An Approach for Privacy-Preserving Mobile Malware Detection Through Federated Machine Learning

Giovanni Ciaramella[1,2][a], Fabio Martinelli[2][b], Francesco Mercaldo[3,2][c], Christian Peluso[2][d] and Antonella Santone[3][e]

[1] *IMT School for Advanced Studies Lucca, Lucca, Italy*

[2] *Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy*

[3] *University of Molise, Campobasso, Italy*

Keywords: Deep Learning, Federated Machine Learning, Mobile, Malware, Security.

Abstract: Considering the diffusion of smart devices and IoT devices, mobile malware detection represents a task of fundamental importance, considering the inefficacy of signature-based antimalware free and commercial software, which can detect a threat only if its signature is present in the antimalware repository. In the last few years, many methods have been proposed by academia to identify so-called zero-day malware through machine learning: these techniques typically extract a series of features from the mobile device to send to a server where the detection model is located. Typically, these features include network traces or installed applications, among other information that may compromise user privacy. In this context, Federated learning is emerging with privacy advantages because the raw data never leaves the local device. In this paper, we propose a method to integrate federated machine learning in malware detection.Malicious software typically aims to extract sensitive and private data, and mobile devices emerge as particularly enticing targets from the perspective of attackers. In the experimental analysis, comprising a pool of 10 clients from which 7 are uniformly sampled at each round, we demonstrate the efficacy of the proposed method by achieving an accuracy of 0.940.

1 INTRODUCTION

Over the years due to the large amount of personal data employed by technological devices, governments in each continent introduced several regulations to preserve the users' privacy. The European Union (EU), in May 2018, introduced the GDPR (General Data Protection Regulation)[1] to protect personal data from possible abuse. In detail, the latter allows for direct identification of a natural person, like name and surname, while from the digital point-of-view, personal data includes IP addresses, biometric data, and location information. The rules imposed by the EU are very stringent, as reported by Statista[2], in 2023, the European Commission imposed 465 fines (2.1 billion euros) due to violations of GDPR. One of the problems related to technological progress is the usage of personal data employed by a large amount of Internet of Things (IoT) devices, which are widely adopted in different fields. An IoT device is any physical object connected to the internet, capable of sending and receiving data for monitoring, control, or automation purposes, and the number of them is bound to grow. The alarming aspect of IoT devices is the potential for attacks and the resulting theft of personal data. In 2016, Google introduced Federated Learning (FL)[3] as a concept to securely manage personal data. In essence, FL offers a solution to tackle privacy concerns linked with centralized machine learning methods (Mercaldo et al., 2022b; Huang et al.,

[a] https://orcid.org/0009-0002-9512-0621
[b] https://orcid.org/0000-0002-6721-9395
[c] https://orcid.org/0000-0002-9425-1657
[d] https://orcid.org/0009-0001-2972-4374
[e] https://orcid.org/0000-0002-2634-4456
[1] https://gdpr-info.eu/

[2] https://www.statista.com/chart/30053/gdpr-data-protection-fines-timeline/
[3] https://blog.research.google/2017/04/federated-learning-collaborative.html?m=1

Ciaramella, G., Martinelli, F., Mercaldo, F., Peluso, C. and Santone, A.
An Approach for Privacy-Preserving Mobile Malware Detection Through Federated Machine Learning.
DOI: 10.5220/0012730600003690
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 1*, pages 553-563
ISBN: 978-989-758-692-7; ISSN: 2184-4992
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

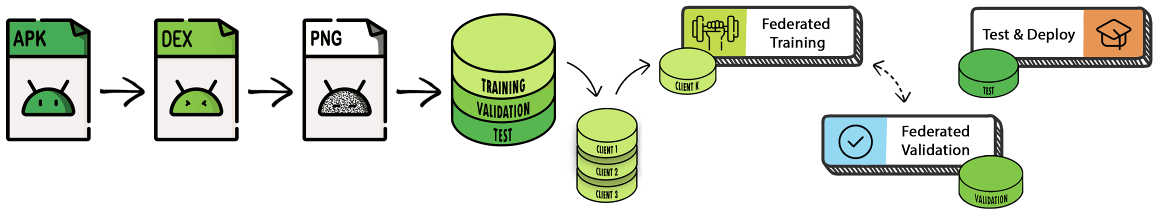

Figure 1: The proposed method aimed to identify malware using Federated Learning to preserve users' privacy.

2024). These methods involve aggregating and storing sensitive user data in a central server. By training machine learning models directly on decentralized devices, while keeping user data local, FL enables collaborative model training without compromising data privacy.
A potential threat is a "data breach" which targets the personal data of numerous users by exploiting various techniques, such as malware. To identify the latter, different methods have been introduced over the years. By the early 2010s, researchers started to employ artificial intelligence (AI) in the malware detection field. In detail, researchers in AI are divided into two main fields: Machine Learning and Deep Learning (Rathore et al., 2018) (Liu et al., 2020). Recently, thanks to the introduction of FL, researchers started to use that also to classify malware in different IoT environments (Lin and Huang, 2020) (Rey et al., 2022).
In this paper, we propose a method to identify malware mobile field precisely in the Android environment leveraging Federated Learning to be compliant with the user's privacy. Specifically, to construct the dataset for training our model, we obtained grayscale images by converting the `.dex` files obtained after reverse engineering from the `.apk` (Android Package) files. Once the dataset was built, we started training the model using the MobileNetV3 architecture, which aimed to be the best mobile vision model.
The paper proceeds as follows: the steps of the method employed are introduced in Section 2, Section 3 presents the results of the experimental analysis; in Section 4 we report the current state-of-art literature about the usage of Federated Learning in the malware detection, and, finally, in the last section, conclusion and future research plan are presented.

2 A METHOD FOR PRIVACY-PRESERVING MOBILE MALWARE DETECTION

This section furnishes an overview of the method proposed to identify malware employing Federated Learning. Figure 1 shows the steps performed in the proposed method. In detail, the latter exhibits several steps springing from the image creation via the conversion of Android applications. Subsequently, we built a proper distribution among the various clients of the simulation dataset employing the images obtained. As a final step, we trained a model employing Federated Learning using an architecture reachable in literature. These aspects will be elaborated upon in detail, as they form the foundation upon which the malware classifier can learn without compromising user privacy, enabling access to data never before available and thereby enhancing the capabilities that AI can achieve.

2.1 Image Generation

In the Android environment, once the source code is developed, the SDK (Software Development Kit) tool is invoked to compile any data and resource files into an APK (Android Package)[4]. The code compiled and contained in the Android Package turns out not to be of understanding to humans as converted into bytecode. To obtain the source code to generate the image, in this paper, we employed APK Tool[5] that helps to disassemble the application under analysis and obtain files into a format that can be edited/read by humans. The files related to the code are saved using the `.dex` extension. The Dalvik Executable code allows developers and security researchers to understand how an application works at a lower level,

[4]https://developer.android.com/guide/components/fundamentals
[5]https://apktool.org/

enabling them to make modifications or extract information from the application. To obtain the dataset to perform the experiments, we extract the dex file from the apk file of the application under analysis and convert it into a grayscale image (Iadarola et al., 2021b), (Iadarola et al., 2021a).
Figure 2 reports an example of `aa9fb22d1f22ceb6edda1677db0e29e3` APK file belongs to the Fusob ransomware converted into an image.

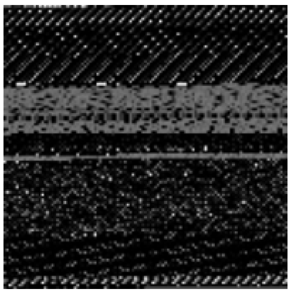
Figure 2: An example of an image obtained from the Android malware with the *aa9fb22d1f22ceb6edda1677db0e29e3* hash belonging to the Fusob ransomware.

2.2 Dataset

The next step was to obtain a balanced dataset to reach good outcomes. In this research paper, we employed a dataset composed of images retrieved from six malware families *Airpush, Downgin, FakeInst, Fusob, Jisut,* and *Mecor*, together with trusted applications.
Employing Federated Learning is crucial to understand how to distribute the data. In detail, we express our distributions in probabilities $P_i \neq P_j$, for different nodes i and j. Using the Bayesian Rule, we can rewrite these probabilities starting from specific data and its label $P_i(x,y)$ as $P_i(y|x)P_i(x)$ and $P_i(x|y)P_i(y)$, the latter allows us to characterize the typologies more precisely (Kairouz et al., 2021):

- **Feature distribution skew:** the marginal distributions $P_i(x)$ may vary across collaborators, even if $P(y|x)$ is shared — programmers' formats can differ or they could provision a peculiar setup before inserting the payload, so the application can differ even if the probability of being a member of a specific malware family is the same;
- **Label distribution skew:** the marginal distributions $P_i(y)$ can change for the edges, even if $P(x|y)$ is the same — clients are tied to particular georegions; WeChat or Alipay are used mainly in Asian countries, certain phishing or ransomware behave only in the English-speaking world;
- **Quantity skew:** each node can hold vastly different amounts of data — users that make heavier use of the smartphone will accumulate more apk files.

Malware classification hinges on each of the aforementioned disparities, as demonstrated in the respective examples. The nature of the application is not synthetic, intended as a process in which the data are constructed starting from a sampling process, such as Monte Carlo or Gibbs. The data have been assembled by scavenging `.apk` files from the web, selecting all the applications with certified malicious code inside. This fact is significant for distinguishing our distribution from a function that could generate data directly for each client. In conclusion, the nature of the problem, coupled with the volume of accumulated data, renders it impossible to address in an environment with a highly non-identically independently distributed dataset (non-IID).

2.3 Federated Learning

In the last few years, the concept of Federated Learning has taken hold. The idea introduced with this technique is to allow the participation of several devices in the process of AI model development, making it more accessible, transparent, and accountable, without compromising the privacy of the collaborators.
The main idea at the bottom of Federated Learning is that each user can participate in collaborative training. In our simulation, each smartphone acts as an edge node, contributing to the collective defense against malware threats. At the heart of this system lies a central command center, orchestrating the FL process and fostering collaboration among the smartphones. Instead of consolidating all data in one central repository, the command center dispatches model parameters to each smartphone, enabling them to conduct local model training using their unique datasets of observed malware behaviors. Through iterative rounds of training, composed in turn of multiple local epochs, smartphones enhance the model's ability to detect and mitigate emerging malware threats, leveraging the diverse insights gleaned from their respective datasets. Personal information remains safeguarded within the confines of individual devices, with only anonymized model updates transmitted to the central command center for aggregation. As the federated learning progresses, the central command center aggregates these refined updates, synthesizing a collective understanding. The group intelligence forms the basis for enhancing the model's efficacy in detecting malware threats. Finally, the central command center disseminates the refined model back to smartphones, empow-

ering them with enhanced classification capabilities, as illustrated in Figure 3.

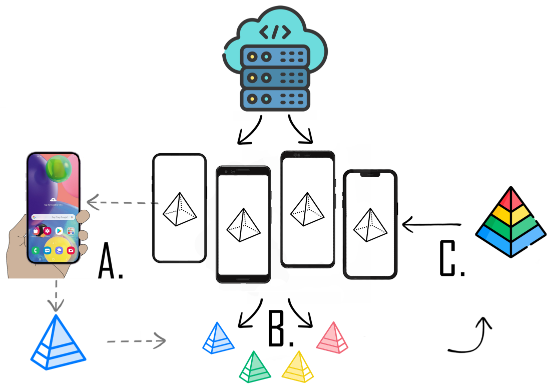

Figure 3: The server deploys a model, each client generates private data (step A) on which the local models fit (step B) and finally, the specialized models are aggregated to refine the global model (step C). The loop is closed by distributing back the enhanced model.

2.4 Experiments and Data Analysis

After acquiring the dataset and introducing the method to preserve client privacy, the next step was to train and test the model. In addition to those two phases, we also employed the validation. The latter allowed us to evaluate the performances of the model. The model underwent training utilizing the MobileNetV3 (Howard et al., 2019) architecture. Before this version, Google presented other two versions: MobileNet (Howard et al., 2017) and MobileNetV2 (Sandler et al., 2018). The first introduced depthwise and pointwise convolution, which simultaneously address spatial and feature generation mechanisms, thereby reducing the computation needed. In contrast, MobileNetV2 adopted linear bottleneck and inverted residual structures. The former influences the latter, as the previous layer dimensions were wide-narrow-wide, whereas inverted convolutions reversed this trend, impacting network performance. To mitigate this, the linear bottleneck prevents activation in the final convolution layer. MnasNet further enhanced MobileNet architectures by incorporating lightweight attention modules based on squeeze and excitation into the bottleneck structure of the final block. MobileNetV3 builds upon these advancements by introducing the hard-swish activation function, which combines the Rectified Linear Unit (RELU) with a sigmoid function to optimize properties near zero.
Additionally, we employed two different techniques to optimize hyperparameters, namely, Grid Search and Random Search. The first one enables the definition of all possible combinations and was utilized to explore various parameter combinations for refining the modified model on the CIFAR-10 dataset, serving as a preparatory phase for the latter. Instead, in Random Search, a predefined search space is specified, and the hyperparameters are randomly selected. In certain scenarios, Random Search can be more efficient than Grid Search in large environments due to its avoidance of complete exploration of all possible combinations. This is the reason why it was employed to search for the expansive federated context.

3 EXPERIMENTAL RESULTS

In this section, we illustrate the results retrieved after the test phase. In detail, in Section 3.1 are reported the outcomes obtained after using the Grid Search technique, while in Section 3.2 the results acquired after the Random Search are shown. The experiments reported in this section were conducted on two different computers. Specifically, the machine on which the Grid and Random Searches were executed features an Intel(R) Xeon(R) Gold 6140M CPU @ 2.30GHz with 16 Cores and 36 threads, along with 64 GB of RAM. For the training phase, a different machine was utilized, equipped with an Intel Core i5-11400 @ 2.60GHz, 6 Cores, and 12 threads, complemented by 32 GB of RAM and an NVIDIA GeForce RTX 3060 with 12 GB of VRAM.

The population of clients from which updates are gathered can be quite large. In our simulation, we limit this to M, and for each round t, we uniformly and randomly select $\mathcal{S}^{(t)}$ clients. In practice, the sequence of active clients, or the cohort size, is typically influenced by complex circumstances beyond the control of the orchestrating server. For example, mobile devices might only participate when idle, connected to specific unmetered networks, and charging (Wang et al., 2021).

In this paper, we employed a dataset composed of images generated from Android applications. In detail, these applications belong to six different malware families with a total of 7,386 malicious samples divided into 1,170 Airpush, 763 Downgin, 2,129 FakeInst, 1,275 Fusob, 550 Jisut, and 1,509 Mecor, and 1,060 trusted applications (Mercaldo et al., 2022a; Iadarola et al., 2021b). Once the dataset was acquired, we partitioned it into training, validation, and testing, adhering to a split proportion of 65-15-20, as reported in Table 1. This allocation ensures that 65% of the data is dedicated to training the model, 15% for validating its performance during training, and the remaining 20% for final evaluation, thereby maintaining a balanced distribution for robust model assessment. The training data is distributed among the clients to start the federated simulation. The dynamics of this process are independent and identically distributed (IID), as illustrated by the example distribution across $M = 7$ clients in Figure 4.

Table 1: Number of samples per class of malware and relative subdivision.

Class Name	# Training	# Validation	# Test
Airpush	749	187	234
Dowgin	489	122	152
FakeInst	1364	340	425
Fusob	816	204	255
Jisut	352	88	110
Mecor	960	240	299
Trusted	678	170	212
Total	5408	1351	1687

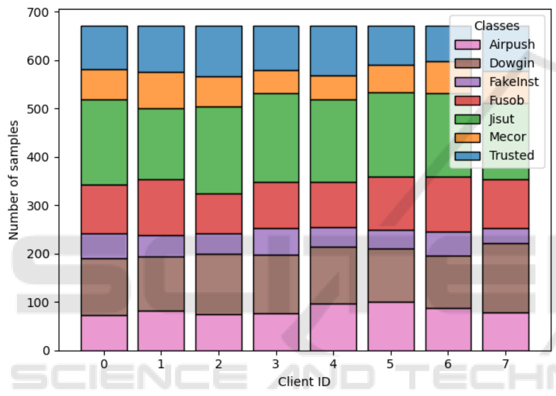

Figure 4: The classes follow the most suitable distribution, being identically distributed on each local dataset.

Concluding the dataset-building phase, we trained the MobileNetV3 architecture, obtaining the results discussed in the following sections. In addition, to clarity, reproducibility, and comparative analysis, in the experiment conducted, we utilized the loss function "Categorical Focal Cross Entropy" (Lin et al., 2018). The latter was used to push the training to focus on miss represented classes, especially considering the dataset's particular imbalance:

$$\mathcal{L}(p_t) = \delta \cdot (1 - p_t)^{\gamma} \cdot \text{CrossEntropy}(y_{\text{true}}, y_{\text{pred}}) \quad (1)$$

In Equation 1, it is possible to identify the parameter δ as the weight factor for the classes and γ as a parameter that focuses on the misclassified samples, encouraging the gradient to concentrate on the minor classes. Thus, setting $\gamma = 0$ reduces the function to a classical Categorical Cross Entropy; categorical since the labels are provided in a one-hot encoding representation.

In our setup, we employ two optimizers: one designated for the server ($_S$) and another replicated for each client ($_C$), along with their respective models. While it's possible to customize the optimizer for the server and clients separately, we opted for Stochastic Gradient Descent with Momentum (SGDM) for both. Consequently, we have the learning rates η_S and η_C, as well as the momentums β_S and β_C. The client's optimizer operates at each local epoch (τ), whereas the server optimizer only engages when it needs to aggregate the collaborators' updates occurring at the end of each round (T).

The de facto standard algorithm, thus more commonly preferred, for the cross-device setting is Federated Average (FedAvg). Where each client performs multiple updates in a single round. This way, the aggregated gradient will retain more information, and the server could require fewer rounds to converge, thus saving communication costs.

With the introduction of schedulers, we used symbols to represent their variables: υ denotes the minimum learning rate, while ν signifies the decay steps with the `CosineDecay()` scheduler. Conversely, ϕ and φ correspond to the decay steps and rate with the `InverseTimeDecay()` scheduler.

3.1 Grid Search

To discover the optimal parameter combination, we initiated a Grid Search, followed by selecting the resultant parameters to standardize the model's capacity across all layers. This step served as a preparatory phase for the subsequent Random Search.
For clarity, we instantiated the MobileNetV3 model and utilized TensorFlow's Keras method to save its weights. Each weight was associated with a layer name, which we used to reload them into the modified architecture. Specifically, each layer retained the same name, except for the custom ones. Subsequently, we iterated through each layer of the newly created model to set the *trainable* flag to True exclusively for the GroupNorm layers. This approach allowed us to freeze and preserve the training of the original architecture, acknowledging the efforts invested by Google in the ImageNet dataset, which is composed of wider images, like those in our dataset. This type of search aims to investigate the weight parameters over a lower bound of epochs, ensuring that extensive training results in a well-fitted architecture. Therefore, all results presented pertain to the training of the CIFAR-10 dataset limited to 100 epochs and optimized with the Root Mean Square Propagation (RMSProp) algorithm. The stopping criterion is applied when the validation loss value has decreased by a threshold of $\varepsilon = 1e-3$ for 5 successive epochs.

To visualize the relationships among the selected

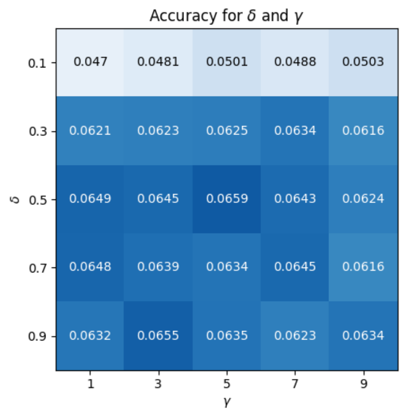

Figure 5: Combination of γ and δ to achieve best performances in terms of accuracy.

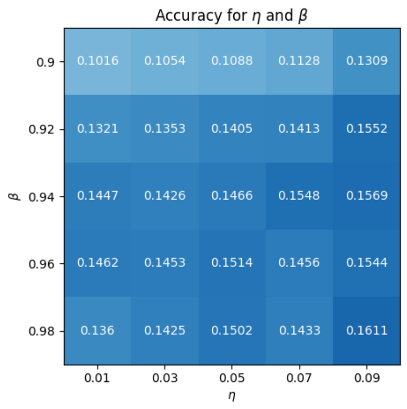

Figure 6: The η and β parameters significantly contribute to the accuracy results.

hyperparameters, we utilized a confusion matrix. In detail, in Figure 5, we illustrate the relationship between the δ and γ parameters that dictate the weighting factor for the classes and the focus on relatively easy-to-classify samples. It is possible to denote that the accuracy increases when the parameters are proportionate; indeed, the combination of $\gamma = 5$ and $\delta = 0.5$ achieves the highest score. It's noteworthy to recall that with $\gamma = 0$, we have the standard CrossEntropy loss, and the Focal version becomes unstable without the weighting factor δ. These observations are confirmed by our experiments, which show the peak of accuracy when the parameters are properly tuned.

In addition, Figure 6 illustrates that utilizing higher learning rates and momentums enabled the model to better understand the dataset. We observe that increasing the values of η and β parameters significantly improves classification performance with this model and dataset. One possible explanation could be that the embedding created by MobileNetV3 effectively represents the images, thereby necessitating only minor adjustments to the weights of the dense layers.

3.2 Random Search

Built the model, the next step was to proceed to the hyperparameter search phase in the federated environment. Given its broader scope, this required thorough preparation. In this subsection, the search phase is described step by step.

In detail, all the experiments were conducted with consistent parameters: the same number of epochs (τ == 1), rounds (T == 15), number of distributed clients (M == 10), number of sampled clients ($\mathcal{S}^{(t)}$ == 7), training batch size (32), and validation/test batch size (128).

When retraining or refitting the best combination, the operations remained consistent as well, with the same number of epochs (τ == 3), rounds (T == 12), training batch size (32), and validation/test batch size (128). The dimensions of distributed and sampled clients were kept equal.

Each of the random searches has the results reported in Table 2. We conclude the section by describing the thought process involved in searching for hyperparameters:

1. On top of the discussed architecture, we append a funneling-style neural network (NN) built with successive layers, each one half the size of the previous layer, starting from 64 neurons. Everything except the last part of the NN is kept frozen, allowing us to utilize MobileNetV3 as an embedding capable of expressing the images in compressed vectors. Starting with simplicity, we utilize the standard SGD optimizer with momentum.
2. Since the complexity of the funneling-style neural network (NN) is not necessary, we halved the number of neurons, hoping that reducing the size of the layers would decrease complexity and facilitate the training phase.
3. The additional Feed Forward dense NN layer was not beneficial; instead, it slowed down the convergence rate. In our final attempt, we experimented with just a single additional layer. Moreover, we noted that the client momentum parameter β_C positively influenced the learning rate η_C, speeding up the descent towards local minima when successive directions were similar. Consequently, we kept $\beta_C \approx 1$ and directed our attention towards optimizing η_C.
4. The server should receive all updates from the clients, so we kept the parameters η_S and β_S at their default values. However, realizing that a

Table 2: Here is the loss and accuracy corresponding to the training and validation datasets, along with the elapsed time of all the runs. These results represent the best values per random search, as we have reported the identifiers of each simulation out of the total conducted.

ID/Total	$\mathcal{L}(p_{(\mathbf{TR})})$	$\mathcal{A}(p_{(\mathbf{TR})})$	$\mathcal{L}(p_{(\mathbf{VA})})$	$\mathcal{A}(p_{(\mathbf{VA})})$	**Time**
34/78	0.732	0.769	0.771	0.764	16988 s
117/117	0.872	0.722	0.874	0.748	29367 s
2/60	0.698	0.795	0.728	0.800	16104 s
15/36	0.638	0.801	0.573	0.833	6214 s
2/48	0.605	0.814	0.567	0.838	8693 s
10/72	0.604	0.809	0.574	0.817	14767 s
60/72	0.627	0.804	0.546	0.824	12726 s

fixed η_C was constraining the convergence rate, we employed a scheduler to regulate the rate of convergence, particularly in the initial stages where the learning rate might be sufficiently large to overshoot a satisfactory minimum. We opted for the `CosineDecay()` function as the chosen scheduler.

5. Following theoretical considerations, a value of $\eta \sim 0$ does not permit convergence as $T \to \infty$, while a value of $\eta \sim 1$ may cause oscillations near the minimum. Therefore, we increased the decaying step and its initial value but decreased the minimum.
6. The scheduler previously used decreased the learning rate with a sinusoidal curve, providing less control over the initial phase. Hence, we opted for another scheduler, the `InverseTimeDecay()`, which allows for decreasing the learning rate with a fixed step and rate. Another benefit of this scheduler is that it can act as a stopping criterion when $\alpha = 0$ is reached.
7. We have refined the search by selecting the best parameters from the last round. As observed, a higher initial learning rate can lead to faster convergence towards the minima, and subsequently decaying it with a smaller step can refine the gradient descent process.

3.3 Fine Tuning and Results

To identify the best parameters, we conducted several experiments. In this subsection, we present the outcomes obtained.
To accomplish the grid search, as outlined in Section 3.1, and thus prepare the model for deployment in the federated environment, we fine-tuned the customized architecture on the CIFAR-10 dataset for 400 epochs. This experiment yielded an accuracy of 0.423 and a loss of 0.369 on the test set. However, these results did not achieve satisfactory accuracy. The underlying reason for these values lies in the frozen architecture,

Table 3: Here are all the hyperparameters chosen for the refitting phase.

Sched.	ITD	β_C	0.99
η_S	1	η_C	9e-2
ϕ	100	φ	0.1
γ	5	δ	0.5

which imposes strict control over the convergence capabilities.
Building upon the insights gained from the model during both the grid search and the hyperparameters derived from the random search, as listed in Table 3, we commenced the training phase and subsequent analysis of the results. The training phase consisted of $\tau = 3$ local epochs and $T = 30$ rounds, utilizing the FedAvg algorithm and the Inverse Time Decay (ITD) scheduler, with a population of $M = 10$ clients and a cohort size $\mathcal{S}^{(t)} = 7$.
The rate of convergence of the model through the federated learning rounds is plotted in Figure 7. In the latter figure, both losses (during the training and validation phases) decrease almost simultaneously. In Table 4, the outcomes obtained are reported. These results demonstrate that the model can accurately classify data distributed across the federated community. Upon examination of the performance metrics, we note that the variant with Group Normalization achieves a better loss value for both the training and test sets, albeit with a slightly lower accuracy. This could be attributed to the Categorical Focal Cross Entropy, which considered some samples to be relatively easy to classify, thus reducing their importance during training, similar to how it is used to compute samples within the test set.

In addition, the obtained results were also presented alongside the classification accuracy for each detected malware family, depicted in a confusion matrix reported in Figure 8. As observed, the model demonstrates robust recognition capabilities across most families, except Dowgin and Trusted samples. The model's predictions were met with perplexity due to dataset imbalances or similarities among these two

Table 4: As we can discern from the table, the results appear promising. The MobileNetV3 architecture yields a simple yet effective embedding of the images, facilitating convergence in both classical and federated learning strategies. In this scenario, both Batch Normalization and Group Normalization produce comparable results. However, when training capacities vary among nodes, the utilization of BN may adversely affect global training capabilities due to fluctuations in batch sizes.

Model	$\mathcal{L}(p_{(\mathbf{TR})})$	$\mathcal{A}(p_{(\mathbf{TR})})$	$\mathcal{L}(p_{(\mathbf{VA})})$	$\mathcal{A}(p_{(\mathbf{VA})})$	$\mathcal{L}(p_{(\mathbf{TS})})$	$\mathcal{A}(p_{(\mathbf{TS})})$
Centralized w. BN	0.032	0.933	0.041	0.933	0.035	0.930
Centralized w. GN	0.025	0.918	0.050	0.880	0.044	0.875
Federated w. BN	0.023	0.938	0.037	0.930	0.040	0.940
Federated w. GN	0.010	0.961	0.037	0.910	0.036	0.923

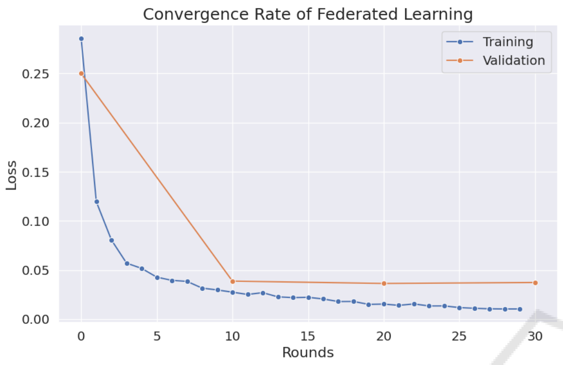

Figure 7: Loss trend during the training and validation phases.

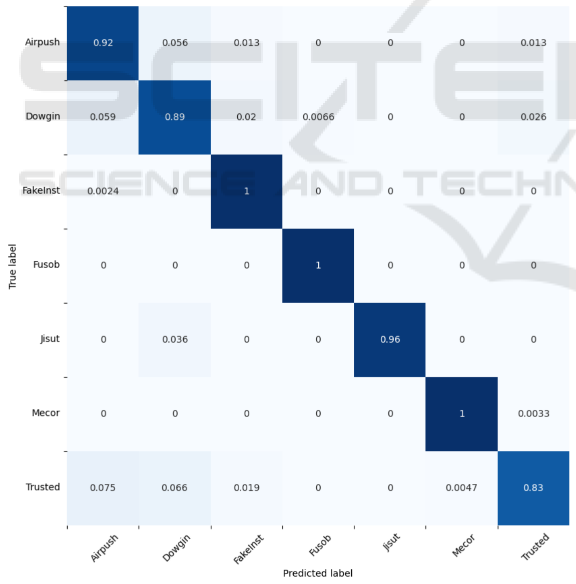

Figure 8: Confusion matrix obtained with the classification of the test set.

converted applications.

4 RELATED WORK

Federated Learning usage is catching on in different fields thanks to the possibility of creating collaborative model training without compromising data privacy. In 2016, Google introduced FL in the literature for the first time to improve language prediction in Gboard. Recently, it has also been employed in other fields like malware detection. In detail, we furnish an overview of different work conducted in malware detection using Machine Learning and Deep Learning in Section 4.1, while in Section 4.2 a resume about the usage of Federated Learning is provided.

4.1 Malware Detection

Authors in (Kim et al., 2018) proposed a framework able to identify malware in the Android environment using 200 diverse features. This article represents the first attempt to apply multimodal neural networks in malware detection. After training the models, they achieved outstanding results, attaining an accuracy rate of 98%. This outcome was realized using a vast dataset comprising over 30,000 images sourced from both malware samples and trusted Android applications.

Researchers in (Li et al., 2018) presented an approach to identify dangerous applications through permissions. In the Android ecosystem, permissions are required to execute particular actions or access specific resources on the device. In SIGPID Li *et al.* employed machine learning reached an accuracy, precision, recall, and F-Measure of 90%.

Thanks to advances in technology, the quantum topic is gaining momentum nowadays. In (Mercaldo et al., 2022a), authors presented for the first time the usage of quantum machine learning in the malware detection field. In detail, they proposed a comparison between several state-of-the-art architectures and two quantum models (*i.e.*, a hybrid quantum model, and a fully quantum neural network) using a dataset composed of almost 9,000 images of malicious and legitimate Android applications.

4.2 Federated Learning

Authors in (Gálvez et al., 2020) presented "LiM" a framework based on Federated Learning able to classify malware. Gálvez *et al.* employed a semi-

supervised classification on a dataset composed of 50,000 apps. As a result, they achieved an F1 score of 95% on the cloud server, while clients obtained just one false positive over 100 apps.
Federated Learning is used in several areas of cybersecurity. Authors in (Khramtsova et al., 2020) presented an FL approach to identify malicious URLs within a managed security service provider context, with consideration of different data characteristics and customer infrastructure sizes.
Non-identicalness remains an open problem, and most research fails to address this issue adequately when discussing Federated Learning. However, authors in (Jiang et al., 2022) have developed a multi-dimensional model comprising a CNN and a Graph Neural Network (GNN), aggregated using the custom algorithm FedAdapt, which dynamically adapts to the best contributor nodes. This method could address variance concerns by initializing sub-optimal weights for the global model and then adapting them locally. Their approach differs from ours in that they address the limitations of CNN by leveraging the capabilities and simplicity of the GNN. In contrast, we propose the use of Group Normalization to address the limitations of FL.
Researchers in (Abdel-Basset et al., 2022) proposed for the first time the usage of agnostic Federated Learning. These techniques aim to be more versatile and scalable, making them suitable for a wide range of applications and settings, including those with limited resources or varying data privacy requirements. In the framework proposed, they obtained the images by converting the bytecode from malware applications. Different from them, we focused our article on the FL applied to the Android environment.
Another approach to addressing the limitations of federated learning involves evaluating contributions from edge nodes. If a contribution effectively improves the global model, it is chosen, thereby enhancing the final distributed model. This idea is explored in (Chaudhuri et al., 2023), where the DW-FedAvg algorithm dynamically weights the effectiveness of gradient updates proposed by clients. While this approach can contribute to the learning rate of federated learning methodologies in non-iid situations, we opt for the FedAvg algorithm, which minimizes the need for additional communications between the server and clients, involving a learning rate scheduler client side in order to better control the rate of convergence.
In (Fang et al., 2023), researchers introduced a novel framework called “FEDroid,” leveraging Federated Learning principles. To obtain a constantly updated framework, Fang *et al.* innovatively employed genetic evolution. This approach simulated the evolutionary dynamics of malware, facilitating the generation of diverse Android malware variants from conventional samples. To train the model they employed a literature dataset named Drebin. Unlike them, we used a dataset built by the authors composed of images obtained from the Dalvik Executable code retrieved after the reverse engineering of the Android Package.

5 CONCLUSIONS AND FUTURE WORK

In conclusion, our work aligns well with the existing landscape of federated projects focusing on malware classification. However, some avenues for further development could enhance the realism of our work, bridging the gap between simulations and the deployment of the federated classifier.
The dataset used for this problem is relatively small, especially considering the distributed nature of the FL environment. As new clients join, the dataset shards become smaller, presenting challenges for local convergence. As a future endeavor, we aim to explore larger datasets tailored to the problem domain or investigate synthetic data generation methods capable of accurately reproducing malware characteristics. This would enable us to experiment with Latent Dirichlet Allocation (LDA) to better understand the significance of clustered data and to increase the label distribution skew, without making prior assumptions.
Moreover, the data preparation process, which includes decompilation, conversion, and labeling of applications, could be automated within a client-side application. Such automation would streamline the initial steps of a decentralized malware classifier. It’s essential to eliminate the need for specialized tools, such as Grad-CAM, from the malware analyst’s toolkit. Alternatively, we could integrate these tools to forward malicious code to an automated malware analyst, such as a Large Language Model (LLM). This approach would enhance the interpretability of the proposed method and make it accessible to non-expert users.
Additionally, it’s imperative to analyze data communication, model deployment, and update protocols in a distributed environment to assess traffic load and performance overhead on the client side. This comprehensive evaluation will help us ascertain the possibilities and limitations of the proposed methodology.

ACKNOWLEDGEMENTS

This work has been partially supported by EU DUCA, EU CyberSecPro, SYNAPSE, PTR 22-24 P2.01 (Cybersecurity), and SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the EU - NextGenerationEU projects.

REFERENCES

Abdel-Basset, M., Hawash, H., Sallam, K. M., Elgendi, I., Munasinghe, K., and Jamalipour, A. (2022). Efficient and lightweight convolutional networks for iot malware detection: A federated learning approach. *IEEE Internet of Things Journal*, 10(8):7164–7173.

Chaudhuri, A., Nandi, A., and Pradhan, B. (2023). *A Dynamic Weighted Federated Learning for Android Malware Classification*, page 147–159. Springer Nature Singapore.

Fang, W., He, J., Li, W., Lan, X., Chen, Y., Li, T., Huang, J., and Zhang, L. (2023). Comprehensive android malware detection based on federated learning architecture. *IEEE Transactions on Information Forensics and Security*.

Gálvez, R., Moonsamy, V., and Diaz, C. (2020). Less is more: A privacy-respecting android malware classifier using federated learning. *arXiv preprint arXiv:2007.08319*.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. (2019). Searching for mobilenetv3.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, P., Li, C., He, P., Xiao, H., Ping, Y., Feng, P., Tian, S., Chen, H., Mercaldo, F., Santone, A., et al. (2024). Mamlformer: Priori-experience guiding transformer network via manifold adversarial multi-modal learning for laryngeal histopathological grading. *Information Fusion*, page 102333.

Iadarola, G., Casolare, R., Martinelli, F., Mercaldo, F., Peluso, C., and Santone, A. (2021a). A semi-automated explainability-driven approach for malware analysis through deep learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Iadarola, G., Martinelli, F., Mercaldo, F., and Santone, A. (2021b). Towards an interpretable deep learning model for mobile malware detection and family identification. *Computers & Security*, 105:102198.

Jiang, C., Yin, K., Xia, C., and Huang, W. (2022). Fedhgcdroid: An adaptive multi-dimensional federated learning for privacy-preserving android malware classification. *Entropy*, 24.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Eichner, H., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. (2021). Advances and open problems in federated learning.

Khramtsova, E., Hammerschmidt, C., Lagraa, S., and State, R. (2020). Federated learning for cyber security: Soc collaboration for malicious url detection. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 1316–1321. IEEE.

Kim, T., Kang, B., Rho, M., Sezer, S., and Im, E. G. (2018). A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security*, 14(3):773–788.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., and Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225.

Lin, K.-Y. and Huang, W.-R. (2020). Using federated learning on malware classification. In *2020 22nd international conference on advanced communication technology (ICACT)*, pages 585–589. IEEE.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2018). Focal loss for dense object detection.

Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., and Liu, H. (2020). A review of android malware detection approaches based on machine learning. *IEEE Access*, 8:124579–124607.

Mercaldo, F., Ciaramella, G., Iadarola, G., Storto, M., Martinelli, F., and Santone, A. (2022a). Towards explainable quantum machine learning for mobile malware detection and classification. *Applied Sciences*, 12(23):12025.

Mercaldo, F., Zhou, X., Huang, P., Martinelli, F., and Santone, A. (2022b). Machine learning for uterine cervix screening. In *2022 IEEE 22nd International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 71–74. IEEE.

Rathore, H., Agarwal, S., Sahay, S. K., and Sewak, M. (2018). Malware detection using machine learning and deep learning. In *Big Data Analytics: 6th International Conference, BDA 2018, Warangal, India, December 18–21, 2018, Proceedings 6*, pages 402–411. Springer.

Rey, V., Sánchez, P. M. S., Celdrán, A. H., and Bovet, G. (2022). Federated learning for malware detection in iot devices. *Computer Networks*, 204:108693.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

Wang, J., Charles, Z., Xu, Z., Joshi, G., McMahan, H. B., y Arcas, B. A., Al-Shedivat, M., Andrew, G., Avestimehr, S., Daly, K., Data, D., Diggavi, S., Eichner, H., Gadhikar, A., Garrett, Z., Girgis, A. M., Hanzely, F., Hard, A., He, C., Horvath, S., Huo, Z., Ingerman, A., Jaggi, M., Javidi, T., Kairouz, P., Kale, S., Karimireddy, S. P., Konecny, J., Koyejo, S., Li, T., Liu, L., Mohri, M., Qi, H., Reddi, S. J., Richtarik, P., Singhal, K., Smith, V., Soltanolkotabi, M., Song, W., Suresh, A. T., Stich, S. U., Talwalkar, A., Wang, H., Woodworth, B., Wu, S., Yu, F. X., Yuan, H., Zaheer, M., Zhang, M., Zhang, T., Zheng, C., Zhu, C., and Zhu, W. (2021). A field guide to federated optimization.