Information and Software Technology 189 (2026) 107892

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A method for real-world privacy-preserving Android malware detection through Federated Machine Learning

Giovanni Ciaramella [b,a,*], Fabio Martinelli [c], Christian Peluso [b], Antonella Santone [d], Francesco Mercaldo [d,b]

[a] IMT School for Advanced Studies Lucca, Lucca, Italy
[b] Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy
[c] Institute for High Performance Computing and Networking, National Research Council of Italy (CNR), Rende, Italy
[d] University of Molise, Campobasso, Italy

ARTICLE INFO

Keywords:
Malware
Mobile
Machine learning
Federated Machine Learning
Privacy
Android
Security

ABSTRACT

Privacy is one of the most critical issues associated with spreading the Internet of Things and Internet of Everything devices. Over the years, several methods have been introduced to address this phenomenon. In 2017, Google introduced the concept of Federated Machine Learning. This paradigm allows models to be trained collaboratively across multiple decentralized devices or servers, holding local data samples without exchanging them. This approach enhances data privacy and security by ensuring raw data remains on local devices while only model updates are shared and aggregated. This paper presents a privacy-preserving Android malware detector based on Federated Machine Learning. As a first step, we built a dataset comprising over 40,000 Android applications, including trusted and malicious (belonging to 71 malware families) samples. Afterward, we conducted experiments leveraging three different architectures by exploiting the CIFAR-10 and the ImageNet datasets, employing hyperparameters determined through a Grid Search algorithm by exploiting 40 clients. Moreover, the experimental analysis uses two distributions: Independent and identically distributed and non-independent and identically distributed data. To conclude the Federated Machine Learning experiments, we trained models for each architecture, with both weight types and distribution models, by applying the Clipping Norm Aggregator. The results exhibit interesting performances with Independent and identically distributed data, achieving an accuracy of 0.873 without normalization and 0.877 with the Clipping Norm aggregator. However, with non-independent and identically distributed data, the model accuracy equals 0.865 without normalization, 0.864 with the Clipping Norm aggregator using Custom MobileNet 2. In conclusion, to compare Federated Machine Learning with a centralized training approach, we trained several models adopting the same dataset, dataset splitting, and architectures, achieving an accuracy of 0.944 using InceptionV3. The outcomes show that the proposed method can provide engaging performances in privacy-preserving Android malware detection.

1. Introduction

In recent years, malware has become one of the most discussed topics in cybersecurity. Cyberattacks caused by malware have increased over the years, as documented in a report published by Statista in April 2024,[1] worldwide malware attacks reached 6.06 billion in 2023, with an increase of 10% compared to the previous year. Attackers began to create several malware variants to evade the detection of well-known malware. Thanks to the progress of the research and the introduction of new methods to limit this phenomenon, in 2022, detectors blocked nearly 700 million malware hidden in apparent secure files.[2] The most prevalent malware types intercepted were Worms (205 million), Viruses (204 million), and Ransomware (106 million). In detail, attackers adopt Worms to create botnets or deliver other malware and viruses to gain control over systems, steal data [1], or produce backdoors [2]. In addition, Ransomware are used to lock users out of systems, extorting money from victims for restoration [3]. The main problem uniting these types of malware is the collection and selling of

* Corresponding author at: Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy.
E-mail addresses: giovanni.ciaramella@imtlucca.it, giovanni.ciaramella@iit.cnr.it (G. Ciaramella), fabio.martinelli@icar.cnr.it (F. Martinelli), christian.peluso@iit.cnr.it (C. Peluso), antonella.santone@unimol.it (A. Santone), francesco.mercaldo@unimol.it, francesco.mercaldo@iit.cnr.it (F. Mercaldo).

[1] https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/
[2] https://www.statista.com/statistics/1379328/malicious-files-malware-types-blocked/

https://doi.org/10.1016/j.infsof.2025.107892
Received 31 July 2024; Received in revised form 26 August 2025; Accepted 30 August 2025
Available online 3 October 2025

personal data to construct profiling systems, generative models capable of mimicking our voice or face, and tools for guessing passwords using a dictionary of common words. In particular, the latter enables the direct identification of an individual, such as by their name and surname. From a digital perspective, personal data encompasses IP addresses, biometric data, and location information. Personal data is a hot topic during the new millennium, and to protect them, several governments such as Australia, China, and the United Kingdom have introduced regulations to preserve usage [4]. Also, the European Union (EU) in 2018 introduced a specific regulation named GDPR[3] (General Data Protection Regulation). The latter aims to strengthen and unify data protection for individuals within the EU and the European Economic Area (EEA).

Protecting personal data becomes a topic of greater importance considering the growing adoption in the most disparate daily practices of Internet of Things (IoT) devices in several domains like smart homes and healthcare [5–8]. The adoption of these devices currently represents a real and serious issue in terms of security. IoT devices collect and transmit a vast amount of personal data, often without users' explicit awareness. This data can include sensitive information such as location, health metrics, and behavioral patterns. To be compliant with the users' privacy and improve the usage of Artificial Intelligence (AI), Google introduced the concept of Federated Machine Learning (FML) in 2017 to securely manage personal data [9]. The role of FML is to offer a solution to tackle privacy concerns linked with centralized machine-learning methods. These techniques entail collecting and storing sensitive user data in a central server. Federated Machine Learning enables collaborative model training without threatening data privacy by training machine learning models directly on decentralized devices while keeping user data local [10,11].

Starting from these considerations, we design and develop a privacy-preserving method for Android malware detection exploiting FML in this paper. For this purpose, we obtained over 40,000 Android *.apk* files, evenly split between 23,431 malware and 23,431 trusted applications. These applications were then converted into images using a Python script developed by the authors. After the composition of the dataset using images, we trained multiple models using three different architectures: two custom versions of MobileNetV3 and InceptionV3, leveraging CIFAR-10 and ImageNet weights and using a grid search to determine the best hyperparameters. Furthermore, we adopted two distributions, Independent and identically distributed (IID) and non-Independent and identically distributed (non-IID), where the latter helped us better represent a real-world scenario. To conclude experiments using FML, for each architecture with both weight types and distribution models, we trained models by applying the Clipping Norm Aggregator (CNA) to assess the performance of the global model after integrating thresholded contributions from multiple client devices. Finally, using the same dataset, dataset splitting, and architectures, we also trained and tested different models using a centralized approach.

The paper proceeds as follows: Section 2 discusses the current state-of-the-art literature in malware detection, Section 3 describes in detail the proposed privacy-preserving malware detection method; the results of the experimental analysis are reported and discussed in Section 4. Finally, the last section includes the conclusion and outlines future research plans.

2. Related work

Through the technological and software advancements of the last decade, various malware identification methods have been proposed in the literature, mainly based on machine learning. Given the rapidly increasing growth curve, researchers introduced several mitigation solutions using machine learning. Due to the latter, we considered only research articles that were no older than five years. The proposed method focuses on threat identification in small-end devices, such as IoTs and smartphones. The methodology presented is implemented by leveraging machine learning and deep learning techniques.

The rest of the section is organized as follows: we will initiate our journey on specific proposals of malware detection through the wise use of machine learning in Section 2.1 and delve into classifiers that preserve the privacy of end-users in Section 2.2. In Table 1, a summary of the methodology proposed by similar works is reported, highlighting core differences and performances achieved.

2.1. Machine learning for malware detection

The paper [12] explores the application of two machine learning algorithms, Support Vector Machine (SVM) and K-Nearest Neighbors (KNN), for classifying Android apps as benign or malicious through supervised learning. The study involves static analysis of apps by examining the presence and frequency of keywords in their manifest files, deriving static feature sets from a dataset of 400 apps. The performance of SVM and KNN is evaluated based on accuracy and true positive rate. The experimental results show that SVM achieves an average accuracy of 0.790, while KNN achieves an average of 0.805.

The authors in [17] correctly point out that a robust malware detector should have a preliminary phase in which malware is clustered into categories. This allows the detector to customize its capabilities for each sample, enhancing performance. Features from the dataset are extracted through Principal Component Analysis (PCA) and then fed into a Modified Particle Swarm Optimization (MPSO) algorithm. The proposal is a working demo of the methodology, achieving good results.

A hybrid analysis approach is essential to provide a comprehensive overview of applications. In [13], the authors extract static and dynamic features, enabling a more robust understanding of the system's behavior and performance. Specifically, they target API calls matching some keywords during the reverse engineering of the app and capture the packet sizes traveling out of the sandbox to understand dynamic behavior. The model used is a Support Vector Machine, and the authors compare the partial and complete visions of the features extracted and their relative accuracies.

A systematic review in [16] compares literature using a standard benchmark in antimalware systems. It shows that competitive results can be achieved by correctly extracting features, even when employing just static analysis. The article compares SVM, Decision Tree (DT), and N-Gram models, evaluating the most powerful, the most transparent, and those capturing language features.

The paper [15] proposes a machine-learning model for Android malware detection based on identifying unusual combinations of permissions and API calls requested by malware compared to benign applications. The authors created a new dataset of these co-existing features to validate this approach at various combination levels (second to fifth levels). They applied these features in different contexts: permissions only, APIs only, permissions and APIs, and API frequencies. The frequent pattern growth (FP-growth) algorithm extracted the most relevant features from Android APK samples from the Drebin, Malgenome, and MalDroid2020 datasets. Researchers then used several conventional machine-learning algorithms to evaluate the model. The results demonstrated high accuracy in classifying Android malware, with the best performance (0.980 accuracy) achieved by the Random Forest algorithm using second-level permission combinations.

Finally, the study by [21] elucidates the disparities between conventional antivirus systems reliant on statistical and pattern matching, which can be easily circumvented if not regularly updated, and machine learning algorithms that enable the detection of typical behaviors and comprehension of threats posed by malicious scripts. Utilizing methods such as RNNs and Decision Trees, they evaluate the balance between accuracy and transparency, providing accessibility from a human perspective to delve into the model's reasoning. Their work, based on a Convolutional Neural Network, allows for explainability through the prospective use of deconvolution methods.

[3] https://gdpr.eu/

Table 1
Comparison of state-of-the-art malware detection: centralized and federated approaches.

References	NS	Model	Classification method	Train	I	NI	Aggregator	DR
[12]	400	KNN	Permissions, Intent Filters, Receivers	C	–	–	–	0.805
[13]	21 000	SVM	API Calls, Net Packets	C	–	–	–	0.998
[14]	61 730	SVM	Permissions, LDA, API Calls	F	✓	–	Secure	0.914
[15]	17 341	RF	Permissions, Smali, API Calls	C	–	–	–	0.980
[16]	1.1M	SVM	Signature Base	C	–	–	–	0.986
[17]	15 036	ML-MPSO	PCA, Feat. Extr.	C	–	–	–	0.999
[18]	8510	CNN	Smali, Feat. Extr.	F	–	✓	Affine Shift	0.985
[19]	105 488	Ensemble CNN	TCP Packets Capture	F	✓	–	Custom	0.970
[20]	4294	CNN	Memory Dump	F	–	✓	Custom Augmentation	0.950
					–	–	–	**0.944**
					✓	–	**Average**	**0.873**
Our Method	**43865**	**CNN**	**DEX Code**	**F and C**	✓	–	**Clipping**	**0.877**
					–	✓	**Average**	**0.865**
					–	✓	**Clipping**	**0.864**

NS: Number of Samples, **F:** Federated Training, **C:** Centralized Training, **I:** Independent and identically distributed **NI:** non-Independent and identically distributed, **DR:** Detection Rate.

2.2. Federated machine learning for malware detection

In [22], authors investigate a comprehensive review of Edge Machine Learning (Edge ML), which combines Edge Computing and Artificial Intelligence to bring intelligent solutions closer to users. It focuses on soft computing, exploring over twenty paradigms and techniques for efficient edge inference and learning. The review highlights the importance of adapting ML technologies to edge constraints while maintaining performance and discusses the potential impact of Edge ML across various sectors like healthcare, autonomous vehicles, and smart cities. The paper aims to provide a common understanding of Edge ML and guide future research directions.

With the promising review above, we acknowledge a comprehensive analysis of security vulnerabilities and threats in edge computing by [23], emphasizing the need for a holistic approach to securing the edge network infrastructure. The paper discusses the challenges of architecture, from hardware to system layers, and explores the implications for privacy and regulatory compliance. The authors advocate for a multi-dimensional security analysis that considers the interaction between different domains to address emerging security issues effectively. The core idea is that understanding the unique characteristics of edge computing is crucial for developing robust security solutions. This suggests that solely focusing on images may be limiting given the vast array of features that applications possess, but with the use of models built strategically for use in low-edge devices, interesting results can be achieved using them.

Authors in [18] explore FML as a privacy-preserving cyber threat detection solution. The study evaluates FML's effectiveness, resilience to adversarial attacks, and efficiency through experiments with SMS spam and Android malware detection tasks. Results show FML-trained models perform comparably to central models, are resistant to poisoning attacks, and can be optimized for efficiency with strategies like bootstrapping despite non-IID data distribution challenges. In our work, we have addressed mainly the threat protection through the use of differential privacy aggregation methods and the challenges posed by non-IID data distribution, but it can be interesting to experiment with models that can hold malware detection capabilities together with spam detection and other such battery draining, permitting the training of a model in a horizontal way and aggregating once for all these, and maybe others, tasks as well.

With [20], the technique of collaborative training has been developed to be more robust to non-IID datasets and intermittent clients. The main proposal orbits around the concept of data augmentation, but from the client's perspective. Since some clients could have skewed datasets or be missing some labels, the local updates sent to the central model can be poor, diverging from the generalization goal or even deteriorating its capabilities. Various techniques have been proposed to avoid this, such as update selection, clipping, zeroing, or proximity terms. However, in this way, the updates are deleted, filtered, or regularized, thereby limiting the training phase. For this reason, the authors solved the problem by improving the local updates from the source. This work can be helpful in various areas where clients hold significant but scarce data; in our case, the dataset is important, so we did not require data augmentation techniques.

Banking malware remains a significant threat as cybercriminals exploit vulnerabilities to steal sensitive user information. Developing a privacy-preserving and effective solution for detecting banking malware is challenging. The paper [19] proposes an FML-based banking malware detection system that uses network traffic flow to address this issue. The study tackles challenges such as data heterogeneity in the FML scheme while maintaining the robustness of the global model. Three distinct heterogeneous datasets, each consisting of benign and prevalent malicious flows (Zeus, Emotet, or Trickbot), are used to address data heterogeneity. To ensure robustness, we initially evaluate multiple models and choose a Convolutional Neural Network (CNN) to build the ensemble model. FML is then incorporated to maintain data confidentiality and privacy, with the ensemble model serving as the global model. To improve the FML scheme, conditional updates of client models are introduced, and the evaluation results demonstrate the model's effectiveness, achieving a high detection rates of 0.993.

The authors in [14] propose a privacy-preserving Federated Machine Learning system for Android malware detection, ensuring that raw training data remains unknown to the server. This article introduces privacy-preserving in the malware detection field leveraging FML. Additionally, the system architecture uses edge computing for lower latency and decentralized data analytics in 5G networks. The method utilizes unsupervised clusterization performed by LDA (Latent Dirichlet Allocation) to extract the number of topics, which is then used as the number of clusters for identification by K-means. Moreover, they introduce a secret sharing aggregator, enabling collaborators to share updates without the assistance of a centralized server, which only receives the final trained model. This ensures privacy from the client's perspective while maintaining performance. In our manuscript, we employed two different datasets to perform the pretraining. Moreover, after training the model using the MobileNetV3 architecture on a dataset of Android malware and trusted applications, we compared performance using and not using the clipping norm aggregator.z

In [24], FEDKIT is presented as a Federated Machine Learning system designed for cross-platform research on Android and iOS devices. It simplifies development by enabling model conversion, hardware-accelerated training, and cross-platform model aggregation. The system supports flexible machine learning operations in production, allowing for continuous model delivery and training. FEDKIT has been effectively deployed in a real-world health data analysis project on university campuses, proving its practicality. The open-source system facilities further research and development in Federated Machine

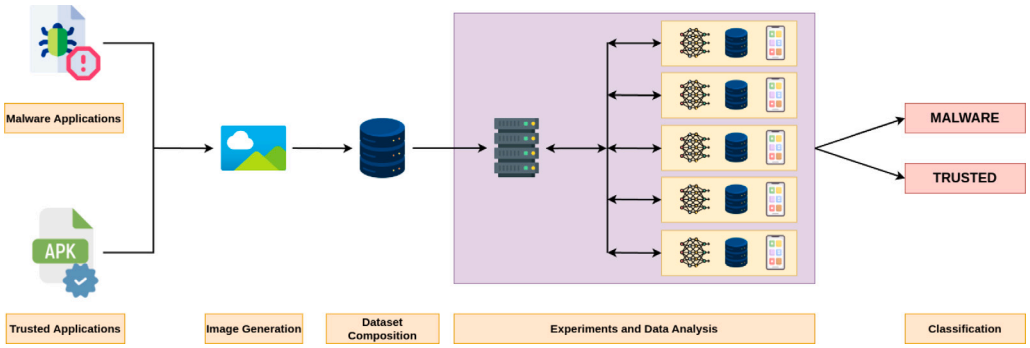


Fig. 1. The workflow of the proposed privacy-preserving Android malware detector.

Learning. This last work can give a glimpse of what a real environment collaborative learning approach is.

Wrapping up the comparison with related work, our method complies with GDPR, respecting user privacy, unlike many non-federated approaches. Furthermore, our model has demonstrated prominent capabilities in the spirit of collaborative and private reciprocal respect. This was shown in a simulation that resembles real-world divergences in data distribution and the computational capabilities of the current Android market; we can compare our model size using the memory occupied by the size of the weights. Indeed, our model has a size of 25KB; instead, the architecture of [19] occupied a total size of 261MB. This is possible because we utilized open-source datasets for pretraining and an extensive and diverse dataset for finetuning, ensuring feature extraction and generalization capabilities with a partially frozen model, the MobileNetV3.

3. The method

This section presents our proposed method to classify malware and trusted applications in the Android environment, leveraging FML for privacy-preserving mobile malware detection. For this task, we obtain a large dataset of images by converting malicious and trusted APKs, and for the classification task, we adopted both a federated and a centralized approach. Fig. 1 summarizes all the steps belonging to the proposed method.

3.1. Image generation

The APK (Android Package) is the file extension commonly used to indicate applications executable on an Android-based operating system (OS). An Android Package can be considered a compressed archive containing several files that the Android OS requires to execute an application. In the APK are stored files like AndroidManifest, META-INF, Assets, and one or more dex (Dalvik Executable) files. The Dalvik Executable file is one of the most significant files because it contains the compiled bytecode of the application's Java or Kotlin classes.

The usage of the images is widely adopted in the malware detection field as they allow visual inspection, code analysis [25], behavioral analysis [26], and detection of phishing attacks [18] and steganography [27]. Over the years, researchers have employed different methods to obtain images from APK files. Generally, experts can use system calls, where a specific color is associated with a precise event performed by the operating system [28]. A similar technique can be employed using Opcode (Operation Code). In this scenario, a color value is associated with each machine language instruction [29]. In this paper, we employed APK Tool[4] an open-source utility for reverse engineering Android APK files. Thanks to this tool, we decompiled a set of applications and extracted the Dex file. From the latter, we applied a script developed by the authors to convert the bytecode into a grayscale image. Fig. 2 reports two samples of images retrieved from malware and trusted applications. In detail, Fig. 2(a) illustrates a sample identified as trusted, characterized by the MD5 hash 831bf91d156985745f385ceb0dc7aa49, while Fig. 2(b) showcases a sample from the SpyBubble family, recognized by the MD5 hash 6b07336f56c3f967f235199356bd73e1.

3.2. Dataset composition

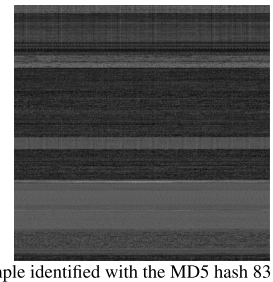
One of the best practices to obtain an effective model able to exhibit good performances in image classification is to create a robust and heterogeneous dataset composed of a collection of data that includes information from different sources or exhibits a variety of formats, structures, or characteristics. For this reason, we exploited an extensive dataset consisting of more than 40,000 samples, meticulously balanced between the two categories, *i.e.* malware and trusted. The malware dataset, composed of 71 families, was used to ensure comprehensive coverage and diversity during the training phase. To retrieve malware samples, we employed a large portion of the Android Malware Dataset (AMD) dataset, which represents one of the most used datasets for malware detection in the Android environment. Even though it comprises 24,553 APK files belonging to malware collected between 2010 and 2016, we only considered 23,431 samples to balance classes. The other class, *i.e.,* Trusted, comprises the same number of APK files retrieved from Google Play.[5] In detail, we employed a crawler developed by the authors to download APK files directly from the original store.

3.3. Experiments and data analysis

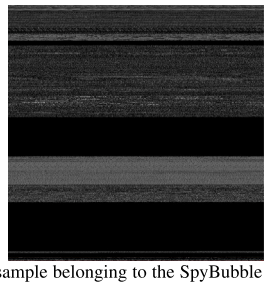
Once we concluded the dataset composition phase, we trained and tested several models leveraging Federated Machine Learning as the next step. About the architectures to employ, we decided to use two different custom versions of MobileNetV3 and the original InceptionV3. In detail, the original version of MobileNetV3 released in [30], while the InceptionV3 in 2016 [31]. Regarding the first is the last version provided. Before the latter, the same authors introduced other versions such as MobileNet [32] and MobileNetV2 [33]. We exploit this specific architecture considering that MobileNetV3 is a CNN tuned on mobile phone CPUs using a blend of hardware-aware network architecture search (NAS) enhanced by the NetAdapt algorithm, followed by refinement via innovative architectural advancements. One of the improvements introduced by the authors is related to a new efficient network design. In detail, they presented the complex swish activation function, an evolution of the Swish activation that uses clipped ReLU (ReLU6) instead of the classical sigmoid. This modification makes the hard swish computationally cheaper, involving simple thresholding operations. Regarding their customization, we employed

[4] https://apktool.org/

[5] https://play.google.com/



(a) Trusted sample identified with the MD5 hash 831bf91d156985745f385ceb0dc7aa49.



(b) A malware sample belonging to the SpyBubble family identified with the MD5 hash 6b07336f56c3f967f235199356bd73e1.

Fig. 2. Examples of trusted and malware images extracted from APK files.

a similar structure for both CNNs that were introduced. In detail, they present additional fully connected layers, dropout regularization, and flexible weight initialization. The main differences between the first and second custom versions rely on using the dropout layers before the final dense layer and softmax activation to improve regularization. Listing 1 shows the structure of the Custom MobileNet 1 without green rows. On the other hand, the Custom MobileNet 2 is represented using the same structure of the previous architecture with the addition of green rows *i.e.,* dropout layers. Regarding the InceptionV3 architecture, it was designed to achieve high accuracy on image classification tasks. Moreover, this convolutional neural network employs Inception modules, which capture multi-scale features through parallel convolutions of different sizes, enhancing feature extraction. Additionally, auxiliary classifiers improve gradient flow during training, mitigating the vanishing gradient problem. By replacing fully connected layers with global average pooling (GAP), InceptionV3 reduces the number of parameters and minimizes overfitting.

In FML, another important aspect of achieving good results in performance and convergence of the model depends on the distribution. From the distribution point of view in literature, there are several approaches, including the Independent and Identically Distributed (IID) less with Non-Independent and Identically Distributed (Non-IID). In detail, in the first case, the data across all clients are independently sampled from the same distribution and have identical statistical properties. This type of distribution represents a "simple" task for machine learning algorithms during simulation, as the model will train the layer's weights homogeneously with data that is conformally representative of the entire dataset.

However, in real-world scenarios, obtaining a perfect IID data distribution is not simple due to the heterogeneity of the data and the entropy of the events that contribute to their random formation. Due to this, most of the methods proposed in the literature using IID distributions are unrealistic. In the Non-IID distribution, the data distribution across clients is not identical. This means that each client's data comes from a different distribution and has different statistical properties, better representing the diversity of the dataset held by the collaborators in the federated approach [34].

To understand the differences in terms of results between these two types of distribution, in this paper, we employed both of them, *i.e.* Default distribution (IID) and Dirichlet distribution (Non-IID). Fig. 3 reports the distribution among 40 clients. We highlight that by exploiting this distribution, we recreated an almost balanced scenario where the number of samples of both classes was about the same. From the other side, the Non-IID distribution is shown in Fig. 4. In the latter, we recreated a real-world scenario where the number of samples is unbalanced and follows the skews that we would find in each other personal devices, such as quantity, quality, and source of data.

Once we identified the architecture to employ and two types of distribution, we adopted the Grid Search technique to determine the best hyperparameters to find the top combination that optimizes the performance of a Federated Machine Learning model. In detail, we employed the CIFAR-10 dataset[6] and the ImageNet dataset[7] as a preparatory phase. The first consists of 60,000 32×32 color images divided into 50,000 for the training phase and 10,000 for testing, belonging to the ImageNet dataset, *i.e.* one of the most extensive datasets employed for image recognition and object detection. In detail, the latter comprises 14,197,122 annotated images according to the WordNet hierarchy. Using these datasets during the preparatory phase permits the creation of initial image features within the hidden layers of the model. This approach reduces variance by setting a common starting point for the experiments and decreases computational costs, as we will freeze the pre-trained section of the model. This section will function as an embedding model, generating vector representations of the images. These representations will be trained using the federated method to classify the applications. After finishing the pre-training phase and recognizing the best hyperparameters, we started the training execution on the dataset built by performing several experiments. Tables 2 and 3 report the best hyperparameters for each distribution *i.e.,* IID and non-IID data. We examined the difference between applying and not applying the clipping norm for each distribution and dataset employed during the preparatory phase. Adopting this technique limits the impact of outliers by capping the gradients above a certain threshold, which was found using a method that ensures sensitive information privacy based on the magnitude of the clients' gradients. In a nutshell, it helps manage uneven contributions from various client devices [35]. Generally, the clipping norm mitigates the effects of extreme model updates from individual clients. This approach involves clipping the gradients by each client to a predefined threshold before averaging them at the server. By limiting the magnitude of individual updates, clipping the gradients helps prevent any single client from disproportionately influencing the global model. However, in this case, Differential Private Quantile Clipping has been adopted to incorporate differential privacy mechanisms into the aggregation process, ensuring that the adapted quantile-clipped value computed on the gradients and used as a threshold to aggregate the model update does not reveal sensitive information about any individual client's data. By adding carefully calibrated Gaussian noise to the private clipping threshold, this aggregator protects against privacy breaches while boosting model training across distributed data sources. Using this norm ensures that the gradient aggregation process is protected from peak updates that could add noise to the training process. Not applying the aggregator at all includes the risk of outliers and overtraining, but can ensure variability and completeness of training on the dataset.

As shown in Tables 2 and 3, we considered scientific notation to express the precision of measurements about the Minimum Learning Rate and Learning Rate. First, for each distribution, we executed eight runs where four were pre-trained using the CIFAR-10 dataset, while the rest used the ImageNet dataset. The models presented in this manuscript were also trained using similar hyperparameters for all architectures. In

[6] https://www.cs.toronto.edu/~kriz/cifar.html

[7] https://www.image-net.org/

Listing 1 Custom MobileNetV3 model with additional Dropout and Fully Connected Layers.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Input(shape=img_shape),
  MobileNetV3,
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(ff_neurons, activation="silu", name="ff_Dense",
                        \\ kernel_initializer=initializer),
  tf.keras.layers.Dropout(dropout),
  tf.keras.layers.Dense(num_classes, name=name + "_Output",
                        \\ kernel_initializer=initializer),
  tf.keras.layers.Dropout(dropout),
  tf.keras.layers.Softmax(name=name + "Softmax")
])
```

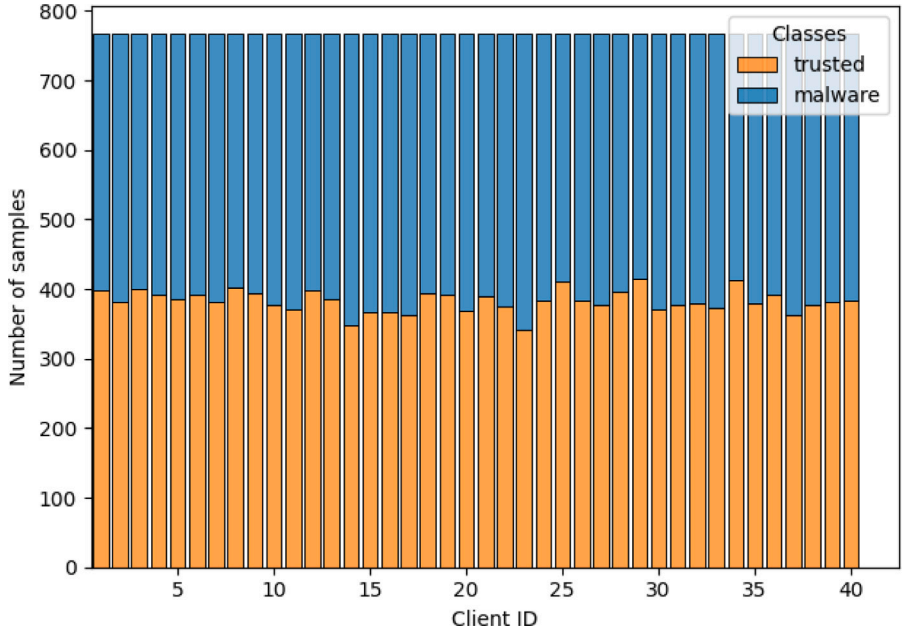


Fig. 3. The Default distribution obtained among 40 different clients.

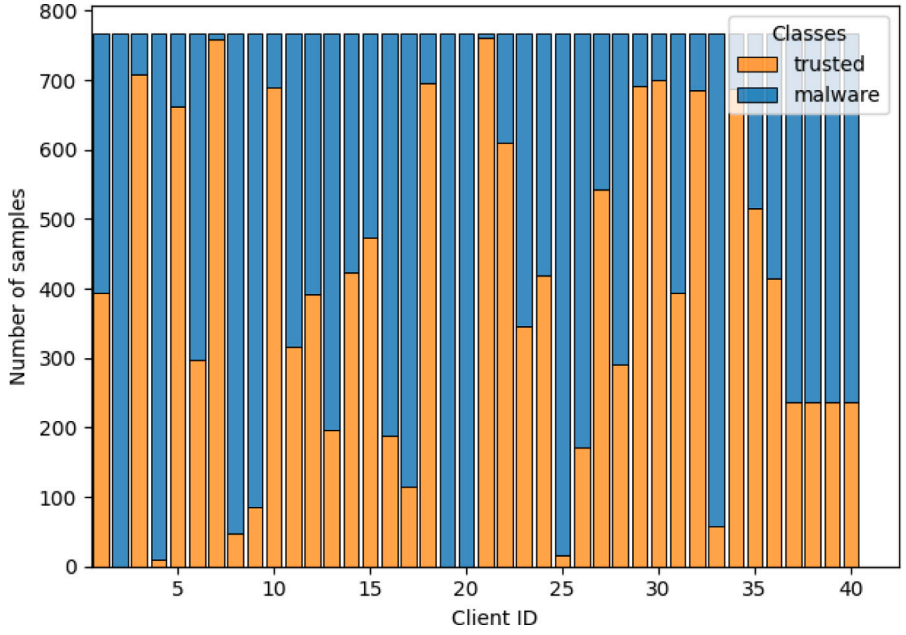


Fig. 4. The Dirichlet distribution obtained among 40 different clients.

Table 2
Hyperparameters adopted to train models using IID data.

Model	Run	Image Size	TC	CR	TR	CER	LR	DIST	IDE	SD	NM	NL2	NQ	NLR	NI	NT	Weights
Custom MobileNet 1	1	160 × 3	40	36	20	3	8e-4	IID	–	–	–	–	–	–	–	–	CIFAR-10
	2	160 × 3	40	36	20	3	8e-4	IID	–	0.2	0.1	16	0.75	0.01	2	Clipping	CIFAR-10
	3	160 × 3	40	36	20	3	8e-4	IID	–	–	–	–	–	–	–	–	Imagenet
	4	160 × 3	40	36	20	3	8e-4	IID	–	0.2	0.1	16	0.75	0.01	2	Clipping	Imagenet
Custom MobileNet 2	1	160 × 3	40	36	20	3	8e-4	IID	–	–	–	–	–	–	–	–	CIFAR-10
	2	160 × 3	40	36	20	3	8e-4	IID	–	0.2	0.1	16	0.75	0.01	2	Clipping	CIFAR-10
	3	160 × 3	40	36	20	3	8e-4	IID	–	–	–	–	–	–	–	–	Imagenet
	4	160 × 3	40	36	20	3	8e-4	IID	–	0.2	0.1	16	0.75	0.01	2	Clipping	Imagenet
InceptionV3	1	160 × 3	40	36	20	5	1e-4	IID	–	–	–	–	–	–	–	–	CIFAR-10
	2	160 × 3	40	36	20	5	1e-4	IID	–	0.2	0.1	16	0.75	0.01	2	Clipping	CIFAR-10
	3	160 × 3	40	36	20	5	1e-4	IID	–	–	–	–	–	–	–	–	Imagenet
	4	160 × 3	40	36	20	5	1e-4	IID	–	0.2	0.1	16	0.75	0.01	2	Clipping	Imagenet

TC: Total Client; **CR:** Client per Round; **TR:** Total Round; **CER:** Client Epochs per Round; **LR:** Learning Rate; **DIST:** Distribution; **IDE:** Identicalness; **SD:** Standard Deviation; **NM:** Noise Multiplier; **NL2:** Norm L2; **NQ:** Norm Quantile; **NLR:** Norm LR; **NI:** Norm Increment; **NT:** Norm Type.

Table 3
Hyperparameters adopted to train models using non-IID data.

Model	Run	Image size	TC	CR	TR	CER	LR	DIST	IDE	SD	NM	NL2	NQ	NLR	NI	NT	Weights
Custom MobileNet 1	1	160 × 3	40	36	20	3	8e-4	non-IID	0.8	–	–	–	–	–	–	–	CIFAR-10
	2	160 × 3	40	36	20	3	8e-4	non-IID	0.8	0.2	0.1	16	0.75	0.01	2	Clipping	CIFAR-10
	3	160 × 3	40	36	20	3	8e-4	non-IID	0.8	–	–	–	–	–	–	–	Imagenet
	4	160 × 3	40	36	20	3	8e-4	non-IID	0.8	0.2	0.1	16	0.75	0.01	2	Clipping	Imagenet
Custom MobileNet 2	1	160 × 3	40	36	20	3	8e-4	non-IID	0.8	–	–	–	–	–	–	–	CIFAR-10
	2	160 × 3	40	36	20	3	8e-4	non-IID	0.8	0.2	0.1	16	0.75	0.01	2	Clipping	CIFAR-10
	3	160 × 3	40	36	20	3	8e-4	non-IID	0.8	–	–	–	–	–	–	–	Imagenet
	4	160 × 3	40	36	20	3	8e-4	non-IID	0.8	0.2	0.1	16	0.75	0.01	2	Clipping	Imagenet
InceptionV3	1	160 × 3	40	36	20	5	1e-4	non-IID	0.8	–	–	–	–	–	–	–	CIFAR-10
	2	160 × 3	40	36	20	5	1e-4	non-IID	0.8	0.2	0.1	16	0.75	0.01	2	Clipping	CIFAR-10
	3	160 × 3	40	36	20	5	1e-4	non-IID	0.8	–	–	–	–	–	–	–	Imagenet
	4	160 × 3	40	36	20	5	1e-4	non-IID	0.8	0.2	0.1	16	0.75	0.01	2	Clipping	Imagenet

TC: Total Client; **CR:** Client per Round; **TR:** Total Round; **CER:** Client Epochs per Round; **LR:** Learning Rate; **DIST:** Distribution; **IDE:** Identicalness; **SD:** Standard Deviation; **NM:** Noise Multiplier; **NL2:** Norm L2; **NQ:** Norm Quantile; **NLR:** Norm LR; **NI:** Norm Increment; **NT:** Norm Type.

detail, we employed the same image size (160 × 3), number of clients (40), clients per round (36), decay steps (3000), learning rate (8e-4), minimum learning rate (9.8e−7), momentum (0.9), server learning rate (0.99), server momentum (0.99), minimum learning rate (9.8e−7), momentum (0.9), server learning rate (0.99), server momentum (0.99), 0.2 as dropout, and 3000 decay steps. Regarding the first CNN, we specifically used 20 total rounds, 3 client epochs per round, and 8e-4 learning rate. On the other hand, using the InceptionV3 architecture, we employed 5 epochs per round and a learning rate of 1e-4. Furthermore, for Runs 2 and 4 for each distribution and for each CNN employed, we applied the clipping norm, which required additional hyperparameters like standard deviation (0.2), L2 norm clipping threshold (16) used to defined after which threshold the updates of the model are clipped, norm quartile (0.75) used to determine the clipping threshold based on a specific quantile of the gradient norm distribution. Moreover, we also applied a norm learning rate (0.01) to control how quickly the clipping threshold or related parameters might be adjusted during training, and a norm increment (2) to increase or change the clipping threshold during training. To provide further protection, we added privacy noise using 0.01 as the noise multiplier and 0.2 as the standard deviation of the noise. The first parameter controls the magnitude of the noise added to the gradients, while the second one controls the amount of noise that is added to each update. The previous values were obtained through GridSearch algorithms, which systematically explore a range of hyperparameter combinations to identify the optimal configuration. In conclusion, we set the identicalness (α) parameter to 0.8 in the Dirichlet distribution. This parameter customizes the heterogeneity of the sample distribution among clients, reducing the probability of assigning certain labels. However, the property of probabilities summing to 1 still holds, so that we will have a peak in a label probability. In the end, each label probability assigns the quantity of data relative to this probability, resulting in a more skewed distribution when the identicalness is close to 0.

Table 4
Hyperparameters adopted to train models leveraging centralized approach.

Model	Epochs	Batch size	Learning rate	Image size
Inception	30	32	3e-5	160 × 3
Inception	25	64	3e-5	160 × 3
Custom MobileNet 1	20	64	3e-5	160 × 3
Custom MobileNet 1	30	32	3e-5	160 × 3
Custom MobileNet 2	20	64	3e-5	160 × 3
Custom MobileNet 2	25	32	3e-5	160 × 3

After defining the hyperparameters to train multiple models with Federated Machine Learning under different data distributions, both with and without a clipping norm aggregator, we conducted an additional grid search to identify the optimal hyperparameters for the centralized approach. In detail, we used a range of epochs between 20 and 30, a batch size of 32 and 64, and a range of learning rate values between 8e-4 and 3e-5. The best hyperparameters that allowed us to achieve the best performances are reported in Table 4.

4. Results

In this Section, we report and discuss the experimental analysis results. In detail, we compared the models obtained using Federated Machine Learning with models trained using a centralized approach, in terms of accuracy and privacy. Regarding Federated Machine Learning, in Section 4.1, we report the results obtained by training multiple models using IID and non-IID data, pre-trained weights from the CIFAR-10 and ImageNet datasets, with and without applying a clipping norm aggregator. On the other hand, Section 4.2 displays the outcomes obtained using the same dataset division and Convolutional Neural Networks in a centralized scenario.

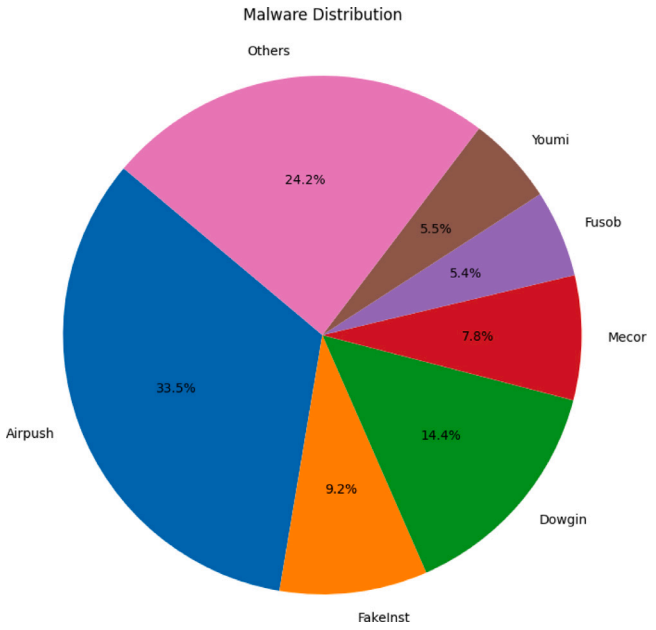


Fig. 5. Distribution of the most popular malware families in the analyzed dataset.

As discussed in Section 3, before the execution of the experiment, we need to build a dataset of Android applications. For this purpose, we downloaded 46,862 *.apk* files, respectively 23,431 trusted and 23,431 malware. To comply with a real-world scenario and to ensure comprehensive coverage, we consider malware belonging to 71 different malware families to cover the current mobile malware landscape. The families are: *Airpush (7494), AndroRAT (43), Andup (43), Aples (21), BankBot (617), Bankun (68), Boqx (207), Boxer (42), Cova (17), Dowgin (3222), DroidKungFu (517), Erop (45), FakeAngry (10), FakeDoc (20), FakeInst (2063), FakePlayer (19), FakeTimer (12), FakeUpdates (5), Fjcon (15), Fobus (4), Fusob (1215), GingerMaster (121), GoldDream (50), Gorpo (35), Gumen (141), Jisut (543), Kemoge (15), Koler (66), Ksapp (33), Kuguo (1139), Kyview (167), Leech (119), Lotoor (313), Mecor (1739), Minimob (196), Mmarketpay (12), MobileTX (17), Mseg (228), Mtk (64), Nandrobox (73), Ogel (6), Penetho (17), Ramnit (8), Roop (46), RuMMS (383), SimpleLocker (168), SlemBunk (169), SmsKey (157), Spambot (15), Stealer (22), Steek (11), Svpeng (11), Triada (207), UpdtKiller (23), Utchi (12), Vidro (22), Vmvol (12), Winge (18), Youmi (1239), Zitmo (24), Ztorg (20), FakeAV (5), Finspy (9), Lnk (5), Obad (7), Opfake (9), SmsZombie (8), SpyBubble (8), Tesbo (5), Univert (10),* and *VikingHorde (5)*. The distribution of the malware families is shown in Fig. 5. As it is possible to denote, the majority of the dataset is composed of Airpush family samples (33.5%), Downgin with 14.4%, FakeInst 9.2%, Mecor 7.8%, Youmi 5.5%, Fusob 5.4%, and other 69 families with a less percentage. Once downloaded *.apk* files, we submitted them to Virustotal[8] and the Jotti[9] web services to verify whether the samples we obtained are malicious or trustworthy. After obtaining the Android applications, we executed a Python script created by the authors to convert the *.dex* file to an image. Following this step, we obtained a large dataset segmented into training, validation, and testing (70-20-10). Before starting our experiments, we conducted two additional preliminary steps: pre-training (on the CIFAR-10 and ImageNet datasets) and Grid Search execution. In detail, we performed pre-training phases on these popular datasets to provide the model a starting model with useful general characteristics. Moreover, we executed the Grid Search algorithm which systematically explores all possible combinations of parameters to optimize a machine-learning model and identify the best hyperparameter combination. Identified the latter, we started to train our model using different runs. In detail, in Tables 2 and 3 are reported eight runs where for each distribution, each architecture and each set of weights obtained from the pre-training, we also introduced the Clipping Norm aggregator to evaluate the global performance of the model following the integration of contributions from different client devices. Also the latter employs differential privacy techniques to ensure that information related to the clients' dataset, such as the magnitude or size of the updates, remains hidden from potential attackers. During the training phase, we also computed the validation every five rounds to ensure the model's progress is tracked efficiently without excessive computational overhead.

[8] https://www.virustotal.com/
[9] https://virusscan.jotti.org/

Table 5
Results obtained after training, validation, and test set execution employing IID data (Default Distribution).

Default Distribution							
Run	Model	TRA	VA	TA	TRL	VL	TL
1	Custom MobileNet 1	0.800	0.761	0.744	0.481	0.545	0.572
2	Custom MobileNet 1	0.793	0.784	0.787	0.562	0.594	0.599
3	Custom MobileNet 1	0.831	0.817	0.816	0.538	0.672	0.646
4	Custom MobileNet 1	0.815	0.805	0.805	0.497	0.535	0.527
1	Custom Mobilenet 2	0.837	0.867	0.873	0.457	0.378	0.341
2	Custom Mobilenet 2	0.836	0.872	0.877	0.459	0.368	0.338
3	Custom Mobilenet 2	0.836	0.853	0.860	0.458	0.417	0.378
4	Custom Mobilenet 2	0.834	0.863	0.870	0.460	0.386	0.347
1	InceptionV3	0.833	0.831	0.831	0.546	0.550	0.562
2	InceptionV3	0.833	0.831	0.831	0.546	0.550	0.562
3	InceptionV3	0.873	0.871	0.873	1.983	30.816	31.818
4	InceptionV3	0.873	0.871	0.873	1.983	30.816	31.818

TRA: Training Accuracy **VA:** Validation Accuracy **TA:** Test Accuracy
TRL: Training Loss **VL:** Validation Accuracy **TL:** Test Accuracy.

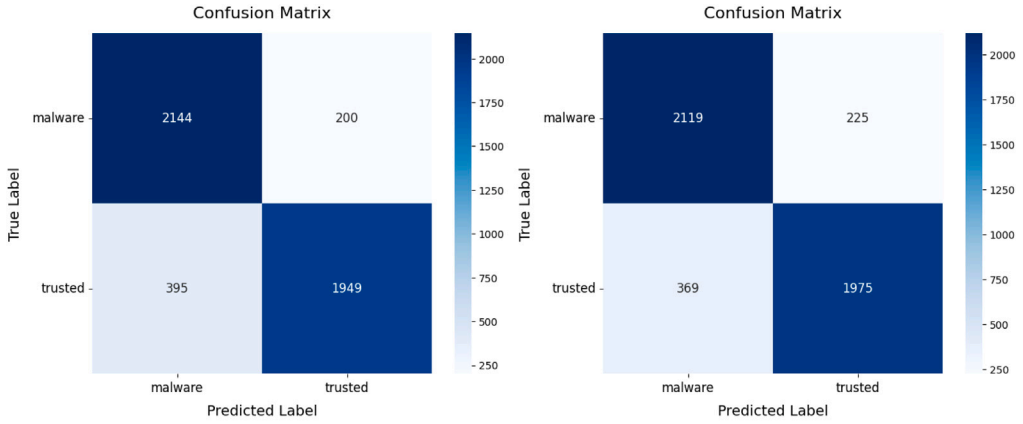
4.1. Federated machine learning approach

Before starting our experiments using Federated Machine Learning approach, we conducted two additional preliminary steps: pre-training (on the CIFAR-10 and ImageNet datasets) and Grid Search execution. In detail, we performed pre-training phases on these popular datasets to provide the model a starting model with useful general characteristics. Moreover, we executed the Grid Search algorithm which, systematically explores all possible combinations of parameters to optimize a machine-learning model and identify the best hyperparameter combination. Once we identified the latter, we started to train our model using different runs. In detail, in Tables 2 and 3 are reported eight runs where for, each architecture, each distribution and each set of weights obtained from the pre-training, we also introduced the Clipping Norm aggregator to evaluate the global performance of the model following the integration of contributions from different client devices. Also, the latter employs differential privacy techniques to ensure that information related to the clients' dataset, such as the magnitude or size of the updates, remains hidden from potential attackers. During the training phase, we also computed the validation every five rounds to ensure the model's progress is tracked efficiently without excessive computational overhead.

4.1.1. Default distribution

This Section describes the results obtained by employing an IID data *i.e.,* Dafault distribution. Table 5 reports all results obtained during the training, validation, and test phases. In detail, about the outcome of training and validation, only the values obtained at the last round *i.e.,* 20th are shown.

No Norm Aggregator Application: Using an IID data without applying any aggregation norm, we performed several experiments using



(a) Confusion matrix obtained from the model trained using Custom MobileNet 2.

(b) Confusion matrix obtained from the model trained using InceptionV3.

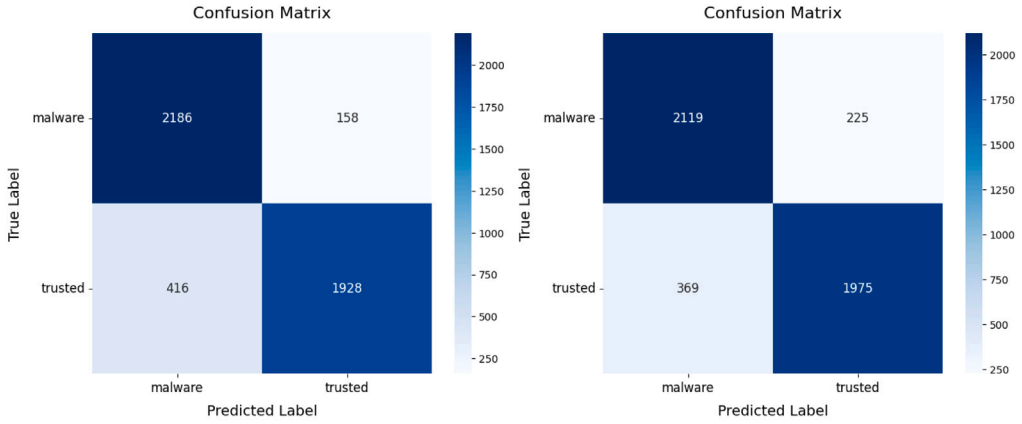
Fig. 6. Confusion matrices obtained after the test phase for the top-performing models: Custom MobileNet 2 (trained with CIFAR-10 weights) and Inception V3 (trained with ImageNet weights), evaluated without normalization and on IID data.

three different architectures, Custom MobileNet 1, Custom Mobilenet 2, and InceptionV3, obtaining interesting results for each. In detail, using Custom Mobilenet 1, we received an accuracy value of 0.744 using a model pre-trained on the CIFAR-10 dataset and 0.816 using a pre-trained model on the ImageNet dataset. On the other hand, with Custom MobileNet 2, the accuracy values increased for both weights employed. Specifically, using CIFAR-10 weights, we achieved 0.873 accuracy, while 0.860 using Imagenet weights. In conclusion, using the InceptionV3 architecture, we achieved an accuracy of 0.831 on CIFAR-10 and 0.873 using pre-trained ImageNet weights. However, despite the higher accuracy in the latter case, the loss values increased significantly, rising from 1.983 in the final training round to 31.818 during the test phase. This drastic rise in loss suggests potential overfitting or poor generalization.

After concluding the training and validation phases, we also performed a test calculating the confusion matrix. Fig. 6 reports the confusion matrices retrieved from the models that obtained higher values in terms of accuracy *i.e.,* Custom MobileNet 2 and InceptionV3. In detail, Fig. 6(a) shows the confusion matrix referred to the model trained using Custom MobileNet 2, where the model correctly classifies 2144 samples as malware and 1949 samples as trusted. However, it misclassifies 200 malware samples as trusted and 395 trusted samples as malware. Regarding the performances achieved for each class during the test phase, the model achieved similar results in terms of loss (0.339), accuracy (0.873), and AUC (0.873). Moreover, malware samples were classified with a precision of 0.844, while trusted samples achieved a precision of 0.906. In terms of recall, malware reached 0.914, whereas trusted samples had a recall of 0.831. On the other hand, the confusion matrix obtained from the model trained on InceptionV3 is displayed in Fig. 6(b). Despite the higher accuracy value (31.818), the model classifies most samples correctly. In detail, the model correctly classified 2119 samples as malware and 1975 trusted samples. On the other hand, it misclassifies 225 malware samples, recognizing them as trusted, and 269 trusted samples are recognized as malware. Concerning the performance of each class during the testing phase, the performance metrics for both classes are quite comparable, with a loss of 2.002, an accuracy of 0.873, and an AUC of 0.873. However, malware samples exhibited lower precision (0.851) than trusted samples (0.897). In contrast, the Recall and F-measure values were higher for malware samples, at 0.904 and 0.877, respectively, surpassing the trusted samples, which had values of 0.842 and 0.869. A key observation is the reduction in the loss value, dropping from around 31.818 for the entire model to 2.002 for each class. This difference likely arises because the model's overall loss is calculated across all predictions, while the per-class loss may be computed differently, possibly averaged for each class rather than over the entire dataset.

Clipping Norm Aggregator Application: On the IID data, we also adopted the Clipping Norm aggregator. The latter consists of enhanced robustness and privacy during the model aggregation process. Using the Custom MobileNet 1 to train a model with CIFAR-10 weights, we reached an accuracy value at the end of the test phase of 0.787 and a loss of 0.599. On the other hand, using Imagenet weights, we achieved higher results in terms of accuracy and loss, 0.805 and 0.527, respectively. Notable results were also obtained with the Custom MobileNet 2 architecture, achieving an accuracy of 0.877 and a loss of 0.338 when using CIFAR-10 weights and an accuracy of 0.870 with a loss of 0.347 when using ImageNet weights. Regarding the models trained using InceptionV3, the results did not change from the previous presented without using a norm aggregator application. In detail, we obtained an accuracy value of 0.831 on CIFAR-10 and 0.873 using pre-trained ImageNet weights.

As the next step of the training and validation phases, we also performed the test, obtaining confusion matrices. We employed them to evaluate the performance of the models by showing the number of correct and incorrect predictions for each class. In Fig. 7, we reported the confusion matrices obtained by the best two models *i.e.,* Custom MobileNet 2 trained using CIFAR-10 weights and InceptionV3 trained using Imagenet weights. In detail, Fig. 7(a) reports the confusion matrix obtained from Custom MobileNet 2, which correctly classified 2186 malware and 1928 trusted samples, while 158 cases identified malware samples as trusted and 416 trusted samples as malware. Regarding the performances obtained for each class, the model achieved an overall accuracy and AUC of 0.878, with a balanced loss of 0.336 across both classes. While malware samples had a higher recall (0.933) and F-measure (0.884), trusted samples demonstrated better precision (0.924). The metrics obtained suggest that the model is more effective at identifying malware but slightly better at precisely classifying trusted samples. On the other hand, Fig. 7(b) reports the confusion matrix obtained after the test on the InceptionV3 architecture using ImageNet weights. In this case, the model correctly identified 2119 malware samples while misclassifying 225. It accurately recognized 1975 for trusted samples but mistakenly classified 369 as malware. The model trained using the InceptionV3 architecture using MobileNet weights achieved an overall accuracy and AUC of 0.873, with a loss of 2.002 for both classes. Malware samples had higher recall (0.904) and F-measure (0.877), indicating strong detection capability, while trusted samples showed better precision (0.898), suggesting fewer false positives. This balance highlights the model's effectiveness in identifying malware while maintaining reliable classification of trusted samples.



(a) Confusion matrix obtained from the model trained using Custom MobileNet 2.

(b) Confusion matrix obtained from the model trained using InceptionV3.

Fig. 7. Confusion matrices obtained after the test phase for the top-performing models: Custom MobileNet 2 (trained with CIFAR-10 weights) and Inception V3 (trained with ImageNet weights), evaluated using Clipping Norm Aggregator and on IID data.

Table 6
Results obtained after training, validation, and test set execution employing non-IID data (Dirichlet Distribution).

Dirichlet Distribution							
Run	Model	TRA	VA	TA	TRL	VL	TL
1	Custom MobileNet 1	0.810	0.738	0.731	0.458	0.561	0.572
2	Custom MobileNet 1	0.810	0.744	0.737	0.478	0.578	0.577
3	Custom MobileNet 1	0.817	0.773	0.773	0.466	0.567	0.562
4	Custom MobileNet 1	0.816	0.728	0.744	0.490	0.687	0.671
1	Custom Mobilenet 2	0.858	0.868	0.865	0.502	0.464	0.471
2	Custom Mobilenet 2	0.858	0.868	0.864	0.496	0.467	0.473
3	Custom Mobilenet 2	0.859	0.870	0.864	0.490	0.462	0.468
4	Custom Mobilenet 2	0.856	0.866	0.864	0.506	0.496	0.499
1	InceptionV3	0.823	0.718	0.720	0.371	0.506	0.504
2	InceptionV3	0.827	0.721	0.722	0.369	0.504	0.502
3	InceptionV3	0.890	0.812	0.808	1.540	7.607	7.463
4	InceptionV3	0.890	0.812	0.808	1.540	7.607	7.463

TRA: Training Accuracy **VA:** Validation Accuracy **TA:** Test Accuracy
TRL: Training Loss **VL:** Validation Accuracy **TL:** Test Accuracy.

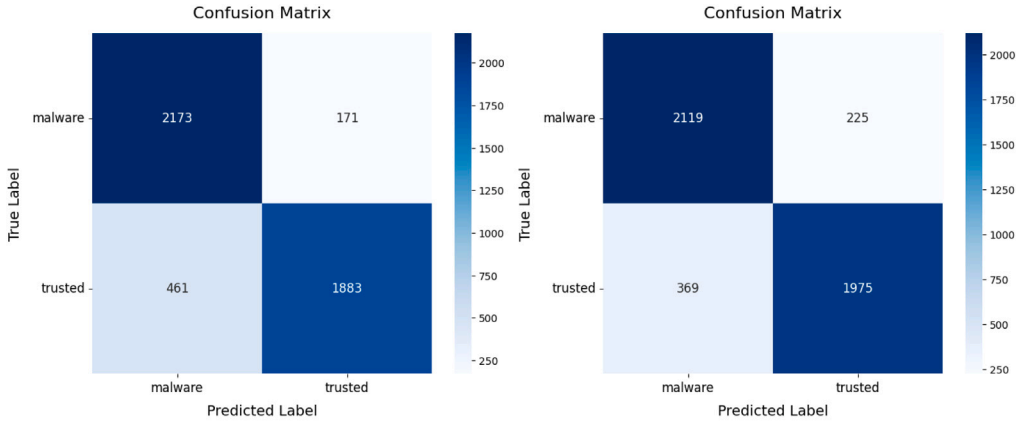
4.1.2. Dirichlet distribution

In this section, we discuss the outcomes obtained during the experiments by exploiting non-IID data. In detail, we employed the Dirichlet Distribution widely adopted in the Federated Machine Learning field because of the possibility of simulating data across different clients or devices. Table 6 reports the outcomes obtained during the last round of training and validation and the end of the test phase for each Run.

No Norm Aggregator Application: As the first step in non-IID data, we executed two runs without using any norm aggregator employing three different Convolutional Neural Networks *i.e.,* Custom MobileNet 1, Custom MobileNet 2, and Inception V3. In detail, in Run 1, we employed a pre-trained model on the CIFAR-10 dataset for each of the CNNs taken into account. Using the first architecture, we obtained an accuracy value of 0.810 in training and 0.738 in the validation process during the last round. After the previous phases, the model achieved an accuracy of 0.731 and a loss of 0.572 in the test phase. On the other hand, using Custom MobileNet 2, we achieved higher accuracy results. In detail, the model achieved an accuracy of 0.865 and a loss of 0.471 during the test. To conclude the experimentation phase, we also trained a model using the InceptionV3 in which, using CIFAR-10 weights, we achieved far lower accuracy than the other two networks. Specifically, the model obtained an accuracy value of 0.720 and a loss of 0.471 during the test phase. Once we concluded the training, validation, and test phases using weights from CIFAR-10, we executed experiments using the same hyperparameters using weights from Imagenet. The outcome obtained highlighted an improvement in terms of accuracy for two of the three architectures employed. In detail, using the Custom MobileNet 1, the accuracy value during the test phase increased to 0.773, decreasing the loss to 0.562. Similarly, the Inception V3 model achieved a higher accuracy of 0.808; however, the loss increased to 7.463. Regarding the Custom MobileNet 2, the results were slightly lower than those of the previous model, achieving an accuracy of 0.863 (a decrease of 0.001). However, it outperformed in terms of loss, achieving a lower value of 0.468, suggesting better overall model stability and fewer prediction errors.

At the end of the test phase, we also calculated the confusion matrices for the models that achieved the best accuracy. In detail, Fig. 8 reports the confusion matrices obtained from the Custom MobileNet 2 trained using CIFAR-10 weights and InceptionV3 trained using Imagenet weights. Regarding the first model, as shown in Fig. 8(a), Custom MobileNet 2 correctly classified 2173 malware and 1883 trusted samples. While 171 times committed misclassification errors identifying malware samples as trusted, 461 recognized trusted samples as malware. The model achieved interesting classification performances regarding loss, accuracy, and AUC, with both classes reaching 0.465, 0.865, and 0.865, respectively. Moreover, the precision value achieved in the trusted class is higher (0.916) than in the malware class (0.824). On the other hand, the model achieved higher recall and F-measure values in the malware class, with scores of 0.927 and 0.873, respectively. In comparison, the trusted class saw lower values, with recall and F-measure scores of 0.803 and 0.856, respectively. This indicates that the model performed more effectively in identifying malware samples, while the performance on trusted samples was slightly lower. The confusion matrix obtained from the model trained using InceptionV3 using Imagenet weights is reported in Fig. 8(b). The model successfully identified 1721 malware and 2071 trusted samples in this case. However, it misclassified 623 malware and 273 trusted samples. Regarding the performance metrics for each class, the loss, accuracy, and AUC values were identical across both classes, each achieving scores of 2502, 0.808, and 0.808, respectively. The model showed varying performance in precision, recall, and F-measure. The trusted class achieved better results with a precision of 0.768, a recall of 0.884, and an F-measure of 0.822. In comparison, the malware class achieved a precision of 0.863, a recall of 0.734, and an F-measure of 0.793.

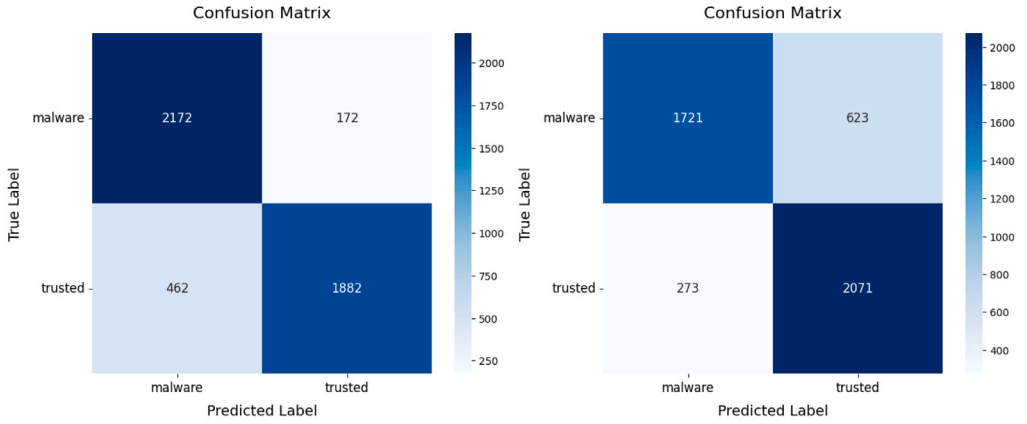
Clipping Norm Aggregator Application: In the last two experiments Run 2 and 4, we used the Clipping Norm aggregator applied on a non-IID data. Using that norm, the model trained using Custom MobileNet 1 reached an accuracy value of 0.737 and a loss of 0.572 using CIFAR-10 weights. On the other hand, using the Custom Mobilenet 2 *i.e.,* the architecture with two more Dropout layers, we achieved



(a) Confusion matrix obtained from the model trained using Custom MobileNet 2.

(b) Confusion matrix obtained from the model trained using InceptionV3.

Fig. 8. Confusion matrices obtained after the test phase for the top-performing models: Custom MobileNet 2 (trained with CIFAR-10 weights) and Inception V3 (trained with ImageNet weights), evaluated without normalization and on non-IID data.



(a) Confusion matrix obtained from the model trained using Custom MobileNet 2.

(b) Confusion matrix obtained from the model trained using InceptionV3.

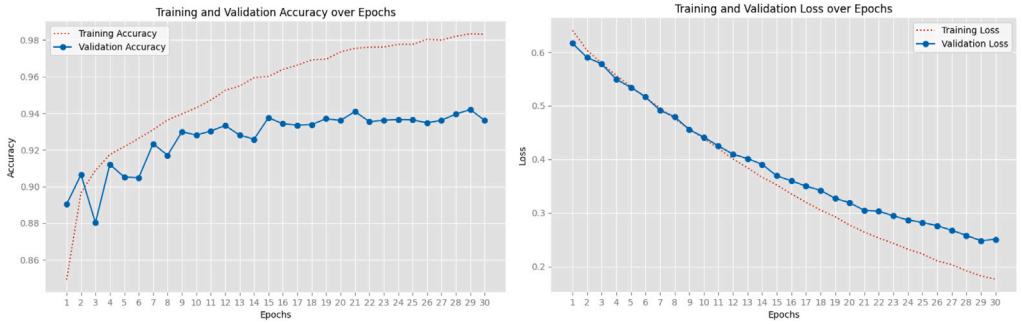
Fig. 9. Confusion matrices obtained after the test phase for the top-performing models: Custom MobileNet 2 (trained with CIFAR-10 weights) and Inception V3 (trained with ImageNet weights), evaluated using Clipping Norm Aggregator and on non-IID data.

higher results in terms of accuracy (0.864) and a lower loss (0.473). The InceptionV3 architecture yielded lower results, with the model achieving an accuracy of 0.722 and a loss of 0.504 when using CIFAR-10 weights. After completing the training, validation, and testing phases using CIFAR-10 weights, we conducted additional experiments using the same hyperparameters but with ImageNet weights. This approach improved performance for both the Custom MobileNet 1 and Inception models. Conversely, the accuracy remained stable with Custom MobileNet 2. Specifically, Custom MobileNet 1 achieved an accuracy of 0.744, accompanied by a higher loss of 0.671. In comparison, the InceptionV3 model achieved an accuracy of 0.808, though it had a significantly higher loss of 7.463. Finally, the Custom MobileNet 2 yielded identical accuracy results (0.864) but with a higher loss value of 0.499.

After completing the training and validation phases, we conducted a test to calculate the confusion matrix. Fig. 9 displays the confusion matrices obtained from the models that achieved the best results in terms of accuracy using CIFAR-10 and Imagenet weights *i.e.,* Custom MobileNet 2 and InceptionV3. In detail, Fig. 9(a) presents the confusion matrix for the model trained using Custom MobileNet 2. The model successfully identified 2172 malware and 1882 trusted samples. However, it misclassified 172 malware and 462 trusted samples. The trained CNN achieved interesting loss, accuracy, and AUC results, where both classes reached identical values, 0.468, 0.864, and 0.864, respectively. Moreover, the trusted samples were classified with higher precision (0.916) compared to the malware class, which achieved an accuracy of 0.824. On the other hand, the malware class outperformed the trusted class in terms of recall and F-measure, with values of 0.926 and 0.872, respectively, compared to 0.802 and 0.855 for the trusted class. Fig. 9(b) reports the confusion matrix obtained from the model trained with the InceptionV3 using Imagenet weights that achieved higher results in terms of accuracy. In detail, the model was able to recognize 1721 malware and 2071 trusted samples correctly. On the other hand, it wrongly identified 623 malware samples as trusted and 273 trusted samples as malware. In terms of performance, both classes yielded identical results for loss, accuracy, and AUC, with values of 2.502, 0.808, and 0.808, respectively. However, the trusted class outperformed the malware class in recall and F-measure, achieving scores of 0.883 and 0.822, compared to 0.734 and 0.793 for the malware class. In contrast, the malware class achieved a higher precision of 0.863, while the trusted class reached 0.768.

4.2. Centralized approach

To compare the results obtained using the Federated Machine Learning approach, we employed the same Convolutional Neural Networks, the same dataset composed of images, and the same dataset division to train several models using a centralized approach. Moreover, we



(a) Accuracy achieved in each epoch during the training and validation phases.

(b) Loss achieved in each epoch during the training and validation phases.

Fig. 10. Accuracy and Loss over epochs during training and validation using the InceptionV3 architecture.

Table 7
Results obtained after test set execution using centralized approach.

Model	TRA	VA	TA	TRL	VL	TL
Inception	0.983	0.936	0.944	0.176	0.250	0.238
Inception	0.969	0.916	0.924	0.370	0.413	0.407
Custom MobileNet 1	0.995	0.893	0.892	0.014	0.403	0.420
Custom MobileNet 1	0.995	0.881	0.876	0.012	0.501	0.524
Custom MobileNet 2	0.996	0.911	0.904	0.014	0.329	0.355
Custom MobileNet 2	0.994	0.836	0.836	0.015	0.720	0.755

TRA: Training Accuracy **VA:** Validation Accuracy **TA:** Test Accuracy
TRL: Training Loss **VL:** Validation Accuracy **TL:** Test Accuracy.

employed the same image size used in previous experiments to obtain a better comparison. In detail, to identify the best hyperparameters, we also used a grid search algorithm. Table 4 reports the hyperparameters, allowing us to obtain the best results, while Table 7 reports the best outcome from the trained models.

IIn the latter, it is possible that we achieved the best results in terms of accuracy, training the model using the InceptionV3 architecture. In detail, we trained the models for 30 epochs with a batch size of 32 and a learning rate of 3e-5, using an image size of 160 × 3. With these settings, we achieved an accuracy, precision, recall, and F-Measure of 0.944, and an AUC score of 0.969. We also achieved interesting results with less accuracy regarding other architectures considered *i.e.,* Custom MobileNet 1 and Custom MobileNet 2. Unlike InceptionV3, we noticed that increasing the number of epochs for MobileNet led to a decline in accuracy, suggesting possible overfitting or inefficient learning with prolonged training. This event can also be observed in Table 7, where the training (TRA) accuracy was consistently higher than that of the validation (VA) and test (TA) phases. Starting from these considerations, we achieved interesting results using the Custom MobileNet 2 with accuracy, precision, recall, and F-Measure of 0.904, a loss of 0.355, and an AUC of 0.959. On the other hand, using Custom MobileNet 1, we obtained fewer results in terms of accuracy, precision, recall, and F-Measure (0.892).

Fig. 10 shows the trend achieved using the InceptionV3 architecture during the training and validation phases. In detail, Fig. 10(a) displays the accuracy reached over the epochs, while Fig. 10(b) reports the loss trend. In these figures, the red line represents the accuracy/loss values recorded throughout the training, while the blue line indicates the accuracy/loss achieved at each epoch during validation. Once the training and validation phases were concluded, we proceeded with the test phase.

At the end of the latter, we obtained the confusion matrix reported in Fig. 11. The model trained using InceptionV3 reached interesting performances while classifying both classes *i.e.,* malware and trusted. In detail, it correctly identified 2240 samples as malware but misclassified 104 of them. Additionally, it accurately recognized 2187 trusted

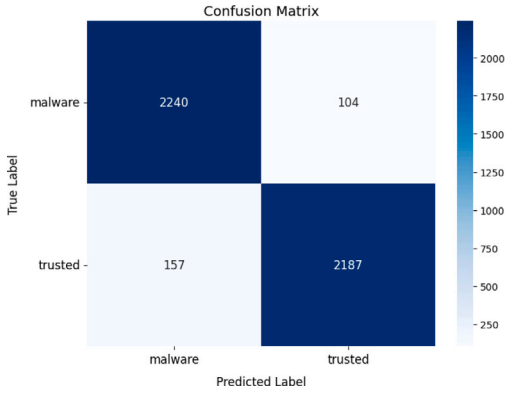
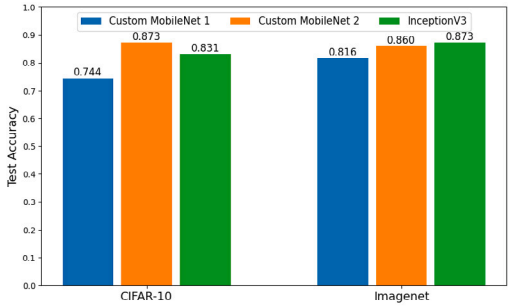


Fig. 11. Confusion matrix obtained from the model trained using InceptionV3.

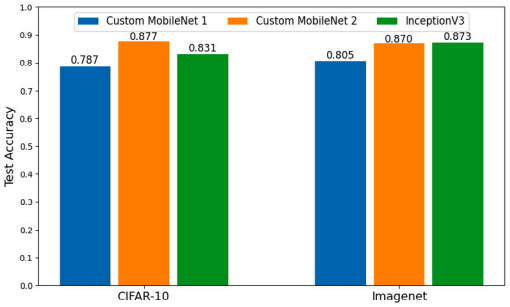
samples, though it incorrectly labeled 157 as malware. Regarding the performances achieved for each class, the model classified malware samples with an accuracy, F-measure, and AUC of 0.944, a precision of 0.934, and a recall value of 0.955. Similarly, for trusted samples, it gained an accuracy and AUC of 0.944, a precision of 0.954, a recall of 0.933, and an F-measure of 0.943.

4.3. Discussion

Concluded all experiments where we employed two different distributions *i.e.,* Default (IID data) and Dirichlet (non-IID data), we examined the results obtained. Figs. 12 and 13 show the accuracy values obtained after the test phase across three architectures: blue indicates Custom MobileNet 1, orange represents Custom MobileNet 2, and green corresponds to InceptionV3. In detail, Fig. 12 displays the outcomes reached using IID data, comparing the results obtained with (on the right) and without (on the left) applying the Clipping Norm Aggregator. In two out of three experiments without applying the Clipping Norm Aggregator, models pre-trained on the ImageNet dataset outperformed those with other weight initializations in accuracy. Specifically, the Custom MobileNet 1 and InceptionV3 architectures achieved accuracy values of 0.816 and 0.873, respectively. Interestingly, the Custom MobileNet 2 model, when initialized with CIFAR-10 weights instead of ImageNet, also reached an accuracy of 0.873. As the next step, using the same hyperparameters, we applied the Clipping Norm Aggregator and observed a similar trend in the results. Specifically, in two out of three cases, models pre-trained on ImageNet weights outperformed others. The Custom MobileNet 1 and InceptionV3 architectures achieved accuracy values of 0.805 and 0.873, respectively. However, the Custom MobileNet 2 model performed better with CIFAR-10 weights, achieving an accuracy of 0.877. After concluding the experiments with IID data,

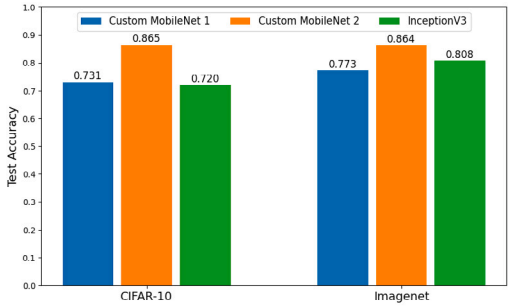


(a) Accuracy values obtained without the application of Clipping Norm Aggregator.

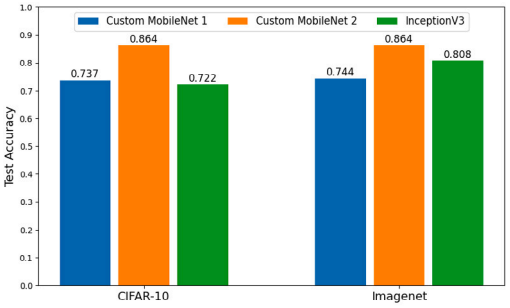


(b) Accuracy values obtained with the application of Clipping Norm Aggregator.

Fig. 12. Comparison of accuracy values achieved using IID data (Default distribution) with and without the application of Clipping Norm Aggregator.



(a) Accuracy values obtained without the application of Clipping Norm Aggregator.



(b) Accuracy values obtained with the application of Clipping Norm Aggregator.

Fig. 13. Comparison of accuracy values achieved using non-IID data (Dirichlet distribution) with and without the application of Clipping Norm Aggregator.

we extended our analysis to a more realistic scenario by employing non-IID data, utilizing the Dirichlet distribution. The results from the three Convolutional Neural Networks are presented in Fig. 13. Similarly to the IID experiments, Fig. 13(a) illustrates the outcomes without applying any norm, while Fig. 13(b) shows the results after employing the Clipping Norm Aggregator. In line with the trend observed in the IID experiments, Custom MobileNet 1 and Inception V3 achieved higher accuracy when using Imagenet weights than CIFAR-10 weights in the experiments without norm application. Specifically, Custom MobileNet 1 improved its accuracy from 0.731 (CIFAR-10 weights) to 0.773 (Imagenet weights), while Inception V3 saw a similar increase, rising from 0.720 to 0.808. Conversely, Custom MobileNet 2 performed slightly better with CIFAR-10 weights, achieving an accuracy of 0.865 compared to 0.864 with Imagenet weights.

To conclude the experimentation phase, we also applied the Clipping Norm Aggregator on non-IID data, observing improved results for Custom MobileNet 1 and InceptionV3 using Imagenet weights. Specifically, Custom MobileNet 1 increased from 0.737 to 0.744, while InceptionV3 improved from 0.722 to 0.808. Unlike the previous experiments, we obtained similar accuracy when using Custom MobileNet 2 with either CIFAR-10 or Imagenet weights (0.864). However, the loss was higher with Imagenet weights (0.499) than with CIFAR-10 weights (0.473). Generally, the higher results obtained using the model pre-trained using the ImageNet dataset can be attributed to multiple factors, like the feature extraction, where models pre-trained on ImageNet have learned rich feature representations that capture various aspects of images, or to the generalization, where models pre-trained on ImageNet have been exposed to a diverse range of visual patterns and concepts. In conclusion, we achieved interesting results using different Convolutional Neural Networks, mainly using custom versions of Mobilenet, which allowed us to obtain results in an average of 5 h. Different from this, InceptionV3 took around 16 h to get relevant outcomes. Moreover, the latter does not show an improvement in all metrics when using the clipping norm, as it results in higher loss values during the testing

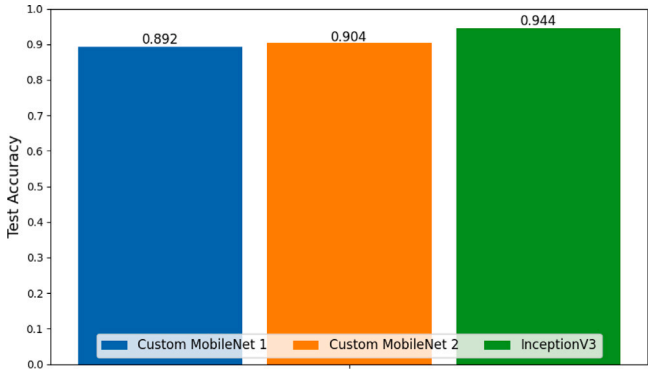


Fig. 14. Best accuracy results obtained using centralized approach.

phases across both the IID and non-IID distributions. We also performed several experiments using a centralized approach to compare the proposed method. In detail, we employed the same dataset, dataset split, and convolutional neural networks, obtaining interesting outcomes. Specifically, as denoted in Fig. 14, we received the best results in terms of accuracy using the InceptionV3 architecture (0.944). In addition, the proposed custom versions of the MobileNet also achieved interesting results, with the Custom MobileNet 1 reaching 0.892 and the Custom MobileNet 2 reaching 0.904. In conclusion, the proposed method to detect malware in the Android scenario leveraging Federated Machine Learning offers a more stable and privacy-preserving alternative to the traditional centralized method. By decentralizing the learning process, our approach mitigates privacy risks associated with centralized data collection while maintaining competitive performance. In particular, in this manuscript, we employed the Clipping Norm Aggregator, which allowed us to improve the stability by limiting the influence of extreme updates and preventing potential model degradation caused by

outliers or adversarial manipulations. Our results demonstrate that the proposed approach, particularly when utilizing the Custom MobileNet 2 architecture, remains efficient even without the application of CNA. This represents a crucial point in malware detection where potentially malicious users could disrupt the model's performance. Even though the results achieved using a centralized approach were higher in accuracy and lower in loss. On the other hand, this approach presents numerous challenges, like data privacy and possible adversarial attacks.

5. Conclusion and future work

In this paper, we proposed a method for privacy-preserving malware detection in the Android environment leveraging Federated Machine Learning. To this aim, we built a dataset of more than 40,000 *.apk* files divided into trusted and malware samples. Due to the enormous number of the latter, we created a robust and heterogeneous malware dataset of 71 different families of malicious applications. The next step was the conversion of all *.apk* files into images. This phase was possible thanks to the usage of a Python script written by the authors. In detail, once retrieved *.dex* files, using ApkTool, we converted them into images. After acquiring the images, we conducted several experiments using three different architectures on two pre-trained models using the CIFAR-10 and ImageNet datasets. In detail, we utilized two customized versions of MobileNet and InceptionV3, where the latter belongs to the state-of-the-art. Moreover, the hyperparameters were identified by executing the Grid Search algorithm. Determined the latter, two different distributions *i.e.,* Default Distribution (IID data), and Dirichlet Distribution (non-IID data) have been created. Moreover, we also proposed two different experiments for each distribution and architecture pre-trained using CIFAR-10 and Imagenet: one without using any norm and another where we employed the Clipping Norm aggregator.

As a future research plan, we aim to evaluate different models to enhance the robustness and applicability of the proposed system and understand whether achieving better performances in privacy-preserving Android malware detection is possible. In detail, we intend to use Differential Privacy and homomorphic encryption. Furthermore, we will adopt of Grad-CAM for explainability. As a matter of fact, we consider the Grad-CAM adoption crucial for understanding the rationale behind specific predictions, providing researchers and practitioners valuable insights to understand which part of the image under analysis can be symptomatic, from the model of view, of a certain prediction (malware or trusted).

CRediT authorship contribution statement

Giovanni Ciaramella: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Conceptualization. **Fabio Martinelli:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization. **Christian Peluso:** Writing – review & editing, Writing – original draft, Software. **Antonella Santone:** Writing – review & editing, Validation, Supervision, Conceptualization. **Francesco Mercaldo:** Writing – review & editing, Writing – original draft, Supervision, Validation, Software, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

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, by MUR - REASONING: foRmal mEthods for computAtional analySis for diagnOsis and progNosis in imagING - PRIN, e-DAI (Digital ecosystem for integrated analysis of heterogeneous health data related to high-impact diseases: innovative model of care and research), Health Operational Plan, FSC 2014–2020, PRIN-MUR-Ministry of Health, the National Plan for NRRP Complementary Investments D^3 4 Health: Digital Driven Diagnostics, prognostics and therapeutics for sustainable Health care, Progetto MolisCTe, Ministero delle Imprese e del Made in Italy, Italy, CUP: D33B22000060001, FORESEEN: FORmal mEthodS for attack dEtEction in autonomous driviNg systems CUP N.P2022WYAEW and ALOHA: a framework for monitoring the physical and psychological health status of the Worker through Object detection and federated machine learning, Call for Collaborative Research BRiC -2024, INAIL.

Data availability

Data will be made available on request.

References

[1] A. Bacci, A. Bartoli, F. Martinelli, E. Medvet, F. Mercaldo, Detection of obfuscation techniques in android applications, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, 2018, pp. 1–9.

[2] G. Canfora, F. Mercaldo, C.A. Visaggio, P. Di Notte, Metamorphic malware detection using code metrics, Inf. Secur. J.: A Glob. Perspect. 23 (3) (2014) 57–67.

[3] A. Cimitile, F. Martinelli, F. Mercaldo, et al., Machine learning meets iOS malware: Identifying malicious applications on apple environment, in: ICISSP, 2017, pp. 487–492.

[4] G. Ciaramella, L. Petrillo, M. Varilek, F. Mercaldo, G. Comandé, F. Martinelli, Leveraging pre-trained LLMs for GDPR compliance in online privacy policies, 2025.

[5] F. Mercaldo, X. Zhou, P. Huang, F. Martinelli, A. Santone, Machine learning for uterine cervix screening, in: 2022 IEEE 22nd International Conference on Bioinformatics and Bioengineering, BIBE, IEEE, 2022, pp. 71–74.

[6] H. He, H. Yang, F. Mercaldo, A. Santone, P. Huang, Isolation forest-voting fusion-multioutput: A stroke risk classification method based on the multidimensional output of abnormal sample detection, Comput. Methods Programs Biomed. 253 (2024) 108255.

[7] Y. Qu, X. Zhou, P. Huang, Y. Liu, F. Mercaldo, A. Santone, P. Feng, CGAM: An end-to-end causality graph attention Mamba network for esophageal pathology grading, Biomed. Signal Process. Control. 103 (2025) 107452.

[8] M. Di Giammarco, F. Mercaldo, X. Zhou, P. Huang, A. Santone, M. Cesarelli, F. Martinelli, A robust and explainable deep learning method for cervical cancer screening, in: International Conference on Applied Intelligence and Informatics, Springer, 2023, pp. 111–125.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.

[10] G. Ciaramella, F. Martinelli, F. Mercaldo, C. Peluso, A. Santone, An approach for privacy-preserving mobile malware detection through federated machine learning, in: Proc. of 26th International Conference on Enterprise Information Systems, 2024, pp. 553–563.

[11] C. Peluso, G. Ciaramella, F. Mercaldo, A. Santone, F. Martinelli, A federated learning-based android malware detector through differential privacy, in: International Conference on Computer Aided Systems Theory, Springer, 2024, pp. 307–319.

[12] M. Kakavand, M. Dabbagh, A. Dehghantanha, Application of machine learning algorithms for android malware detection, in: Proceedings of the 2018 International Conference on Computational Intelligence and Intelligent Systems, CIIS '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 32–36, http://dx.doi.org/10.1145/3293475.3293489, URL http://dx.doi.org/10.1145/3293475.3293489.

[13] J. Du, H. Chen, W. Zhon, Z. Liu, A. Xu, A dynamic and static combined android malicious code detection model based on SVM, in: 2018 5th International Conference on Systems and Informatics, ICSAI, 2018, http://dx.doi.org/10.1109/ICSAI.2018.8599356, 801–675.

[14] R.-H. Hsu, Y.-C. Wang, C.-I. Fan, B. Sun, T. Ban, T. Takahashi, T.-W. Wu, S.-W. Kao, A privacy-preserving federated learning system for android malware detection based on edge computing, in: 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), 2020, pp. 128–136, http://dx.doi.org/10.1109/AsiaJCIS50894.2020.00031.

[15] E. Odat, Q.M. Yaseen, A novel machine learning approach for android malware detection based on the co-existence of features, IEEE Access 11 (2023) 15471–15484, http://dx.doi.org/10.1109/ACCESS.2023.3244656.

[16] N.Z. Gorment, A. Selamat, L.K. Cheng, O. Krejcar, Machine learning algorithm for malware detection: Taxonomy, current challenges, and future directions, IEEE Access 11 (2023) 141045–141089, http://dx.doi.org/10.1109/ACCESS.2023.3256979.

[17] S. Sivakumar, S. Saminathan, R. Ranjana, M. Mohan, P.K. Pareek, Malware detection using the machine learning based modified partial swarm optimization approach, in: 2023 International Conference on Applied Intelligence and Sustainable Computing, ICAISC, 2023, pp. 1–5, http://dx.doi.org/10.1109/ICAISC58445.2023.10199796.

[18] Y. Bi, Y. Li, X. Feng, X. Mi, Enabling privacy-preserving cyber threat detection with federated learning, 2024, URL https://api.semanticscholar.org/CorpusID:269005672.

[19] N. Ferdous Aurna., M. Hossain., H. Ochiai., Y. Taenaka., L. Khan., Y. Kadobayashi., Banking malware detection: Leveraging federated learning with conditional model updates and client data heterogeneity, in: Proceedings of the 10th International Conference on Information Systems Security and Privacy - ICISSP, SciTePress, INSTICC, 2024, pp. 309–319, http://dx.doi.org/10.5220/0012409700003648.

[20] F. Ullah, G. Srivastava, S. Ullah, L. Mostarda, Privacy-preserving federated learning approach for distributed malware attacks with intermittent clients and image representation, IEEE Trans. Consum. Electron. 70 (1) (2024) 4585–4596, http://dx.doi.org/10.1109/TCE.2023.3342644.

[21] Z. Pan, J. Sheldon, P. Mishra, Hardware-assisted malware detection and localization using explainable machine learning, IEEE Trans. Comput. 71 (12) (2022) 3308–3321, http://dx.doi.org/10.1109/TC.2022.3150573.

[22] W. Li, H. Hacid, E. Almazrouei, M. Debbah, A comprehensive review and a taxonomy of edge machine learning: Requirements, paradigms, and techniques, AI 4 (3) (2023) 729–786, http://dx.doi.org/10.3390/ai4030039, URL http://dx.doi.org/10.3390/ai4030039.

[23] X. Jin, C. Katsis, F. Sang, J. Sun, A. Kundu, R. Kompella, Edge security: Challenges and issues, 2022, arXiv:2206.07164.

[24] S. He, B. Tang, B. Zhang, J. Shao, X. Ouyang, D.N. Nugraha, B. Luo, FedKit: Enabling cross-platform federated learning for android and iOS, 2024, URL https://api.semanticscholar.org/CorpusID:267740529.

[25] A. Shabtai, Y. Fledel, Y. Elovici, Automated static code analysis for classifying android applications using machine learning, in: 2010 International Conference on Computational Intelligence and Security, IEEE, 2010, pp. 329–333.

[26] G. Shrivastava, P. Kumar, Android application behavioural analysis for data leakage, Expert Syst. 38 (1) (2021) e12468.

[27] W. Chen, Y. Wang, Y. Guan, J. Newman, L. Lin, S. Reinders, Forensic analysis of android steganography apps, in: Advances in Digital Forensics XIV: 14th IFIP WG 11.9 International Conference, New Delhi, India, January 3-5, 2018, Revised Selected Papers 14, Springer, 2018, pp. 293–312.

[28] F. Mercaldo, G. Ciaramella, A. Santone, F. Martinelli, Obfuscated mobile malware detection by means of dynamic analysis and explainable deep learning, in: Proceedings of the 18th International Conference on Availability, Reliability and Security, 2023, pp. 1–10.

[29] J. Zhang, Z. Qin, H. Yin, L. Ou, S. Xiao, Y. Hu, Malware variant detection using opcode image recognition with small training sets, in: 2016 25th International Conference on Computer Communication and Networks, ICCCN, IEEE, 2016, pp. 1–9.

[30] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al., Searching for mobilenetv3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

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

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[34] Q. Li, Y. Diao, Q. Chen, B. He, Federated learning on non-iid data silos: An experimental study, in: 2022 IEEE 38th International Conference on Data Engineering, ICDE, IEEE, 2022, pp. 965–978.

[35] G. Andrew, O. Thakkar, B. McMahan, S. Ramaswamy, Differentially private learning with adaptive clipping, Adv. Neural Inf. Process. Syst. 34 (2021) 17455–17466.