behaviour of an application such as system calls, network connections, etc [16]. Obtaining a notable 97.84% F1-Score and a 2.76% false positive rate, [7] uses dynamic analysis with neural networks and effectively examines malware behaviors. Their semi-supervised deep learning model outperforms LP, Random Forest, Decision Tree, SVM, and K-NN, even with minimal (1% - 10%) labeled samples. In addition, hybrid detection technique is the combination of static and dynamic analysis in a two-step process for improved accuracy.

Besides these three common techniques, vision-based analysis approach offered a new direction to deploy CNN algorithms to effectively detect malicious applications by classify them with their features converted to pictures. [6] proposed a system which is used to transform the classes.dex file of the APK into RGB image and subsequently fed it to the CNN model for experimental purposes. [10] created a detection system for malicious software in the Industrial Internet of Things (IIoT) using a CNN algorithm focused on visual elements. Their findings demonstrated that using colored visualizations for malware detection surpasses the efficiency of utilizing gray-scale images. [18] designed a sophisticated mapping algorithm for APK files into RGB images. In their innovative system, the Manifest file was assigned to the green channel, while API calls and opcodes were allocated to the red channel. Additionally, the blue channel was dedicated to representing malicious behaviors. [5] conducted an additional inquiry using the Drebin database. In their system, malware detection involves identifying locations with malicious opcode sequences within Android apps. Notably, the detection performance can be improved by integrating machine learning algorithms with the CNN model.

2.2 Federated Learning concept

Machine learning has recently shifted from centralized models to decentralized learning, distributing the learning process across local nodes to address privacy concerns and enable collaborative learning without centralized data storage. This transition unlocks the potential for privacy-preserving machine learning in various applications.

In traditional centralized method, participants upload data to a central server for machine learning model training, eliminating the need for their resources. However, this method poses security risks to client data and increases the likelihood of data transmission failures due to the substantial volume involved.

Distributed learning offers an efficient and scalable solution for complex problems with large datasets, outperforming centralized learning. With the rising popularity of edge computing, it processes applications, including basic machine-learning algorithms, facilitating the creation of public cloud-computing services for big data applications [17]. Unlike centralized learning, distributed machine learning tasks are performed independently on each participant's machine. Initially, pre-trained models are provided by the server to clients. After using their datasets for training, participating devices send model parameters w^i to the server. Following a set number of exchanges, the server calculates global model parameters, undergoes testing, and updates the global model. Crucially, clients can't access results from others until receiving the updated global model [1].

FL shares similarities with distributed learning, often regarded as a subset of distributed machine learning. However, in federated learning, each client independently conducts training without direct collaboration with other clients in the network. Clients declare local epochs during model training, and once completed, the local model sends its parameters w^i. After collecting parameters from all clients in a round, the server computes and updates global parameters, sending them back to all clients for the next round. This iterative process in federated learning continues until the global model achieves the desired accuracy or a specified number of rounds. The key distinction lies in the decentralized nature of federated learning, where clients perform independent training, contributing local updates to a global model, fostering privacy and reducing communication requirements compared to traditional distributed learning [4].

[1]https://github.com/builetrongduc/Federated-Learning-for-Android-Malware-Classification.git
[2]https://github.com/baosenguo/Kaggle-MoA-2nd-Place-Solution

3 PROPOSED APPROACH

3.1 *Dataset*

We use the CICMalDroid2020 dataset [8] which comprises 17,341 samples of Android apps. There are five different categories of these samples. Each malware category can be briefly described below:

- **Adware:** Injects ads into legitimate apps, persistently displaying them even if the user tries to close the app. Can root-infect devices and steal personal information.
- **Banking Malware:** Specialized in mimicking banking apps to steal login credentials and transmit the data to a command and control server.
- **SMS Malware:** Operates via SMS, intercepting messages, sending malicious SMS, and stealing data.
- **Mobile Riskware:** Legitimate programs that can be exploited to cause damage, potentially transforming into various malware types.
- **Benign:** All other applications not inherently malicious. To verify their benign nature, all samples in this category are scanned with VirusTotal.

However, there are just 11,598 samples are successfully compiled while the remaining ones failed due to time-out, invalid APK files, memory allocation failures, and "unterminated string". This property is shown in Table 1.

Table 1: A summerization of CICMalDroid2020

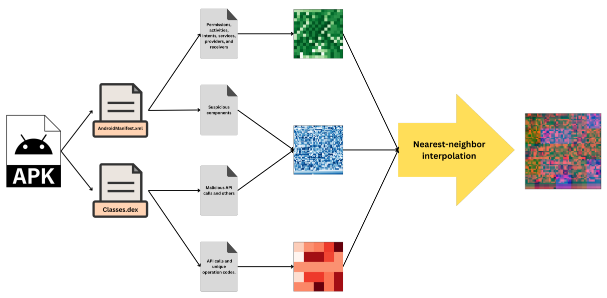
Category	Number of samples
Adware	1253
Banking Malware	2100
SMS Malware	3904
Mobile Riskware	2546
Benign	1795
Total	11698

3.2 *RGB-based Android Malware's family classifier*

Comprising a compressed archive, an APK encapsulates all components necessary for the installation and execution of an Android app on devices. The structure delineates a systematic organization of key elements, each contributing to the overall functionality and behavior of the application.

- *META-INF/:* This directory contains metadata files, including MANIFEST.MF, crucial for verifying the APK's integrity.
- *lib/:* accommodates native libraries tailored for diverse CPU architectures, ensuring optimal performance across different devices.
- *AndroidManifest.xml:* a pivotal XML file that orchestrates crucial details such as permissions, activities, services, and receivers, defining the app's configuration.
- *classes.dex:* contains compiled bytecode of the Java source code, executed by the Dalvik or ART virtual machine.

Figure 1: A visualization of the APK to RGB image's conversion


APK
AndroidManifest.xml
Classes.dex
Permissions, activities, intents, services, providers, and receivers
Suspicious components
Malicious API calls and others
API calls and unique operation codes.
Nearest-neighbor interpolation

- *resources.arsc:* streamlines resource access, contributing to an efficient runtime experience. Defines access permissions and necessary hardware requirements.

At first, we would like to analyze and extract permission and app components from the manifest file, including activities, intents, services, providers, and receivers using *Androguard*[3]. All the extracted components are converted into pixel values by using the predefined dictionary [4]. The final value used to map to the dictionary is calculated by summing every character's ASCII codes (P), then taking the mod by 256. So that the component would be normalized in range of [0;255]. The general equation (1) is given below.

$$Pixel\ value = \sum_{i=0}^{n} P_i \pmod{256} \tag{1}$$

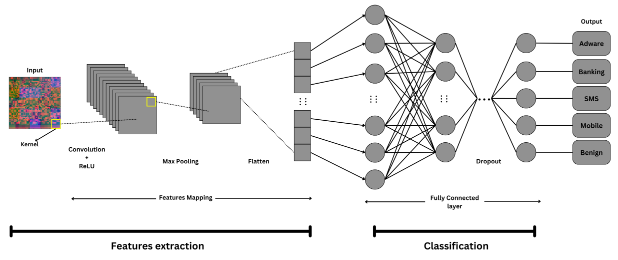
We use that calculation to map extracted benign permissions and others components from the analyzed manifest file on the green channel in a particular order. Similar to the above conversion, extracted API calls and unique operation codes from the .dex file would be mapped on the red channel; suspicious components from both manifest and .dex file are mapped on the blue channel. Whether there is any undefined suspicious activity, it is still mapped on the red and green channel. These above conversions can be visualized in Figure 1.

The size of each resultant channel depends on the properties of .dex and manifest files. Therefore, there would be a variety of sizes of those transformed ones. We decide to reshape them to a consistent size then merge them to a single RGB image to feed our CNN classifier by utilizing the Nearest Neighbour interpolation (NNI)[13]. NNI's concept is newly created value, in this case is Pixel values, based on the original image's existing values, so that the pattern of the old one would be preserved.

Secondly, we provided Convolutional Neural Network (CNN) classifier(Figure 2) designed for this image classification tasks. The model consists of an initial convolutional layer with 64 filters and a Rectified Linear Unit (ReLU) activation function, followed by max-pooling to reduce spatial dimensions. A flattening layer prepares the data for two fully connected layers, the first with 500 neurons

[3]https://github.com/androguard/androguard.git
[4]https://github.com/asimswati553/RGB-based-Andorid-Malware-detection.git

Figure 2: The utilized Convolutional Neural Network (CNN)



and a ReLU activation function, along with dropout rate of 0.5 for regularization. The second fully connected layer produces logits for the final classification into five output classes corresponding to five categories of Android Malware. The architecture incorporates convolutional and fully connected layers, utilizing ReLU activation and dropout to enhance non-linearity and prevent overfitting. We also use Cross Entropy loss function(2) which is considered to be the suitable one for this multi-class classification problem.

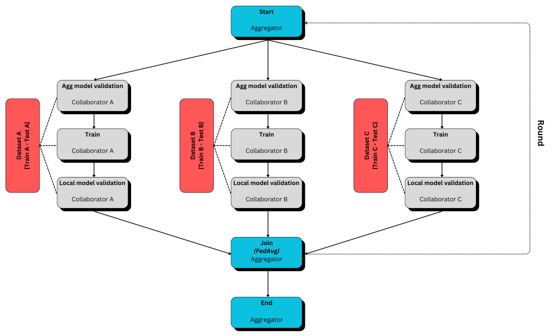
$$L(y, \hat{y}) = -\sum_{c=1}^{M} y \log(\hat{y}) \tag{2}$$

3.3 *The Federated Learning workflow*

Within this workflow, the utilization of OpenFL targets to make the communication between Aggregator and Collaborator, as shown in ... . Each collaborator trains its local model using the dataset and the hardware that are owned by that participant. Moreover, during the entire time, the dataset always stays inside the COL nodes. Aggregator's concerns include selecting clients, configuring the global model, and also aggregating received parameters from chosen participants in the FL network. In particular, we would use *FedAVG*[9] for the aggregation of received weights. aggregator also have the responsibility of transferring the collective settings to each node, such as IP address, rounds to train, or procedures for given federation tasks, which are all comprised in the FL plan, to the selected collaborators. A FL rounds is the process of aggregator sharing the configurations, collecting weights from collaborators through their state and aggregating all parameters to the global model.

The general workflow of the FL system is shown in Figure 3 . Firstly, the aggregator starts by initializing random weights w_0 for the global model and moves them to the stochastically m selected ones out of P participants in that round. Within this step, a number of rounds R, local epochs E_p, batch sizes β_p, learning rates η_p, and especially the aggregation algorithm *FedAvg* are also defined by the aggregator. After that, each p chosen participant begins its local training progress using the local D_p dataset and others previously defined training hyper-parameters. In this work, we utilize Adam optimizer as it has a positive efficiency in distributed training data problems. Subsequently, aggregator collects w_r^p weights corresponding to each p client taking part in the process within that r

Figure 3: The Federated Learning workflow



round and assembles them utilizing *FedAvg* algorithm. However, we realize that there is a small drawback with the traditional *FedAvg*. In particularly, incorporating a poorly performing local model in the construction of the global model during each round could lead to the overall global model performing inadequately as *FedAvg* has the same priority on every local models while averaging. Therefore, we decide to duplicate the weights $w_r^{p^*}$ from the local models having the highest accuracy acc^{p_r} and join it to the aggregation task at every round. Afterward, the aggregator node continues the training process by transferring acquired information to each participant that are randomly selected again in the following round. This loop is terminated when it finishes repeating in a specified number of rounds R. The entire algorithm is written in 1

Algorithm 1 Federated Learning workflow with modified *FedAvg*.

Input: Set of P participants indexed by p, R rounds of the workflow. E_p, D_p, η_p are respectively the number of local epochs, local dataset and learning rate.
Output: Final aggregated global model's parameters.
Server executes:

```
1:  initialize $w_0$
2:  for each round $r = 1, 2, \ldots$ do
3:      $m \leftarrow \text{random}(P; 1)$
4:      $P_r \leftarrow$ (random set of $m$ participants)
5:      for each $p \in P_r$ do
6:          $w_r^p \leftarrow \text{ParticipantUpdate}(p, w_r^p)$
7:      end for
8:      $p^* \leftarrow$ (the participants having $\max(acc^{p_r})$)
9:      $w_{r+1} \leftarrow \frac{1}{m+1}[(\sum_p^{P_r} w_r^p) + w_r^{p^*}]$
10: end for
```

ParticipantUpdate(p, w)**:**

```
1:  $\beta_p \leftarrow$ (split $D_p$ into batches of size $\beta_p$)
2:  for each $e \in E_p$ do
3:      for each $b \in \beta_p$ do
4:          $w \leftarrow \text{Adam}(w, \eta_p)$
5:      end for
6:  end for
7:  Return $w, acc$
```

4 EXPERIMENT

4.1 Experimental setup

In this study, we harnessed the potential of the CICMalDroid2020 dataset, a comprehensive and widely acknowledged repository in the field of Android malware detection. Given the scale and complexity of the dataset, we faced resource constraints that necessitated a judicious approach to data utilization. Consequently, we opted to strategically subsample 50% of the CICMalDroid2020 dataset, allowing us to maintain a balance between computational efficiency and the robustness of our analyses. Within this subsampled set, 30% of the chosen instances were earmarked for the critical evaluation phase of our experiments. The original label set of dataset is flipped as: Adware $\Rightarrow$ 0; Banking Malware $\Rightarrow$ 1; SMS Malware $\Rightarrow$ 2; Mobile Malware $\Rightarrow$ 3; Benign $\Rightarrow$ 4. The specific quantity of each label is mentioned in Table 2. This approach enables us to draw meaningful conclusions while addressing the limitations imposed by resource constraints.

Table 2: Quantity of each label

Label	Number of samples
0	764
1	1208
2	1942
3	2436
4	1969

Approaching the experimental setup, we executed the entire study on the Google Colab platform, leveraging its cloud-based infrastructure for efficient and collaborative research. To streamline the federation, three dedicated collaborators, denoted as A, B, and C, actively participated in the experimental process, each contributing to the global model located within the aggregator.

The primary dataset was methodically divided into three distinct subsets, as illustrated in Table 3, so that we could indicate the generality of FL. Each subset was assigned to a specific collaborator, fostering a distributed approach to dataset management and analysis. Within each subset, a standard practice was adopted for dataset partitioning, with a 7:3 ratio for training and testing sets, respectively. This partitioning scheme ensured a robust evaluation of the developed models while maintaining an adequate training sample to foster model generalization.

Table 3: Quantity of each label

Name of subsets	0	1	2	3	4
Dataset_A	80%	20%	10%	0	20%
Dataset_B	0	80%	20%	10%	50%
Dataset_C	20%	0	70%	90%	30%

The core of our classification framework was built upon Convolutional Neural Networks (CNNs). The architecture comprised one convolutional layer, one max-pooling layer, one flattening layer, and fully connected neural network layers, enabling the model to classify instances into five distinct categories. This standardized model architecture ensured consistency across all collaborators' experiments, providing a fair basis for comparative analysis and evaluation.

4.2 Performance evaluation

To assess and make a comparative analysis between centralized and federated machine learning, the evaluation process employed the following metrics:

- **Accuracy**(3)**:** quantifies the overall correctness of a model by determining the ratio of accurately predicted instances to the total number of instances within the dataset. Formally expressed as the division of true positives and true negatives by the total predictions, accuracy provides a comprehensive measure of a model's proficiency across all classes.

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{Total Predictions}} \tag{3}$$

- **Precision**(4)**:** is computed as the proportion of true positives to the sum of true positives and false positives. Precision serves as an indicator of the model's capacity to minimize false positives, thereby elucidating its reliability in affirmatively identifying positive instances.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \tag{4}$$

- **Recall** (5)**:** is quantified as the ratio of true positives to the sum of true positives and false negatives, recall is instrumental in evaluating the model's effectiveness in minimizing false negatives, thus ascertaining its adeptness in recognizing positive instances.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \tag{5}$$

- **F1 Score** (6)**:** is expressed mathematically as twice the product of precision and recall divided by their sum, the F1 score is particularly salient in scenarios marked by imbalanced class distributions. A high F1 score attests to a model's equilibrium between precision and recall, rendering it pertinent in contexts where a harmonious compromise between the two is requisite.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision + Recall}} \tag{6}$$

In this study, we aimed to compare two machine learning approaches: the traditional centralized method and a more advanced technique called Federated Learning. Federated Learning is a state-of-the-art mechanism designed to meet modern security needs. Our goal was to understand and highlight the differences in effectiveness between the basic centralized approach and the advanced Federated Learning framework. This choice is especially relevant as Federated Learning addresses the evolving security requirements in today's computing environments.

In the federated learning (FL) training scenario, three rounds of aggregation were conducted with three participants. Each local training session consisted of a single epoch, employing a batch size of 64 for both training and testing phases. The global model optimization utilized the Adam optimizer with a learning rate set at 0.001, coupled with the Cross-entropy loss function.

Table 4 presents the accuracy results obtained from both centralized and FL approaches. In centralized learning, the model was trained on each introduced dataset, and 30% of the data was reserved for evaluation, as previously detailed. It is noteworthy that FL demonstrated a noteworthy accuracy of 82.91% with 3 rounds of aggregation and 85% with 5 rounds. In contrast, centralized learning achieved a maximum accuracy of 76.9% over 3 epochs and 78.94% over 5 epochs specifically on Dataset_C. These findings underscore the superior performance of FL in comparison to centralized learning, emphasizing its potential for enhanced accuracy and efficacy in the context of our experiments. This discrepancy underscores the potential advantages of federated learning in terms of efficiency and accuracy. The collaborative nature of FL, leveraging decentralized model training across multiple participants, appears to contribute significantly to its ability to outperform centralized learning. This observation suggests that the federated approach may offer a more efficient means of model training, particularly in scenarios where data decentralization and security considerations are paramount.

Table 4: Comparison of the utilization of model in centralized learning and federated learning for 3 datasets over 3 and 5 rounds/epochs.

Rounds/Epochs	Type of Learning	Dataset	Accuracy
3	Centralized	Dataset_A	0.5126
		Dataset_B	0.7649
		Dataset_C	0.769
	Federated		**0.8291**
5	Centralized	Dataset_A	0.5407
		Dataset_B	0.7698
		Dataset_C	0.7894
	Federated		**0.8532**

Due to inherent limitations in resources and our subjective capacity, our validation efforts for the federated learning (FL) workflow were focused on specific metrics outlined in the preceding section. As depicted in Table 5, we present a comprehensive comparison of federated training outcomes over 3 rounds and 5 rounds, aiming to classify input data into one of the five designated classes.

The assessments reveal the effectiveness of our study, particularly showcasing improved performance with a higher number of aggregation rounds. Notably, the F1-Score reaches 0.798 over 3 rounds and 0.8419 over 5 rounds, underscoring the favorable impact of prolonged federated training on the model's classification accuracy.

It is imperative to acknowledge certain nuances in our results, especially regarding the categorization of the Adware class. The relatively low performance in this category can be attributed to the limited number of samples corresponding to this label. Additionally, our employed Convolutional Neural Network (CNN) model, while fundamental and straightforward, may contribute to less optimal predictions for the Adware class when compared to other classes.

Conversely, the evaluations pertaining to SMS Malware yield highly favorable results, particularly evident in the F1-Score of 0.97

Table 5: Evaluation metrics for FL over 3 and 5 rounds

Rounds	Classes	Precision	Recall	F1 Score
3	Adware	0.91	0.2	0.33
	Banking Malware	0.85	0.67	0.75
	Mobile Riskware	0.69	0.84	0.76
	SMS Malware	0.95	0.96	0.95
	Benign	0.78	0.96	0.86
	Average	**0.8326**	**0.8171**	**0.798**
5	Adware	0.96	0.32	0.48
	Banking Malware	0.9	0.74	0.81
	Mobile Riskware	0.77	0.89	0.82
	SMS Malware	0.99	0.95	0.97
	Benign	0.77	0.98	0.86
	Average	**0.8711**	**0.8532**	**0.8419**

achieved over 5 rounds of federated training. This success can be attributed to the substantial quantity of samples associated with the SMS Malware class, as detailed in Table 2.

As illustrated in Figure 4, predictions with 5 rounds of federations on Adware, Banking Malware, Mobile Riskware, and Benign classes are markedly improved, evident in the darker color on the confusion matrices' main diagonals. This reaffirms the positive correlation between higher aggregation rounds and enhanced performance in our study.

However, it is important to note that this study is currently not comprehensive enough. Therefore, the forthcoming sections will delve into discussions regarding potential avenues for future work and enhancements to address existing limitations.

5 CONCLUSION AND FUTURE WORKS

In recent years, the rapid proliferation of Android platforms has been accompanied by a surge in cyber threats due to the inherent vulnerabilities in the open-source architecture. This study addresses the escalating challenge of Android malware by shifting focus towards categorizing variants, offering insights into system vulnerabilities, and prioritizing mitigation strategies. Leveraging static-based techniques, we propose a lightweight system that transforms APK files into RGB images for classification using a CNN.

Recognizing the limitations of centralized ML approaches in the cybersecurity domain, we advocate for the adoption of FL. Our study integrates OpenFL, an efficient and versatile open-source FL framework. Through this approach, we aim to provide a privacy-preserving and nuanced solution to Android malware categorization, bridging advancements in image-based classification with the benefits of decentralized collaboration.

While the current system effectively addresses the classification of Android malware families, ensuring data security and meeting specified requirements, several aspects remain unvalidated. Although our model and federation exhibited promising results with an increased number of federating rounds, a definitive conclusion on its scalability remains pending. The simplicity of our constructed CNN raises questions about its performance in diverse scenarios, prompting the need for further exploration.

Resource constraints and dataset limitations have left our current dataset imbalanced, and we acknowledge the need to assess

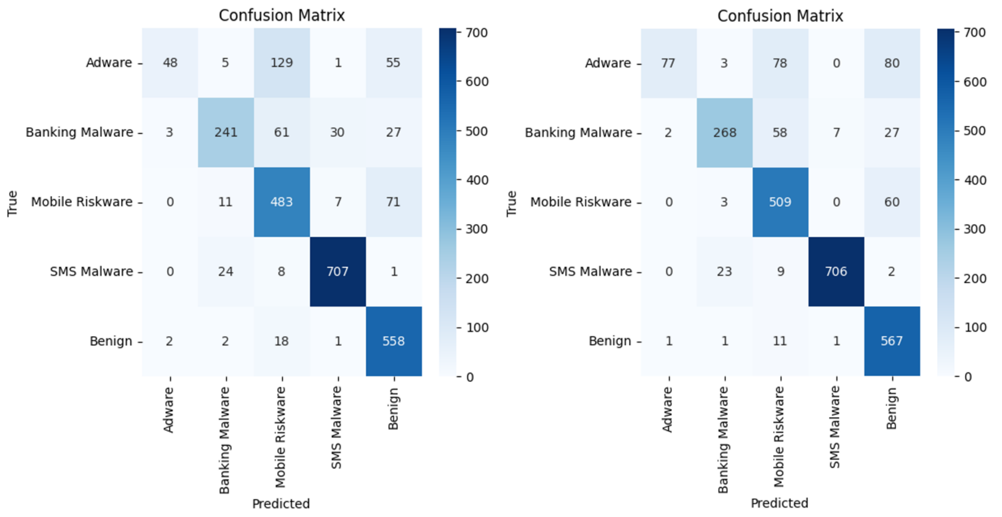


Figure 4: The comparison between 3 and 5 rounds of federation

scenarios where each node equally contributes to model training. Additionally, our modified FedAvg algorithm, while a foundational concept, may not perform optimally with an expanded number of participants. Therefore, our future work will focus on data balancing for federated learning, exploring diverse circumstances to ascertain the system's versatility, and optimizing our aggregation algorithm to enhance performance across varied scenarios. These endeavors aim to fortify the robustness and applicability of our system in real-world scenarios.

ACKNOWLEDGMENTS

This work was supported by our lecturer, PhD. Tan Cam Nguyen. The team is grateful for the support by him since we can have a more comprehensive view of this topic.

REFERENCES

[1] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. 2021. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE Internet of Things Journal* 8, 7 (2021), 5476–5497. https://doi.org/10.1109/JIOT.2020.3030072

[2] Mohammed M. Alani and Ali Ismail Awad. 2022. PAIRED: An Explainable Lightweight Android Malware Detection System. *IEEE Access* 10 (2022), 73214–73228. https://doi.org/10.1109/ACCESS.2022.3189645

[3] Mamoun Alazab, Swarna Priya RM, Parimala M, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Quoc-Viet Pham. 2022. Federated Learning for Cybersecurity: Concepts, Challenges, and Future Directions. *IEEE Transactions on Industrial Informatics* 18, 5 (2022), 3501–3509. https://doi.org/10.1109/TII.2021.3119038

[4] Kunal Chandiramani, Dhruv Garg, and N Maheswari. 2019. Performance Analysis of Distributed and Federated Learning Models on Private Data. *Procedia Computer Science* 165 (2019), 349–355. https://doi.org/10.1016/j.procs.2020.01.039 2nd International Conference on Recent Trends in Advanced Computing ICRTAC -DISRUP - TIV INNOVATION , 2019 November 11-12, 2019.

[5] Asim Darwaish and Farid Naït-Abdesselam. 2020. RGB-based Android Malware Detection and Classification Using Convolutional Neural Network. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. 1–6. https://doi.org/10.1109/GLOBECOM42002.2020.9348206

[6] TonTon Hsien-De Huang and Hung-Yu Kao. 2018. R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based AndroiD Malware Detections. arXiv:1705.04448 [cs.CR]

[7] Saneeha Khalid and Faisal Bashir Hussain. 2022. Evaluating Dynamic Analysis Features for Android Malware Categorization. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*. 401–406. https://doi.org/10.1109/IWCMC55113.2022.9824225

[8] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A. Ghorbani. 2020. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. 515–522. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094

[9] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2023. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629 [cs.LG]

[10] Hamad Naeem, Farhan Ullah, Muhammad Rashid Naeem, Shehzad Khalid, Danish Vasan, Sohail Jabbar, and Saqib Saeed. 2020. Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. *Ad Hoc Networks* 105 (2020), 102154. https://doi.org/10.1016/j.adhoc.2020.102154

[11] Charu Negi, Preeti Mishra, Pooja Chaudhary, Harsh Vardhan, Harsh Vardhan, and Harsh Vardhan. 2021. A Review and Case Study on Android Malware: Threat Model, Attacks, Techniques and Tools. *Journal of Cyber Security and Mobility* (2021). https://doi.org/10.13052/jcsm2245-1439.1018

[12] Cong-Danh Nguyen, Nghi Hoang Khoa, Khoa Nguyen-Dang Doan, and Nguyen Tan Cam. 2023. Android Malware Category and Family Classification Using Static Analysis. In *2023 International Conference on Information Networking (ICOIN)*. 162–167. https://doi.org/10.1109/ICOIN56518.2023.10049039

[13] Rukundo Olivier and Cao Hanqiang. 2012. Nearest Neighbor Value Interpolation. *International Journal of Advanced Computer Science and Applications* 3, 4 (2012). https://doi.org/10.14569/ijacsa.2012.030405

[14] Zinal D. Patel. 2018. Malware Detection in Android Operating System. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. 366–370. https://doi.org/10.1109/ICACCCN.2018.8748512

[15] G. Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, Jason Martin, Brandon Edwards, Micah J. Sheller, Sarthak Pati, Prakash Narayana Moorthy, Hans Shih-Han Wang, Prashant Shah, and Spyridon Bakas. 2021. OpenFL: An open-source framework for Federated Learning. *CoRR* abs/2105.06413 (2021). arXiv:2105.06413 https://arxiv.org/abs/2105.06413

[16] Rajan Thangaveloo, Wong Wang Jing, Chiew Kang Leng, and Johari Abdullah. 2020. DATDroid: Dynamic Analysis Technique in Android Malware Detection. *International Journal on Advanced Science, Engineering and Information Technology* 10, 2 (2020), 536–541. https://doi.org/10.18517/ijaseit.10.2.10238

[17] Xiaolong Xu, Bowen Shen, Xiaochun Yin, Mohammad R. Khosravi, Huaming Wu, Lianyong Qi, and Shaohua Wan. 2021. Edge Server Quantification and Placement for Offloading Social Media Services in Industrial Cognitive IoV. *IEEE Transactions on Industrial Informatics* 17, 4 (2021), 2910–2918. https://doi.org/10.1109/TII.2020.2987994

[18] Wenhui Zhang, Nurbol Luktarhan, Chao Ding, and Bei Lu. 2021. Android Malware Detection Using TCN with Bytecode Image. *Symmetry* 13, 7 (2021). https://doi.org/10.3390/sym13071107

[19] Hui-Juan Zhu, Liang-Min Wang, Sheng Zhong, Yang Li, and Victor S. Sheng. 2022. A Hybrid Deep Network Framework for Android Malware Detection. *IEEE Transactions on Knowledge and Data Engineering* 34, 12 (2022), 5558–5570. https://doi.org/10.1109/TKDE.2021.3067658