International Journal of Information Security (2026) 25:102
https://doi.org/10.1007/s10207-026-01268-4

REGULAR CONTRIBUTION

An explainable privacy-preserving method for mobile malware detection through federated machine learning

Giovanni Ciaramella[1,2] · Fabio Martinelli[3] · Antonella Santone[4] · Francesco Mercaldo[4]

Received: 13 March 2025 / Accepted: 28 April 2026

Abstract
Over the last few years, the number of cyberattacks has increased drastically, becoming more resistant to detection systems introduced by researchers. Traditional centralized approaches have been replaced by distributed methods such as Federated Machine Learning to enhance efficiency. The latter enables the creation of a central model by training multiple clients locally and aggregating only the model's weights rather than sharing raw data. This approach enhances computational efficiency and significantly improves privacy, as sensitive data remains on the client's devices. This research paper proposes a method for detecting malware in the Android environment leveraging Federated Machine Learning. In detail, we employed two datasets: one of almost 20,000 malicious and trustworthy Android packages and the second of nearly 7,250 applications which were converted into grayscale images by representing their smali code as pixel intensities, enabling Convolutional Neural Networks to process them effectively. After concluding the dataset composition, we trained several models using two Convolutional Neural Networks, such as InceptionV3 and a custom version of MobileNet. After identifying the best-performing model on the binary dataset, we applied the same hyperparameters to train, validate, and test a second dataset composed of four distinct classes. Additionally, we compared the performance of the federated approach with that of centralized training using the same model architectures. At the end of this process, we identified the best-trained model on the binary dataset and applied the two class activation map algorithms to perform explainability using the model. Moreover, as the last step, we also applied the Structural Similarity Index Measure to quantify the consistency and reliability of the generated heatmaps.

Keywords Android · Federated learning · Malware · Security · Privacy · Explainable artificial intelligence

1 Introduction

Widespread usage of mobile devices has caused an increase in cyberattacks, driving up the costs associated with protecting personal user data. In a report published by Statista in August 2024[1], it shows the cybersecurity market was continuously increasing over the past years, with an expected cost of 15.63 trillion U.S. dollars in 2029 to counter actions performed by malicious users. Despite increased resources to curb this problem, in 2023, 73%[2] board members of worldwide companies see a risk of a material cyber attack in the future. One of the most prevalent cyberattacks in recent years has targeted the theft of personal information. In 2023, most Chief Information Security Officers (CISOs) worldwide expressed that 70% of sensitive data breaches in organizations result from negligent users. Furthermore, 48.1% of respondents indicated that compromised systems led to data loss. Additionally, roughly 20% of respondents cited malicious employees or contractors as the source of

✉ Giovanni Ciaramella
giovanni.ciaramella@iit.cnr.it

✉ Francesco Mercaldo
francesco.mercaldo@unimol.it

Fabio Martinelli
fabio.martinelli@icar.cnr.it

Antonella Santone
antonella.santone@unimol.it

1 IMT School for Advanced Studies Lucca, Lucca, Italy

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

3 Institute for High Performance Computing and Networking, National Research Council of Italy (CNR), Rende (CS), Italy

4 University of Molise, Campobasso, Italy

[1] https://www.statista.com/forecasts/1280009/cost-cybercrime-worldwide

[2] https://www.statista.com/statistics/1367360/risk-cyber-attack-companies-worldwide-board-directors-by-country/

Published online: 28 May 2026

their incidents[3]. For this reason, privacy has also become increasingly important during the last few years. People worldwide use Internet of Things (IoT) devices that contain personal information every day, making data protection a critical issue. To address this, many governments have introduced regulations. In April 2016 (implemented in May 2018), the European Union introduced the GDPR (General Data Protection Regulation) to regulate the usage of the personal data of each European citizen. Another essential topic widely adopted nowadays is Artificial Intelligence (AI), which is used in several fields such as computer science, healthcare, and engineering. In many cases, AI relies on personal data, raising concerns about user privacy. To address this, Google introduced the concept of Federated Machine Learning (FML) in 2017 [22]. [15] defined Federated Machine Learning as "a machine learning setting where multiple entities collaborate in solving s machine learning problem, under the coordination of a server. Each client's raw data is stored locally and not exchanged or transferred; focused updates intended for immediate aggregation are used to achieve the learning objective". Because many approaches to malware identification are derived from several methods in the literature, it is advantageous to adopt a Federated Machine Learning-based approach model. In the traditional sense of things, malware detection models use centralized datasets that may be limited in diversity and size, thereby not being able to generalize well on other new or different malware, like Machine Learning and Deep Learning [27, 30]. This is unlike Federated Machine Learning, where institutions use their various devices to train a model without disclosing their sensitive data. The latter training method recognizes that data sets differ across environments, thus leading to more comprehensive and robust models. In particular, mobile malware detection represents a critical and rapidly evolving cybersecurity challenge. Android devices account for the largest share of the global mobile operating system market, making them a primary target for attackers. Unlike traditional desktop environments, mobile ecosystems are highly fragmented, applications are frequently distributed through third-party marketplaces, and devices operate under resource constraints. Moreover, user data and application usage patterns are inherently decentralized across millions of smartphones, making centralized data collection both impractical and privacy-sensitive. These characteristics make Android malware detection a particularly suitable domain for Federated Machine Learning, as model training can occur directly on devices without transferring sensitive application data. Consequently, the proposed approach is not merely a generic application of FML but an ad hoc solution for the mobile environment, accounting for decentralized data availability, device constraints, heterogeneous malware families, and privacy regulations.

This research proposes a method for identifying malware in the Android environment, leveraging Federated Machine Learning. Specifically, we trained and tested models using two distinct Convolutional Neural Networks (CNNs) (*i.e.,* InceptionV3 and Custom MobileNet) using datasets composed of images retrieved from the smali code. Using that method we were able to preserve structural and functional patterns of the application under analysis. Specifically, we created two datasets: one for binary classification comprising 20,000 APKs and another for four-class classification, including 7,250 APKs, including three malware families and trusted samples. The latter dataset was used to validate the proposed method and to evaluate the efficiency and capability of the model for multi-class malware identification. These datasets were employed to perform experiments in centralized and decentralized settings, in which the latter used non-Independent and Identically Distributed (non-IID) data to simulate a real-world scenario better. Subsequently, we identified the best-performing model trained on a binary dataset. On the latter, we applied two Class Activation Map (CAM) algorithms, namely Gradient-weighted Class Activation Mapping (Grad-CAM) and Score-weighted Class Activation Mapping (Score-CAM), for explainability. As the last step, we calculated the similarity between the heatmaps obtained for each class and for each explainability algorithm adopted, using the Structural Similarity Index Measure (SSIM). To the best of our knowledge, this is the first article to introduce explainability in image-based malware detection using images derived from smali code, leveraging Federated Machine Learning.

The paper is organized as follows: the next section provides an overview of the state-of-the-art regarding using centralized and decentralized approaches in Android malware detection. Section 3 introduces the concept of Federated Machine Learning, providing an overview of its technical aspects and outlining its main advantages and challenges. Sect. 4 then presents the proposed methodology, while Sect. 5 reports the results of the experimental evaluation conducted on real-world Android applications. The concluding section discusses the findings and potential directions for future research.

2 Related work

Cybersecurity become a hot topic during the last few years. For this reason, researchers conducted different studies to identify new solutions to mitigate attacks. In this section, we furnish a literature review of some techniques introduced over the years. Table 1 presents a structured overview of the

[3] https://www.statista.com/statistics/1387393/loss-sensitive-information-organizations-cause-worldwide/

Table 1 Comparative analysis with previous works

Refrerence	Dataset Composition	Training Method		Data Distribution	Platform	XAI	Accuracy	IF/IM-SSIM
		C	F					
Sewak *et al.* [30]	Features extracted from PE files	✓	✗	-	Windows	✗	0.997	✗
Mercaldo *et al.* [23]	APK files converted into images using extracted system calls	✓	✗	-	Android	✓	> 0.728	✗
Hemalatha *et al.* [9]	PE files converted into grayscale images	✓	✗	-	Windows	✗	> 0.980	✗
Ibrahim *et al.* [12]	Feature extaction from APK files	✓	✗	-	Android	✗	> 0.970	✗
Sharma *et al.* [31]	Feature extraction from Smali code	✓	✗	-	Android	✗	0.984	✗
Yapici *et al.* [40]	APK files converted into grayscale and RGB images	✓	✗	-	Android	✗	∼0.987	✗
Gálvez *et al.* [8]	Feature extaction from APK files	✗	✓	IID	Android	✗	N/A	✗
Rey *et al.* [28]	Network packet	✗	✓	non-IID	IoT	✗	∼0.990	✗
Lin *et al.* [19]	Feature extraction from PE files	✗	✓	N/A	Windows	✗	∼0.916	✗
Jiang *et al.* [13]	Feature extaction from APK files	✓	✓	non-IID	Android	✗	0.913	✗
Ciaramella *et al.* [4]	APK files converted into grayscale images	✓	✓	IID and non-IID	Android	✗	F: > 0.873 C: 0.944	✗
Yadav *et al.* [39]	Feature extaction from APK files	✓	✓	IID	Android	✗	F and C: > 0.980	✗
Our Method	**APK files converted into images using smali code**	✓	✓	**non-IID**	**Android**	✓	**F: > 0.918 C: 0.946**	✓

LEGEND:
C: Centralized; **FML:** Federated Machine Learning; **XAI:** Explainability

results to facilitate a more comprehensive comparison of the proposed method with other studies in the literature.

Researchers in [30] compared models to identify malware trained and tested using Deep Learning and Machine Learning, focusing on static feature-based analysis of executable files. Moreover, they achieved higher accuracy with the Random Forest algorithm in each of the proposed experiments, particularly when compared against Support Vector Machines, k-Nearest Neighbors, and neural network-based classifiers. Different from the authors, we compared two different types of settings, *i.e.,* centralized and a decentralized approach. Moreover, we evaluated two distinct datasets: one for binary classification and the other for multi-class classification. As the final step, we applied two distinct CAM algorithms to provide explainability, and then we calculated the Structural Similarity Index Measure.

In [23], researchers utilized deep learning to demonstrate that it is possible to distinguish between malware and trusted samples, even when malicious users employ obfuscation techniques. To compose the dataset, the authors emulated multiple APK files and executed a variety of user gestures. During this process, they extracted opcode sequences and mapped each opcode to a specific RGB color, thereby generating image representations. After training and testing their models, they also applied explainability methods to ensure the models' resilience and reliability. Unlike them, in our proposed article, we extracted the smali code and generated images from it to compose our dataset. Moreover, we also compared a centralized approach with a federated setting using the same datasets. Concluding, like them we also utilized explainability using two state-of-the-art algorithms *i.e.,* Grad-CAM and Score-CAM on the best Federated Machine Learning model trained on the binary dataset.

Authors in [9] proposed a methodology to identify malware leveraging the DenseNet *i.e.,* a Deep Learning architecture available in the literature. The outcomes were satisfactory, achieving an accuracy value of 0.984. Moreover, to create the final dataset, a set of open science datasets was combined to build a balanced dataset of almost 20k images of general malware from publicly available datasets. Differently from them, we employed two datasets (one for binary and the other for multi-class classification) using APK files. In addition, we obtained the images from the Android Packages using the smali code retrieved after the reverse engineering. Moreover, we combined the centralized approach with the decentralized approaches, applying two explainability algorithms to them.

In [12], they proposed a deep learning-based method for classifying Android malware, utilizing features like permissions, API calls, and opcode-level information to train and evaluate the model. Their approach achieved 0.980 precision in identifying benign samples and 1.000 precision in detecting malware samples. In contrast, our approach leverages images obtained from the smali code for training and testing models. Moreover, we employed two datasets to compare a centralized and a federated approach and applied two different CAM algorithms for explainability.

The authors in [31] proposed a method for Android malware detection named MOSDroid, which is resilient to obfuscation. Specifically, they are statically analyzing Android applications to extract opcode sequences from disassembled code using APKTool using the Data-MOS. Moreover, to perform the obfuscation, they used a tool named Obfuscapk. Subsequently, the adopted machine learning classifiers were used to train several models, achieving interesting accuracy (0.984) and AUC values (0.994). Similarly, we extracted smali code using APKTool. However, instead of employing a machine learning-based approach, we trained models using features extracted from the code; we then converted the code into images using a state-of-the-art tool. Moreover, the images obtained were used to create two datasets: one for binary classification and the other for multi-class classification. Subsequently, we employed them to train models in centralized and decentralized settings, comparing the results.

Yapici *et al.* in [40] proposed an approach for Android malware detection and classification leveraging Deep Learning. Specifically, researchers converted a set of APK files from bytecode to grayscale and RGB. Moreover, they trained several models using state-of-the-art architectures, achieving high detection accuracy and effective malware family classification, confirming that visual patterns extracted from Android binaries can successfully capture malicious behavior. Unlike them, we generated images from smali code and composed two datasets: one for binary classification and one for multi-class classification. Moreover, we trained several models using a centralized and a decentralized approach (federated machine learning). On the best model obtained with the decentralized approach and a binary dataset, we applied the explainability process using two different CAM algorithms. Moreover, we calculated the Structural Similarity Index Measure.

Authors in [8] presented LiM (Less is More), a framework for detecting malware in the Android environment leveraging Federated Machine Learning. To perform binary classification, they employed a dataset of 50,000 APK files perfectly balanced between Malware and Trusted samples. At the end of the experimentations, Gálvez *et al.* reached good results. In our contribution, we trained and tested several models using both centralized and decentralized approaches, leveraging two datasets of images obtained from smali code. In [8], researchers extracted features from files such as declared permissions, activities, and services. Specifically, we generated a dataset for binary and another one composed of four classes (one trusted and three malware families). Concluding, we selected the best-performing model trained on the binary

classification dataset and applied two CAM algorithms for explainability reasons.

Rey *et al.* in [28] presented a method to detect cyberattacks that affect IoT devices. Moreover, they used non-IID data to recreate a real-world scenario and obtained good results. Unlike them, we focused only on APK files generating from them images using smali code. Moreover, we applied two different CAM algorithms, such as Grad-CAM and Score-CAM, to the best-performing model trained on the binary dataset, and quantitatively compared the resulting explanations using the IF/IM-SSIM algorithm.

In [19], researchers proposed a Federated Machine Learning method to identify Windows malware. At the end of the process, they reached an accuracy value of 0.916. In detail, Lin *et al.* presented a multi-class classification, and to train the model, employed a set of features like 1-gram, Entropy, and Images. Unlike them, we proposed a method in this paper to classify malware and trusted applications in the Android environment. Moreover, using the decentralized approach, we proposed non-IID data and explainability leveraged two CAM algorithms from the state of the art. Moreover, we compared the distributed approach with a centralized setting using the same CNNs.

The authors in [13] introduced FedHGCDroid, a method for classifying malware on Android devices. They developed a novel multidimensional malware classification model that leverages Convolutional Neural Networks and Graph Neural Networks. The designed model is efficient in capturing and extracting features related to malicious behavior, enabling more accurate malware classification. Additionally, the researchers addressed the challenge of non-IID data in their approach. In the research we proposed, we only employed CNNs belonging a centralized and distributed settings. Moreover, we applied the Grad-CAM and Score-CAM algorithms to the best-performing model trained on a binary dataset for explanatory purposes.

Researchers in [4] presented a method for identifying malware in the Android environment that leverages Federated Machine Learning. Specifically, they converted a large number of APK files into grayscale images from their dex files. After this, we also compared the IID and non-IID approaches with a centralized approach. Similarly, we compared the federated scenario with a baseline method across two datasets: one for binary classification and the other for multi-class classification. Moreover, unlike these works, we additionally incorporate an explainability analysis by applying two CAM techniques, Grad-CAM and Score-CAM, and computing the Structural Similarity Index Measure to quantitatively assess the consistency and reliability of the generated visual explanations, rather than relying solely on qualitative inspection.

Yadav *et al.* in [39] proposed a malware detection framework based on Federated Machine Learning for the Internet of Medical Things environment. Specifically, they employed two datasets from the state of the art, namely Drebin and Malgenome, from which they extracted features and trained several models using IID data. Researchers also compared results from the decentralized approach with those from a centralized setting, achieving strong performance in terms of accuracy and AUC. Unlike them, we employed CNNs to train our models in both decentralized and centralized settings, using two datasets of images retrieved from the smali code. Moreover, after identifying the best-performing model trained on the binary classification dataset, we applied two CAM-based techniques to it for explainability.

3 Background

In this Section, we furnish an overview of Federated Machine Learning. Specifically, Subsection 3.1 introduces its historical background and technical aspects, while Subsections 3.2 and 3.3 focus on its applications in cybersecurity and its associated advantages and challenges, respectively.

3.1 Overview and technical aspects

The introduction of Federated Machine Learning in 2016 represented a significant technological innovation. The main characteristic of this technology is data privacy, which has become a crucial topic over the last few years. Several governments have introduced directives to regulate the use of personal data over the past few years. For example, the European Union introduced the GDPR (General Data Protection Regulation) on May 25, 2018, while on January 1, 2020, the CCPA (California Consumer Privacy Act) took effect, which builds on HIPAA (Health Insurance Portability and Accountability Act). In Federated Machine Learning, privacy is enhanced by a unique approach: instead of transferring the entire dataset to a central server, only model updates (*e.g.,* gradients) are sent. This methodology ensures that personal data remains on the user's device, significantly reducing the risk of data breaches. FML minimizes exposure by keeping raw data localized and aligns with stringent data privacy regulations, making it a robust solution for privacy-preserving machine learning. The central concept of federated approach is to minimize the objective function:

$$F(x) = \mathbb{E}_{i\sim P}\left[F_i(x)\right] \text{ where } F_i(x) = \mathbb{E}_{\epsilon\sim D_i}\left[f_i(x,\epsilon)\right]$$

Where $x \in \mathbb{R}^d$ represents the parameter for the global model, $F_i : \mathbb{R}^d \to \mathbb{R}$ denotes the local objective function at client i, and P is a distribution over the population of clients I. The local loss functions $f_i(x,\epsilon)$ are often consistent across clients, but the local data distributions D_i can vary, reflecting data heterogeneity. From the practical point of view, the server plays a central role in this methodology during

the training process. The latter phase comprises five key stages: selecting clients, broadcasting, client-side computation, aggregation, and model updating. In the first stage, the server selects a set of clients using specific criteria *i.e.,* mechanisms and metrics. In the first group, possible selection methods include mathematical optimization and clustering; for metrics, a potential way to select a client is based on data type, time, loss, or a combination of all these [21]. Once the clients are identified and selected, the server shares the current model weights with them to proceed to the third step, *i.e.,* the client computation. Subsequently, each client trains the model using its own data. As the next step, *i.e.,* the training process is completed across clients, the updates are aggregated on the central server using techniques such as averaging, zeroing, differential privacy, or clipping. In the final step, the server updates the model using the aggregated updates from the clients that contributed to the current round of computation.

3.2 Applications of federated learning in cybersecurity

After its introduction, several fields, such as healthcare, finance, and manufacturing, adopted Federated Machine Learning. All of the above have in common the need to protect personal data and to improve the efficiency of collaborative learning. In detail, in the healthcare case, FML enables medical institutions to collaborate on developing predictive models while keeping patient data secure and private [5, 26]. For instance, a federated approach is applied to predictive maintenance and quality control in manufacturing settings [1]. Due to the drastic increase in attacks and cyber threats, researchers and cyber experts have begun implementing Federated Machine Learning in cybersecurity strategies [3]. This approach allows organizations to leverage collective intelligence across many distributed datasets while protecting sensitive information. This aspect of FML is particularly appealing for industries that handle personal or sensitive information, such as healthcare, finance, and legal services, where regulations like the GDPR impose strict limitations on data sharing and privacy. Organizations can use a federated approach to enhance their cybersecurity measures while remaining compliant with these regulations. FML also enables more effective malware detection by enabling threat identification without moving raw data to a central server [14, 35]. By utilizing a federated approach, organizations can train models collaboratively using their local data by sharing only updates to the model or insight-driven data drift, rather than transmitting raw data. This is a valuable process for improving machine learning that also addresses sensitive matters, as it minimizes the risk of exposure while making models robust due to the vast diversity of data sources. Furthermore, Federated Machine Learning can adapt to emerging threats in real time, enabling quicker responses and more effective mitigation strategies. As cyber threats continue to evolve, the ability to train models on decentralized data enhances the accuracy and effectiveness of cybersecurity defenses, providing organizations with a crucial advantage in protecting their networks and sensitive information.

3.3 Advantages and challenges

Federated Machine Learning offers a range of advantages, but due to its recent introduction and complex approach, it faces numerous challenges. One of the best benefits of using FML is its ability to enhance data privacy by keeping data localized and addressing concerns about sensitive information. Instead of sending the device's raw local data to a central server, this machine-learning approach sends only model updates (*e.g.,* gradients or weights) [18]. This approach promotes privacy and significantly reduces communication costs [21]. Sending only model updates rather than raw data reduces the information shared between clients and the central server. This reduces strain on network bandwidth and improves speed by lowering latency [41]. In detail, data stored on local devices are processed by the global model and aggregated into the local model. Unlike other machine-learning approaches, collaboration with another device can also improve learning quality [25]. The latter is possible by using a diverse dataset across various devices, which helps improve the accuracy of the learned global model and enhances learning performance. This is especially true when devices are selected efficiently [6]. On the other hand, Federated Machine Learning faces several problems due to its recent introduction. One of the most critical problems is the usage of heterogeneous data. This challenge arises because multiple devices are involved in this machine-learning approach, each containing different data types. Additionally, training local models in Federated Machine Learning with non-IID data can increase convergence time and reduce the accuracy of the global model. Over the years, researchers have proposed several solutions to curb this problem, and one of the most widely accepted is client selection. In detail, [20, 42] proposed methods to select devices by considering their datasets' characteristics. Even though using several clients can help the model achieve good convergence and accuracy, this approach can pose a problem. In detail, using many clients or a large number of model parameters can make communication intensive, leading to high bandwidth consumption, high latency, and scalability issues. Solutions include compressing updates, reducing communication frequency, and using techniques like federated averaging, but these only partially address the challenges. Another challenge in FML is represented by privacy. The latter is also the strength of this machine-learning approach, but it is not immune to privacy risks.

Possible privacy risks include inference attacks, data poisoning, and adding noise to updates via differential privacy. In detail, in the first case of attack, although raw data isn't shared, model updates can sometimes leak sensitive information about the underlying data, especially if the model update patterns reveal unique data characteristics [24]; while regarding data poisoning, malicious clients can inject poisoned data into the model update process to influence the final model in undesirable ways [33]. In conclusion, security also presents a challenge in Federated Machine Learning. Systems are exposed to potential security risks since multiple clients are involved in the training process [32]. In detail, compromised clients can send corrupted updates to manipulate the model subtly, or adversaries can intercept and alter model updates as they travel between clients and the server, leading to inaccurate or harmful model behavior.

4 The method

In this Section, we present and describe in detail the proposed method to classify malware and trusted Android applications leveraging Federated Machine Learning. Specifically, in the method presented, we obtained images from the smali code belonging to a dataset of malicious and trustworthy APKs. After this phase, we trained and tested models leveraging federated (adopting non-IID data) to classify the images obtained. Subsequently, we applied two CAM algorithms *i.e., Grad-CAM and Score-CAM* to identify image regions responsible for a specific prediction using the best models obtained. As a final step, we computed the similarities among the generated CAMs to better evaluate the regions identified. For this purpose, we employed the IF/IM-SSIM algorithm. Figure 1 illustrates the proposed method.

4.1 Dataset composition

The creation of a good dataset helps obtain engaging outcomes from the experiments. In this study, we employed two datasets comprising malware and trusted samples. Particularly, malicious samples coming from the Android Malware Dataset (AMD) [17], which is freely available for research purposes. The first dataset (named Dataset A) was designed to perform binary classification, and the malware folder is composed of several different families, including *Dowgin, FakeInst, Mecor, Youmi, Fusob, Kuguo, BankBot*, and *Jisut*. On the other hand, the second dataset (named Dataset B) was designed to perform multi-class classification and is composed of three malware families: *FakeInst, Fusob*, and *Mecor*. Regarding legitimate applications, they were gathered from Google Play[4],*i.e.,* Google's official app store. This was possible using a Python crawler developed to obtain APK files from the different categories of apps available on the market. Once downloaded trusted applications, we submitted them to Virustotal[5] to verify their trustworthiness.

Table 2 Sample distribution across classes and training, validation, and test phases for Dataset A and Dataset B

	Class	Training	Validation	Test	NSC
Dataset A	*Malware*	7,988	998	999	9,985
	Trusted	7,991	998	1,000	9,989
NSP		15,979	1,996	1,999	
	Class	**Training**	**Validation**	**Test**	**NSC**
Dataset B	*FakeInst*	1,729	216	217	2,162
	Fusob	1,020	127	128	1,275
	Mecor	1,456	182	182	1,820
	Trusted	1,599	199	201	1,999
NSP		5,804	724	728	

Legend: **NSP**: Number of Samples per Phase; **NSC**: Number of Samples per Class

Once we converted both datasets composed of APK files into images using the approach described in Sect. 4.2, we divided them into subfolders using a standard 80-10-10 split adopted in the literature. Regarding Dataset A, composed of 19,974, where 15,979 samples were employed to train models, 1,996 during the validation phase, and 1,999 to test. On the other hand, Dataset B comprised 7,256 samples, split into 5,867 for training, 724 for validation, and 728 for testing. Specifically, Table 2 reports the exact division of samples across datasets and their respective classes.

4.2 Image generation

Having selected the two APK datasets, we advanced to evaluating and identifying the most effective approach for feature extraction using image-based representations. Every day, users use applications on their smartphones for different reasons. These applications can be developed using various programming languages, such as Java, Kotlin, C++, and Swift. In some cases, programmers use frameworks to create hybrid apps that run on any platform using a single code base. Once compiled, applications are saved in the APK (Android Package) format, which contains all the necessary components to install correctly on an Android device. Over the years, different techniques have been introduced to retrieve images from an Android Package. In detail, in literature it is possible to find research papers where authors create images using API-call [36], System Call [2], or converting the DEX (Dalvik Executable) file [7]. In this article, we employed a different technique to obtain the dataset to

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

[5] https://www.virustotal.com/gui/

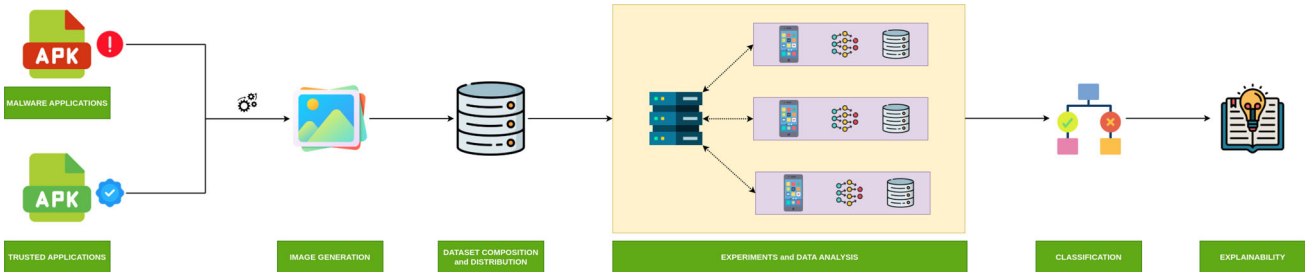


Fig. 1 The proposed method to identify malware in Android environment leveraging Federated Machine Learning and explainability

perform the experiments, *i.e.,* using the smali code [11]. The latter is a human-readable representation of Android application bytecode, usually obtained after a reverse engineering phase using specific tools. In our approach, we performed the reverse engineering process on several Android applications using APKtool[6], and after that, we converted them into images. To accomplish this, we first create a translation rule pairing each set of commands with an ASCII character. Meanwhile, the position of each method will be logged so that the malware analyst can locate suspicious code sections. The resulting images are presented in Fig 2a and 2b . Specifically, Fig 2a displays a malware sample identified by the hash code (MD5) 0a30ba12a76469ad0362d801fb82a546[7], while Fig 2b shows a trusted sample, also identified by the hash code (MD5) 1a3e94d252152ef1da92c08ce8290aa0[8].

Moreover, Fig 3a 3b 3c 3d illustrate example images from three malware families, such as FakeInst, Fusob, and Mecor, as well as trusted samples. Specifically, Fig 3a presents a FakeInst sample (MD5: a025a65da6004a83804ef dcb4fb74018)[9], Fig 3b shows a Fusob sample (MD5: 96a 98430e533233bc8e850b10e347f0c)[10], Fig 3c reports a Mecor sample (MD5: f562618897de73ff8d5329e095e59c 52)[11], and Fig 3d depicts a trusted application (MD5: 715eafc151d7880a45f3b585ee55d05f)[12].

4.3 Experiments and data analysis

Once we obtained the dataset, we used it to train and test models leveraging centralized and decentralized approaches both using CNNs. Unlike other AI approaches, Federated Machine Learning works using several devices. For this reason, to represent the diffusion among devices, it is possible to employ two different types of distributions *i.e.,* Independent and Identically Distributed (IID) and non-Independent and Identically Distributed (non-IID). In the first scenario, data from all clients are independently sampled from the same distribution, sharing identical statistical properties. However, in real-world settings, achieving perfect IID data is challenging due to data heterogeneity and the randomness of the events that shape them. For these reasons, we used non-IID data in this research paper, in which each device exhibits different statistical properties. In detail, we employed a cluster of 20 devices, distributing the generated dataset across them using the Dirichlet distribution [16]. The Dirichlet distribution is widely adopted in Federated Machine Learning research to simulate controlled statistical heterogeneity across clients. In our case, this strategy enables the generation of comparable non-IID partitions for Dataset A (binary classification) and Dataset B (multiclass classification), despite their different class cardinalities and sample sizes. This choice ensures a consistent evaluation framework across heterogeneous dataset configurations. Furthermore, considering the differences in dataset sizes, we selected 20 clients to provide a sufficient level of data fragmentation while preserving a meaningful number of samples per client, thereby enabling a balanced and comparable federated analysis. Figure 4 shows the distribution of the samples among the clients. Specifically, the distribution of Dataset A (*i.e.,* is reported in Figure) 4a while Fig. 4b reports the multi-class dataset distribution.

To create our models leveraging both methods, we employed two different architectures, InceptionV3 and a custom version of MobileNet. The first was introduced in [34] to achieve high accuracy while maintaining computational efficiency, making it suitable for tasks requiring large-scale image recognition. Regarding MobileNet, it was presented by

[6] https://apktool.org/

[7] https://www.virustotal.com/gui/file/0a30ba12a76469ad0362d801fb82a546

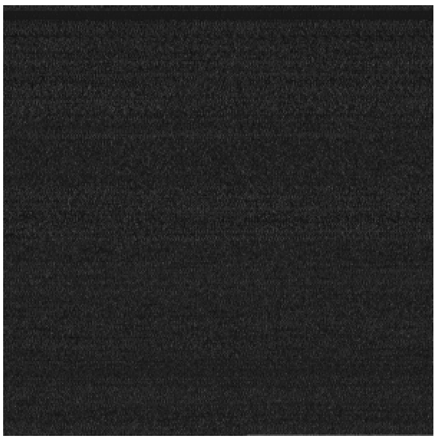
[8] https://www.virustotal.com/gui/file/1a3e94d252152ef1da92c08ce8290aa0

[9] https://www.virustotal.com/gui/file/a025a65da6004a83804efdcb4fb74018

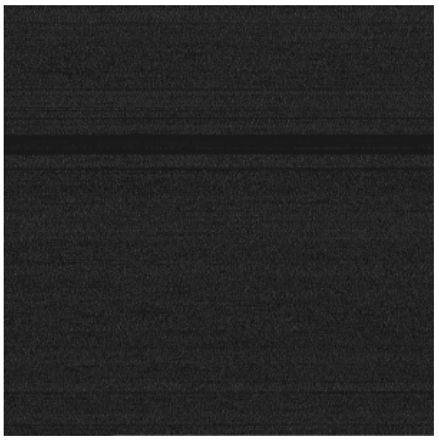
[10] https://www.virustotal.com/gui/file/96a98430e533233bc8e850b10e347f0c

[11] https://www.virustotal.com/gui/file/f562618897de73ff8d5329e095e59c52

[12] https://www.virustotal.com/gui/file/715eafc151d7880a45f3b585ee55d05f



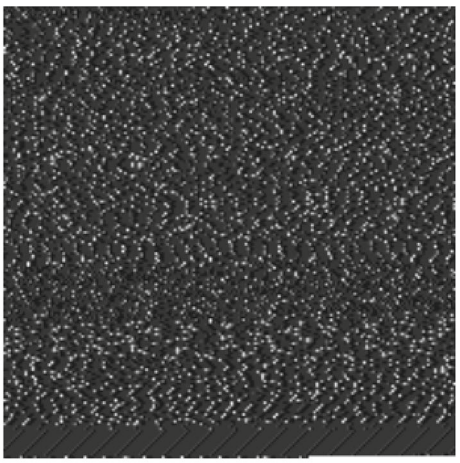
(a) Example of image retrieved from malware *apk* (MD5: 0a30ba12a76469ad0362d801fb82a546).



(b) Example of image retrieved from trusted *apk* (MD5: 1a3e94d252152ef1da92c08ce8290aa0).

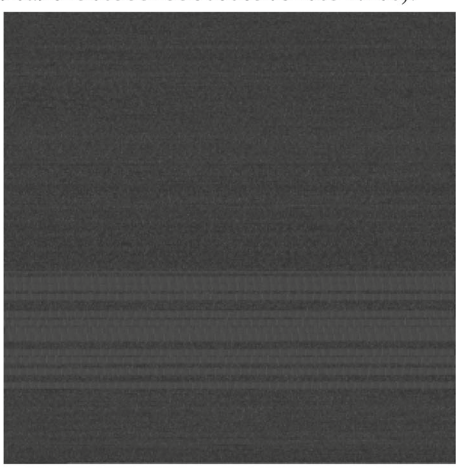
Fig. 2 Example of two different types of images belonging to Malware and Trusted applications in Dataset A

(a) Image representation of a FakeInst malware *APK* sample (MD5: a025a65da6004a83804efdcb4fb74018).



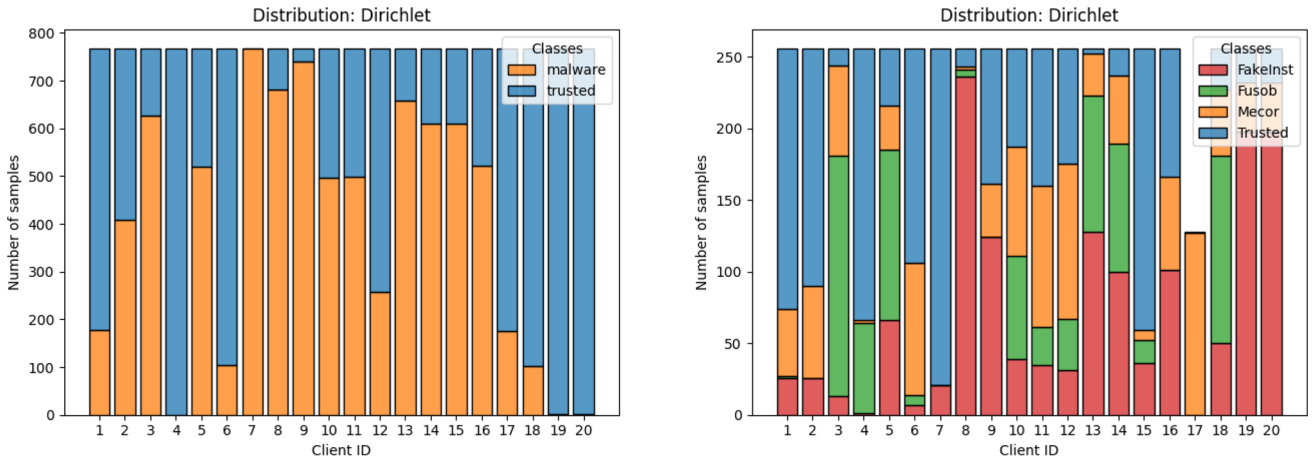
(b) Image representation of a Fusob malware *APK* sample (MD5: 96a98430e533233bc8e850b10e347f0c).

(c) Image representation of a Mecor malware *APK* sample (MD5: f562618897de73ff8d5329e095e59c52).



(d) Image representation of a Trusted *APK* sample (MD5: 715eafc151d7880a45f3b585ee55d05f).

Fig. 3 Example image types for three malware families and trusted applications in Dataset B



(a) Dataset A (b) Dataset B

Fig. 4 Distribution of data among 20 clients modeled using a Dirichlet distribution for both binary and multiclass classification datasets

```
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Input(shape=img_shape),
        MobileNetV3,
        tf.keras.layers.Flatten(),
        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"),
    ],
    name="MobileNetV3-custom",
)
```

Listing 1: A customized MobileNetV3 model with added Dropout and Fully Connected layers.

Google [10]. MobileNetV3 is a Convolutional Neural Network (CNN) designed for mobile phone CPUs, combining and modifying of the hardware-aware Network Architecture Search (NAS) with the NetAdapt algorithm. Among the improvements made by the authors, one notable one is the new efficient network design with the challenging swish activation, in which the swish activation is enhanced by the clipped ReLU instead of the sigmoid. This maneuver reduces computational costs of computations as it makes use of simple thresholding. In this article, we employed a custom version of MobileNet, adding layers such as Dense and Dropout to enhance regularization and mitigate overfitting. These architectures were selected to compare federated performance across models with different complexity levels. The proposed architecture is reported in Listing 1.

Once we identified the architectures and distribution (related to the federated approach), we employed Grid Search to tune the necessary hyperparameters and find the optimal combination that improves the performance of centralized and distributed approaches. Moreover, we employed the ImageNet[13] weights for faster training and better generalization. This transfer learning strategy allows early layers to capture generic visual features, reducing the need for extensive training from scratch. The latter was used for the stated purpose, and some initial image features were acquired in the hidden layers of the model. After we completed the latter phase and thus identified the best hyperparameters inside the framework, we set up the training process using the dataset constructed from different experiments. Regarding the federated approach, to ensure a fair comparison of results, we maintained consistent experimental settings across the adopted datasets, including an image size of 224×3, 20 clients, 17 clients per round, 20 rounds, and 30 client epochs per round. On the other hand, the data distribution across clients followed a Dirichlet distribution with a momentum of 0.99. Additionally, we used a learning rate of 5.7e-5 for Custom MobileNet and 5.9e-5 for InceptionV3. To enhance weight aggregation robustness, we implemented adaptive zeroing of local updates. We utilized an initial L2 norm threshold of 16.0, dynamically adjusted via a norm quantile of 0.75 and a norm increment of 2.0. This mechanism ensures that updates exceeding the adaptive threshold are excluded from the aggregate rather than scaled. The best hyperparameters identified concerning each of the employed architectures are summarized in Table 3.

For the centralized approach, we conducted several experiments using an input image dimension of 224 × 224 × 3,

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

consistent with the federated setting. To ensure a fair comparison, we tested learning rates similar to those used in the federated approach (5.7e-5 and 5.9e-5), as well as additional values (3e-5 and 5e-5). The number of training epochs was set to 5 and 10, and the batch sizes evaluated were 32 and 64. Also in this case, Table 4 reports the best hyperparameters identified using a centralized approach.

4.4 Explainability

After concluding the experimentation phases and identifying the best Federated Machine Learning model, we proceeded to the explainability step by exploiting two Class Activation Mapping algorithms (CAM) algorithms, such as Gradientweighted Class Activation Mapping (Grad-CAM) [29] and Scoreweighted Class Activation Mapping (Score-CAM) [37]. The adopted method was applied to the model with better results to provide a visual explanation behind the model prediction. The Grad-CAM algorithm operates by computing the gradients of the target class score with respect to the feature maps of the last convolutional layer. These gradients are averaged globally to obtain importance weights that reflect the extent to which each feature map influences the predicted class. The weighted combination of these feature maps is then passed through a ReLU activation function, retaining only features that positively contribute to the class of interest. Since convolutional layers preserve spatial information, the resulting activation map can be upsampled and overlaid on the original image, producing a heatmap that visually indicates the areas most influential in the model's decision. On the other hand, the Score-CAM is a related explainability method that does not rely on gradients. Instead of using backpropagated gradients to compute importance weights, Score-CAM perturbs the input image with activation maps obtained from the convolutional layers and measures the change in the model's output score. The increase in the class confidence score determines the importance of each activation map. This gradient-free approach can provide more stable, less noisy visual explanations than gradient-based methods. To quantify the consistency and reliability of the generated heatmaps, we employed the Structural Similarity Index Measure (SSIM) as proposed by Wang *et al.* [38]. This analysis was twofold. First, we calculated the Information Fidelity SSIM (IF-SSIM) to evaluate the internal consistency of each Class Activation Map (CAM) algorithm across the dataset. Second, to assess the robustness of our findings and the degree of consensus between distinct explainability techniques, we calculated the Inter-method SSIM (IM-SSIM). This comparative metric allows for a formal evaluation of whether different attribution methods converge on the same discriminative features.

5 Results

The results obtained during and after the test phase are illustrated in this Section. In detail, Sect. 5.1 shows the outcomes obtained by both Federated Machine Learning models. On the other hand, the results obtained using a centralized method are reported and discussed in Sect. 5.2. After concluding the identification of the best models, we applied them to the two CAM algorithms *i.e.,* Grad-CAM and Score-CAM. Moreover, we used the IF/IM-SSIM algorithm to assess their similarity. These results are reported and discussed in Sections 5.3 and 5.4. The experiments reported in this research article have been executed using the following hardware and software characteristics: Intel Core i7-7700 CPU at 3.60GHz, 32GB RAM, GeForce GTX 1080, and Ubuntu 24.04.1 LTS distro as the Operating System.

5.1 Decentralized models evaluation

5.1.1 Dataset A

As reported in Sections 4.2 and 4.1, once we defined the method to generate images from APK files and obtained datasets, we identified two different CNNs *i.e.,* InceptionV3 and MobileNetV3 Small. We employed the grid search algorithm to determine the best hyperparameters, a widely adopted method that exhaustively evaluates all possible combinations within a predefined parameter grid.

Table 3 reports the hyperparameters that allowed us to reach the best result, while Table 5 displays the outcomes obtained. In the latter, it is possible to denote that the best performance was achieved using Custom MobileNet, despite the accuracy obtained with the model trained on the InceptionV3 architecture being slightly lower. However, the loss values show a significant difference, with Custom MobileNet achieving a loss of 0.259 compared to InceptionV3's much higher loss of 6.707. A high loss is concerning as it indicates that the model is poorly optimizing its predictions, potentially leading to unstable or unreliable outcomes, even if the accuracy appears competitive. High loss often reflects improper learning, overfitting, or difficulty generalizing to new data. Moreover, the model trained using Custom MobileNet achieved precision and recall values of 0.907 and an AUC of 0.965. On the other hand, InceptionV3 reached a precision and recall value of 0.844 and an AUC of 0.891. We also analyzed the training and validation phases to obtain a better comparison, plotting the accuracy and loss achieved during each round. In addition, we also calculated the GAP between the accuracy values obtained during the training and validation phases to understand better the models' ability to generalize. When we concluded the plot generation, we determined that the model trained using MobileNet architec-

Table 3 Hyperparamters adopted to train Federated Machine Learning models over both datasets employed

Architecture	Image Size	TC	CR	TR	CER	Algorithm	Optimizer	Data Distribution	Identicalness	Learning rate	Weights
InceptionV3	224x3	20	17	20	30	FedAvg	SGDm	non-IID	0.8	5.7e-05	imagenet
Custom MobileNet	224x3	20	17	20	30	FedAvg	SGDm	non-IID	0.8	5.9e-05	imagenet

Table 4 Hyperparamters adopted to train centralized models

Dataset	Model	Image Size	Learning Rate	Epochs	Batch Size
Dataset A	InceptionV3	224x3	5e-05	10	32
	Custom MobileNet	224x3	5.7e-05	10	64
Dataset B	InceptionV3	224x3	3e-05	5	64
	Custom MobileNet	224x3	5e-05	5	64

Table 5 Results obtained on the test sets using the Federated Machine Learning approach for both Dataset A and Dataset B

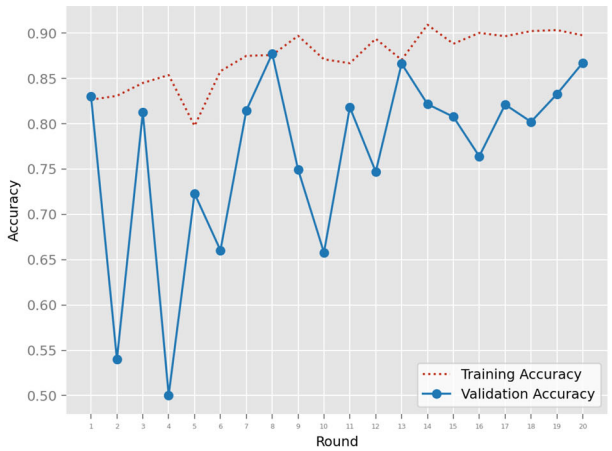
Dataset	Model	Loss	Accuracy	Precision	Recall	F-Measure	AUC
Dataset A	InceptionV3	6.707	0.884	0.884	0.844	0.844	0.891
	Custom MobileNet	0.259	0.908	0.908	0.908	0.908	0.965
Dataset B	InceptionV3	19.225	0.918	0.918	0.918	0.918	0.961
	Custom MobileNet	0.116	0.962	0.966	0.962	0.964	0.995

ture performed better than that trained with the InceptionV3 architecture.

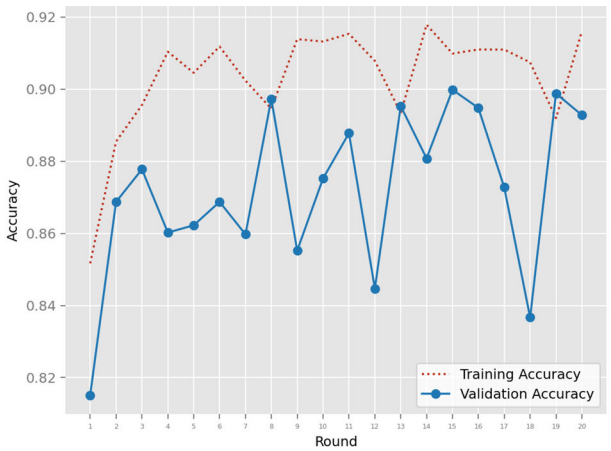
Figure 5 shows the accuracy trends obtained from both models; in detail, Fig. 5a displays the trend achieved by the model trained using InceptionV3, while Fig. 5b reports the tendency obtained during the model's training using Custom MobileNet.

Similarly, Fig. 6 reports the loss obtained during the training and validation phases, where Fig. 6a shows the loss trend for the InceptionV3 architecture, while Fig. 6b displays the loss for the Custom MobileNet model. In these figures, the red line shows the accuracy or loss values recorded during training, while the blue line depicts the accuracy or loss achieved during validation at each epoch.

The GAP comparison for both models is reported in Fig. 7. In detail, Fig. 7a represents the GAP for the InceptionV3 architecture, and Fig. 7b shows the GAP for the Custom MobileNet model. After completing the training and validation phases, we evaluated the models on the test set and generated confusion matrices to assess their classification performance. Figure 8a reports the confusion matrix obtained after testing the model trained using InceptionV3 architec-

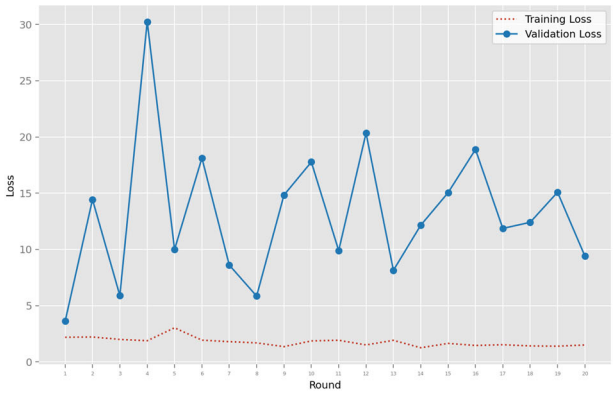


(a) Accuracy achieved in each epoch during the training and validation phases using the InceptionV3 architecture

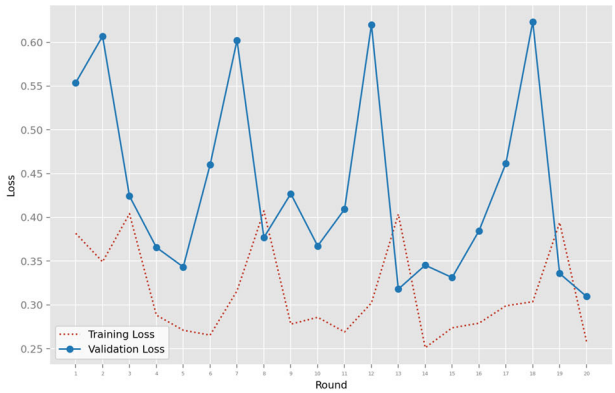


(b) Accuracy achieved in each epoch during the training and validation phases using the Custom MobileNet architecture

Fig. 5 Accuracy comparison of models during training and validation: InceptionV3 vs. Custom MobileNet

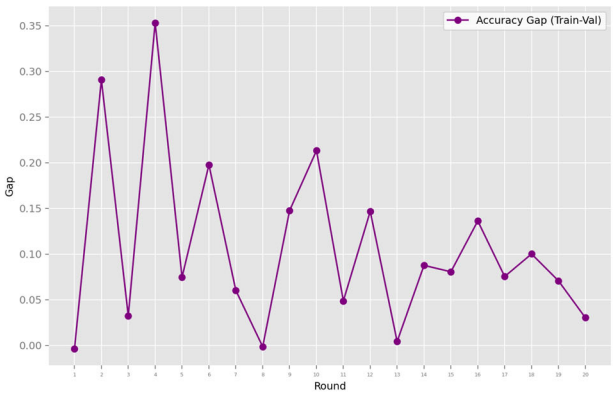


(a) Loss achieved in each epoch during the training and validation phases using the InceptionV3 architecture

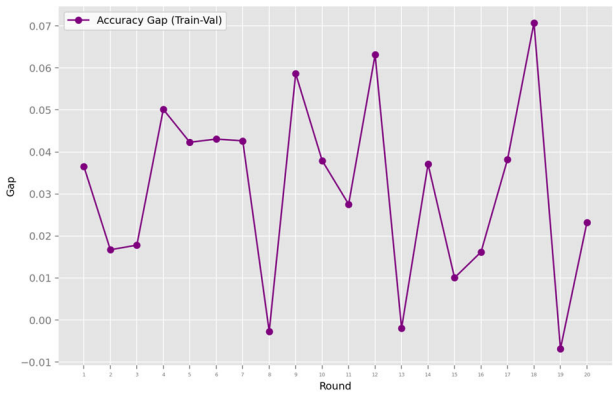


(b) Loss achieved in each epoch during the training and validation phases using the Custom MobileNet architecture

Fig. 6 Loss comparison of models during training and validation: InceptionV3 vs. Custom MobileNet



(a) Gap achieved in each epoch during the training and validation phases using the InceptionV3 architecture



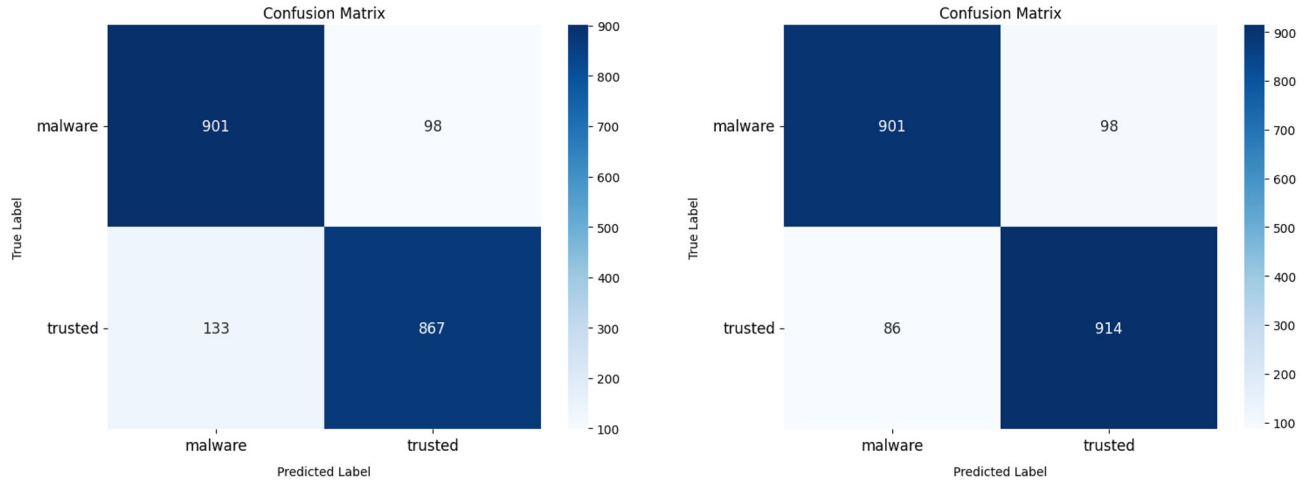
(b) Gap achieved in each epoch during the training and validation phases using the Custom MobileNet architecture

Fig. 7 Gap comparison of models during training and validation: InceptionV3 vs. Custom MobileNet

ture. In detail, the model correctly identified 901 instances of malware and 867 instances of trusted while misclassifying 98 malware instances as trusted and 133 trusted instances as malware. Regarding the metrics for each class, the models achieved a precision value of 0.871 for malware and 0.898 for trusted, indicating a higher accuracy when predicting trusted instances. Regarding the recall, it reached a value of 0.901 in the malware class and 0.867 for the trusted class. On the other hand, the confusion matrix obtained from the model trained using Custom MobileNet is reported in Fig. 8b . In detail, it is possible to denote that for the malware class, the model correctly identified 901 instances as malware but misclassified 98 instances as trusted. For the trusted class, the model correctly predicted 914 instances as trusted, while 86 instances were incorrectly classified as malware. In this case, the precision reached for the malware class increased to 0.912 and 0.903 for the trusted class. Also, the recall increased for both classes, reaching 0.901 for malware and 0.914 for trusted.

5.1.2 Dataset B

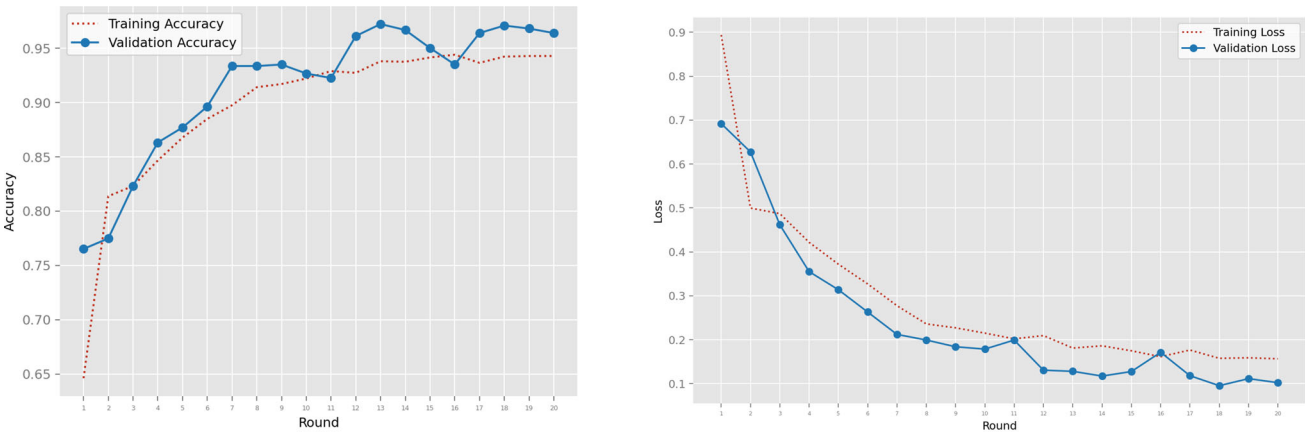
After concluding the experiment phase using the binary dataset and identifying the best hyperparameters, we employed them to train other models, always using the same architectures, to confirm the goodness of the proposed method. In this case, we achieved interesting results, and the custom version of MobileNet overperformed, achieving the best results. Moreover, to further assess potential overfitting and other training-related issues, we analyzed the trends of accuracy and loss across both the training and validation phases. In addition, we computed the generalization GAP between training and validation accuracies to quantitatively assess performance discrepancies and better characterize the model's ability to generalize beyond the training data. The accuracy trend is reported in Fig. 9a . In the latter, it is evident that both training and validation accuracy increase steadily and plateau after approximately round 12, reaching values above 0.940. Another important observation is that the val-



(a) Confusion matrix obtained after the testing phase of InceptionV3.

(b) Confusion matrix obtained after the testing phase of Custom MobileNet.

Fig. 8 Confusion Matrices comparison of models obtained during test phase: InceptionV3 vs. Custom MobileNet



(a) Accuracy trend over training and validation phases.

(b) Loss trend over training and validation phases.

Fig. 9 Training and validation accuracy and loss trends for the best-performing model trained on Dataset B

idation accuracy slightly exceeds the training accuracy in several rounds, reinforcing the absence of overfitting and suggesting effective regularization. Figure 9b illustrates the evolution of training and validation loss across 20 training rounds. Both curves exhibit a rapid decline during the initial epochs, followed by a gradual stabilization (particularly for the training trend), indicating effective convergence of the optimization process. Notably, validation loss closely follows the training loss throughout training, with only minor oscillations during rounds 11 and 16, when they achieve the same values as during training. The absence of divergence between the two curves suggests that the model does not suffer from significant overfitting.

Concluding, we analyzed the GAP obtained to confirm the absence of overfitting and the goodness of the model. The generalization GAP is reported in Fig. 10; it is evident that after an initial transient phase in the first two rounds, the gap remains consistently close to zero, oscillating within a narrow range (approximately ± 0.03). This limited yet stable gap further confirms strong generalization and indicates that performance on unseen data closely matches that on the training data.

After completing the training and validation phases, we evaluated the model on the test set, achieving improved performance in both accuracy and AUC. As reported in Table 5, the model trained with the Custom MobileNet architecture achieved a global accuracy of 0.961 and an AUC of 0.995, demonstrating strong discriminative capability and excellent class separability across all categories. These results confirm the robustness of the proposed approach and its

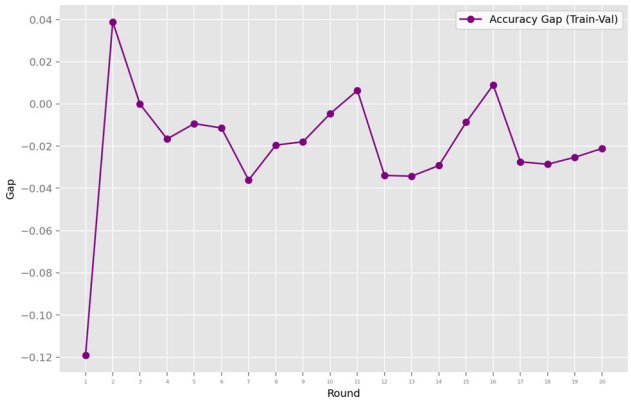


Fig. 10 Generalization gap across training rounds for the best model on Dataset B

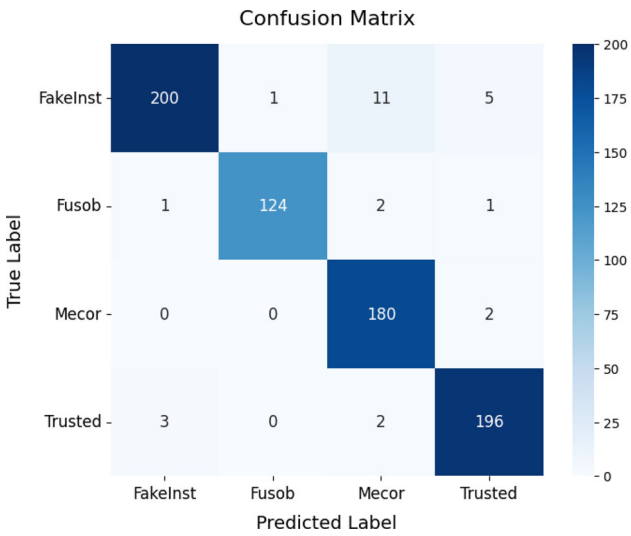


Fig. 11 Confusion matrix of the best-performing model (Custom MobileNet) on Dataset B

effectiveness in correctly distinguishing between the different classes employing a distributed approach. To provide a deeper understanding of the model's behavior across the four classes (FakeInst, Fusob, Mecor, and Trusted), Fig. 11 shows the confusion matrix. Moreover, we also reported the class-wise performance metrics in Table 6. The confusion matrix shows that the model correctly identified 200 samples from the FakeInst fam. In contrast,ile was incorrectly identified as Fusob (1), Mecor (11), and Trusted (5) 17 times. Regarding the other classes, the model correctly identified 124, 180, and 196 samples as Fusob, Mecor, and Trusted, respectively, while misclassifying 4 Fusob samples, 2 Mecor samples, and 5 Trusted samples.

5.2 Centralized models evaluation

After concluding the experimentation using a federated approach, we replicated the experiments using a centralized approach. Specifically, to ensure consistency and fairness in the comparison, we used the same CNNs as in the federated experiments: InceptionV3 and a Custom MobileNet-based CNN architecture. These models were trained and evaluated on both the binary and multi-class datasets generated for this study. The hyperparameters adopted are reported in Table 4, while the results obtained are reported in Table 7.

Table 6 Class-wise evaluation of the best-performing model (Custom MobileNet) on Dataset B, reporting accuracy, precision, recall, F-measure, and AUC for each class to assess classification performance

Class	Accuracy	Precision	Recall	F1-Score	AUC
FakeInst	0.971	0.980	0.922	0.950	0.957
Fusob	0.993	0.992	0.969	0.980	0.984
Mecor	0.977	0.923	0.989	0.955	0.981
Trusted	0.982	0.961	0.975	0.968	0.980

The results demonstrate that the centralized framework achieved higher performance using the InceptionV3 architecture, which reached an accuracy of 0.946 and a loss of 0.458 in binary classification. The confusion matrix obtained during the test phase is shown in Figure 12. From this matrix, it can be observed that the model effectively distinguishes between the two classes, namely malware and trusted samples. In particular, the model correctly identified 957 malware instances, while misclassifying 24 malware samples as trusted. Conversely, trusted samples were correctly classified in 916 cases, whereas 84 instances were incorrectly labeled as malware.

In contrast, the custom MobileNet model exhibited a notable performance degradation in the centralized configuration, yielding a marginal accuracy of 0.511. In the multi-class experiments, InceptionV3 again reported the highest accuracy among the candidate models. However, a granular analysis of the performance metrics revealed a significant failure in model convergence; specifically, precision, recall, and F-measure scores collapsed to zero during testing. This discrepancy indicates that the model likely defaulted to a dominant class, failing to differentiate between specific categories despite nominal accuracy. The relatively high AUC (0.841) alongside null precision and recall further underscores a lack of generalization and potential overfitting. The custom MobileNet architecture faced similar challenges in the multi-class environment, achieving lower accuracy (0.271) and a higher loss (2.137), both substantially higher than those of InceptionV3 (1.321). These findings suggest that both architectures struggle to maintain stability when transitioning from binary tasks to more complex multi-class datasets.

Table 7 Results obtained on the test sets using the centralized approach for both Dataset A and Dataset B

Dataset	Model	Loss	Accuracy	Precision	Recall	F-Measure	AUC
Dataset A	InceptionV3	0.458	0.946	0.946	0.946	0.946	0.966
	Custom MobileNet	0.800	0.511	0.511	0.511	0.511	0.353
Dataset B	InceptionV3	1.321	0.751	0	0	0	0.841
	Custom MobileNet	2.137	0.276	0.276	0.276	0.276	0.447

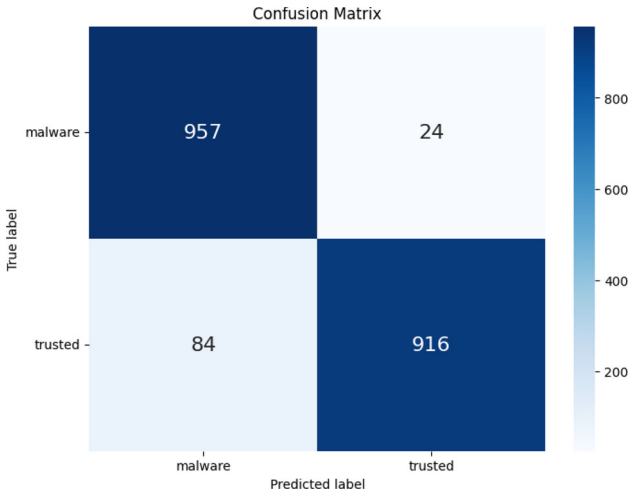


Fig. 12 Confusion matrix obtained during the test phase of the InceptionV3 model on Dataset A

5.3 CAM algorithms application

After completing the training and test phases, we used two CAM algorithms, namely: Gradient-weighted Class Activation Mapping and Score-weighted Class Activation Mapping to assess explainability. In detail, to do that, we used the model that achieved the best performance in terms of accuracy and loss as discussed in Sect. 5. Specifically, we adopted the model trained with the Custom MobileNet architecture and the binary dataset. To provide the most satisfactory possible explanation in this Section, we will discuss examples of the image generated by overlaying the initial PNG image with the heatmap obtained. Moreover, we also provide a VirusTotal link to confirm the category to which it belongs.

Figures 13 and 14 show the application of the Grad-CAM and Score-CAM algorithms on samples belonging to the malware class. Figures 13 shows model recognized by the model as malware with an accuracy percentage of almost 100 (99.88%). Comparing the heatmaps obtained, it appears that the method based on the Score-CAM was focused on multiple areas compared to the Grad-CAM. On the other hand, the latter placed more importance on a small portion of the image, one on the right and one on the left. The result also confirmed the VirusTotal, where 42 antimalware out of 66 recognize the samples as malware[14]. Also, the malware sample shown in Fig. 14 was classified by the model as a malicious file with a perfect confidence score of 100%. Similarly to the previous image, in this case, both algorithms highlight similar regions. However, Score-CAM places greater emphasis on the lower part of the image and extends its attention toward the upper areas as well. Moreover, VirusTotal corroborated this assessment, with 45 out of 66 detection engines flagging the APK file as harmful[15]. It is possible to observe in the heatmap that three distinct colors *i.e.,* yellow, green, and blue, highlight areas of the image that are significant from the model's prediction perspective. While the blue hue emphasizes regions that do not influence the overall pattern, the yellow areas represent the parts of the image that the model is most focused on. Finally, the green hue marks the intermediate regions in the image. Figures 15 and 16 illustrate the prediction outcomes for trusted samples, which were classified as trustworthy with accuracies of 99.89% and 99.51%, respectively. As observed with the malware samples, Grad-CAM also highlighted finer regions than Score-CAM for the trusted samples. However, unlike the malware case, the two algorithms did not indicate the same areas. Moreover, the first sample was recognized as a trusted file by 62 out of 63 antimalware on VirusTotal[16], while in the second case, all detectors identified the sample as trusted[17].

As shown in Fig. 8b , which presents the confusion matrix from the test phase of the model trained with the Custom MobileNet architecture, the model incorrectly classifies malware as trusted 98 times and mistakenly labels trusted instances as malware 86 times. Figures 17 and 18 report two wrong classifications. In detail, regarding the sample reported in Fig. 17 identified with the hash MD5 c35ef5d5bf c82ba89e5d568ed52f715e, only 35 antimalware on 65 recognize it as a malicious application[18]. On the other hand, Fig.

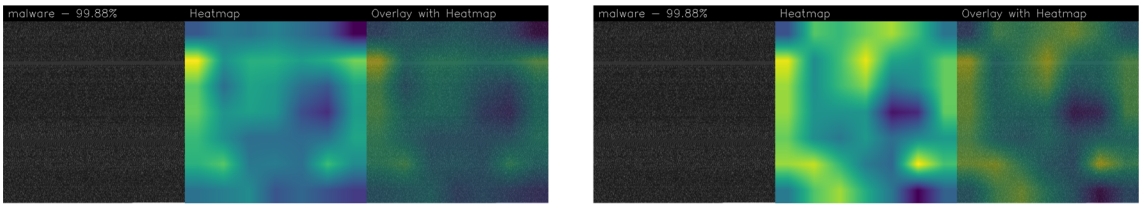
[14] https://www.virustotal.com/gui/file/4ba887cc4ac08496dff94b6c6a6f2d30

[15] https://www.virustotal.com/gui/file/4e577464f7e8e87f6be2433ba30f8381

[16] https://www.virustotal.com/gui/file/532bd176cb19ee3a6c3ebde7875e8e95

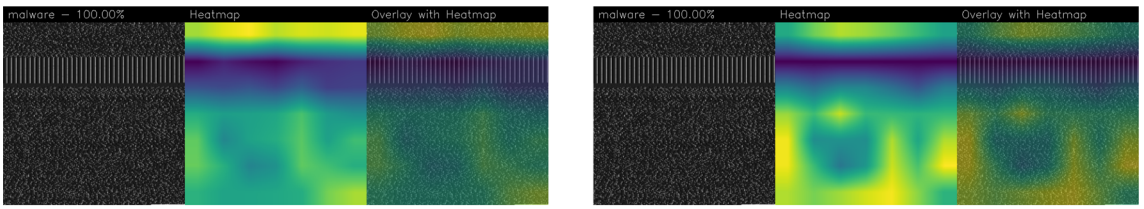
[17] https://www.virustotal.com/gui/file/c66972629cbc0001ec2a95048bc7c786

[18] https://www.virustotal.com/gui/file/c35ef5d5bfc82ba89e5d568ed52f715e



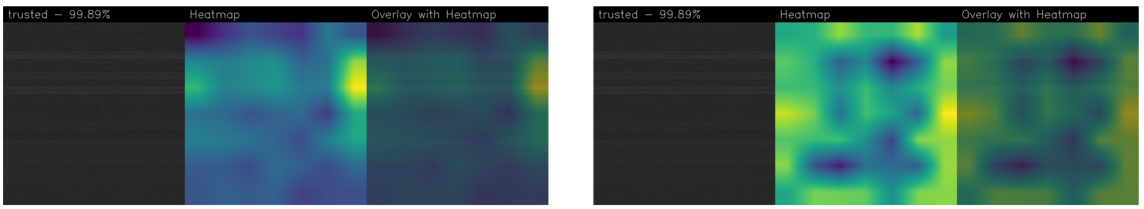
(a) Grad-CAM (b) Score-CAM

Fig. 13 Result of applying CAM algorithms on malware sample (MD5: 4ba887cc4ac08496dff94b6c6a6f2d30) with an high accuracy value (99.88%)



(a) Grad-CAM (b) Score-CAM

Fig. 14 Result of applying CAM algorithms on malware sample (MD5: 4e577464f7e8e87f6be2433ba30f8381) with perfect accuracy value of 100%



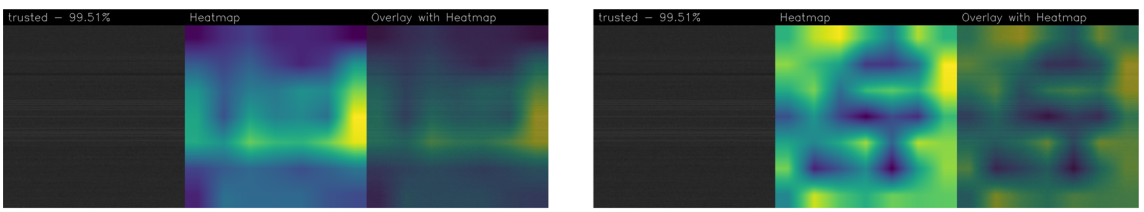
(a) Grad-CAM (b) Score-CAM

Fig. 15 Result of applying CAM algorithms on trusted sample (MD5: 532bd176cb19ee3a6c3ebde7875e8e95) with an accuracy value of almost 100%

18 reports a trusted sample wrongly recognized as malware. One possible reason for this misclassification relies on the areas identified, *i.e.,* on the left and right sides. This happens similarly in most cases regarding malware samples, as shown in Fig. 13. Differently from the cases reported above, in Fig. 17 the Score-CAM produced a more interpretable heatmap with reduced noise, allowing a clearer identification of the regions that most influenced the model's decision. This suggests a more meaningful localization of the relevant features, even though the final classification was incorrect.

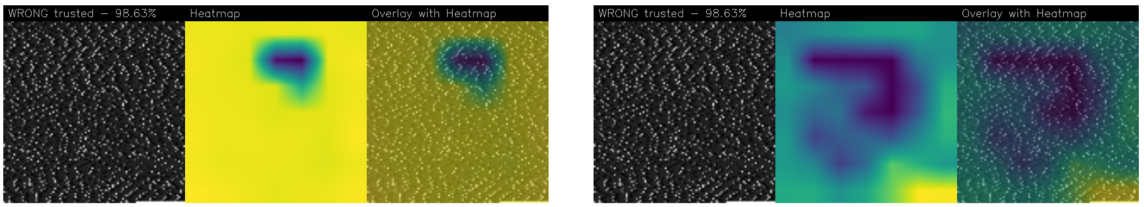
5.4 Similarity

After applying CAM algorithms, we used the Structural Similarity Index (SSIM) to evaluate the stability of the model's explanations. The results reveal a significant disparity in explanation reliability and cross-method agreement between classes. Intra-method SSIM (IF-SSIM) indicates that both Grad-CAM and Score-CAM generate highly consistent attribution maps for the trusted class, yielding scores of 0.814 and 0.789, respectively, which suggests the presence of stable



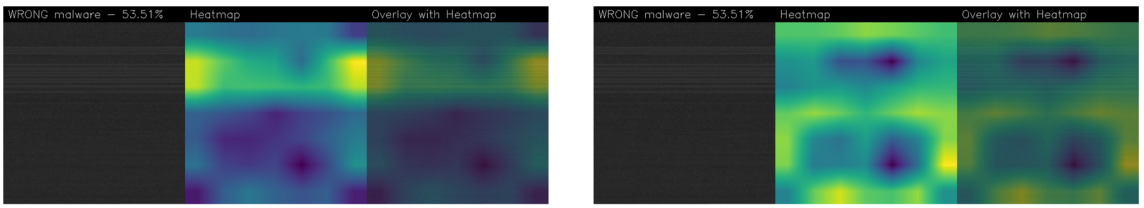
(a) Grad-CAM (b) Score-CAM

Fig. 16 Result of applying CAM algorithms on trusted sample (MD5: c66972629cbc0001ec2a95048bc7c786) with a good accuracy (99.51%)



(a) Grad-CAM (b) Score-CAM

Fig. 17 Result of applying CAM algorithms on malware sample (MD5: c35ef5d5bfc82ba89e5d568ed52f715e) wrongly recognized as trusted



(a) Grad-CAM (b) Score-CAM

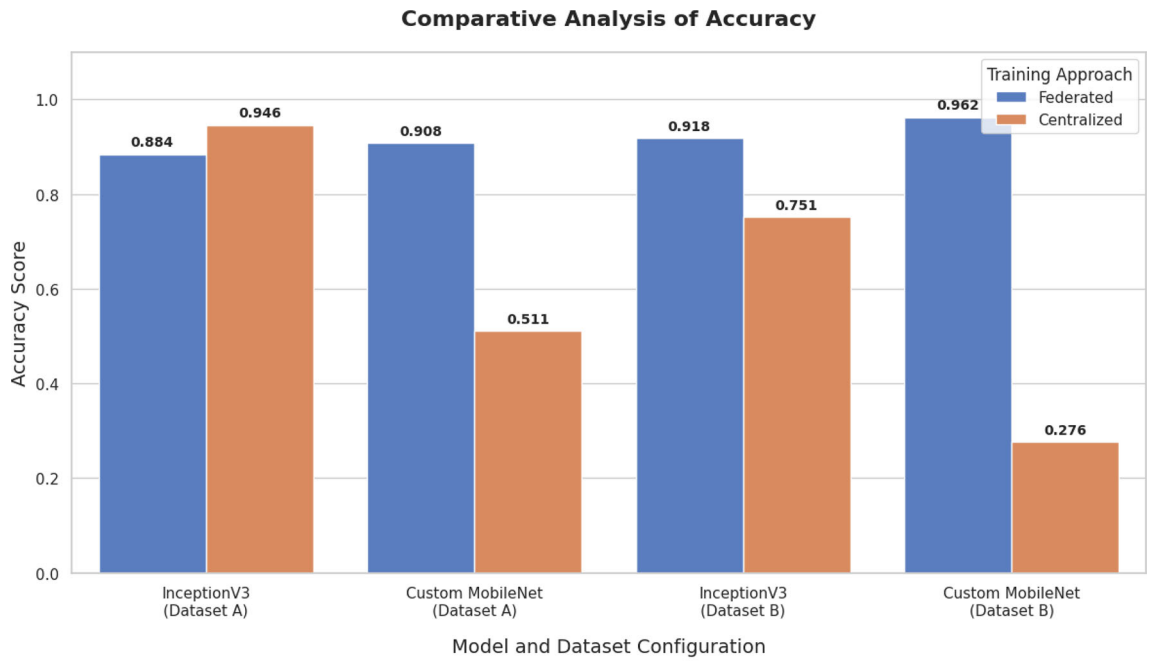
Fig. 18 Result of applying CAM algorithms on ttrusted sample (MD5: 177638da64f79d86511861fde71f8ccc) wrongly recognized as trusted with small accuracy

internal reasoning patterns. However, this consistency diminishes when examining the malware class; here, Grad-CAM demonstrates substantially lower stability (0.531) compared to Score-CAM (0.733). This discrepancy suggests that Grad-CAM explanations for malware exhibit greater variability across samples, potentially complicating the interpretation of malicious triggers. The Inter-method SSIM (IM-SSIM) results further highlight a pronounced class-dependent divergence in explanation logic. While Grad-CAM and Score-CAM demonstrate strong convergence for malware samples (0.795), they exhibit a near-total lack of similarity for trusted samples (0.0009). This contrast indicates that malware classification likely relies on well-defined, method-invariant discriminative features that allow different attribution techniques to reach a consensus. In contrast, trusted predictions appear to depend on more diffuse or heterogeneous patterns, making the resulting explanations highly sensitive to the specific mechanics of the chosen attribution method. Collectively, these findings underscore that explanation robustness is itself class-dependent, emphasizing the critical necessity of cross-method validation in high-stakes explainability studies.

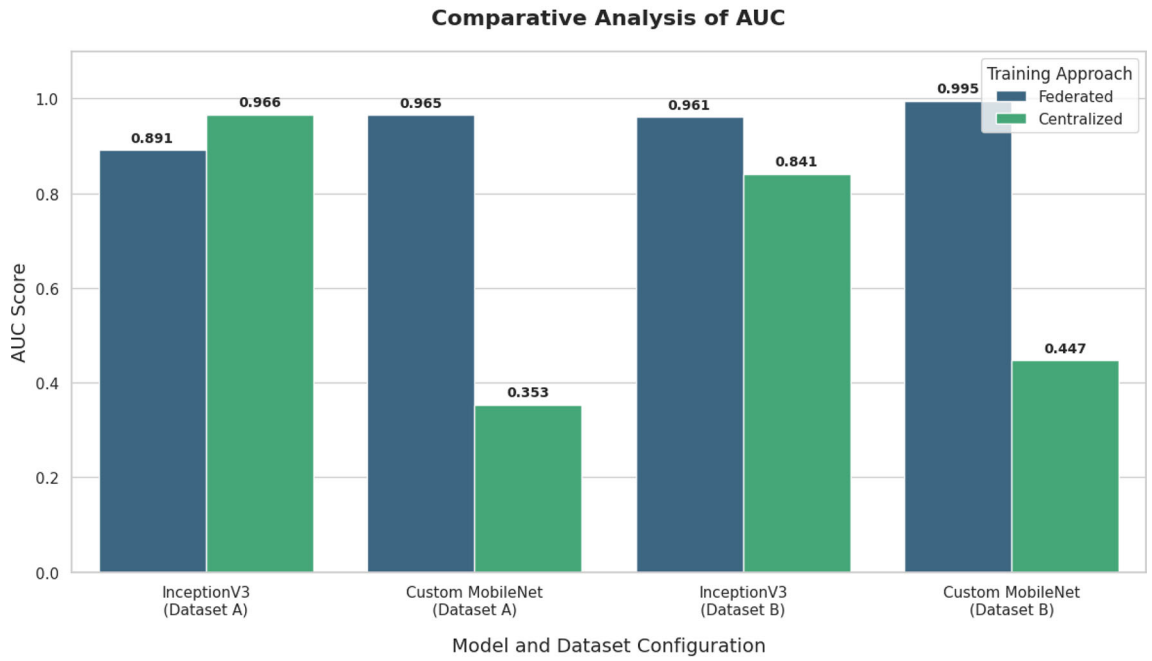
5.5 Discussion

Results achieved using the decentralized and centralized approaches were completely different, particularly concerning the Custom MobileNet architecture. Figure 5 and 19b report the accuracy and AUC values obtained after testing each architecture on two different datasets and approaches.

The results obtained with Dataset A, *i.e.,* the dataset used for binary classification, showed interesting accuracy and AUC values for both approaches, even with two different architectures. Specifically, in the distributed setting, the Custom MobileNet achieved a good accuracy of 0.908 and an AUC of 0.965. Thus, the model demonstrated strong capabilities and robustness in performing binary discrimination under decentralized training conditions. On the other hand, in the binary classification task, the best accuracy was achieved with the InceptionV3 using a centralized approach. The latter demonstrated high effectiveness at extracting discriminative patterns when trained on aggregated data. Specifically, this architecture achieved an accuracy value of 0.946 and an AUC of 0.966, indicating strong generalization performance on the test set. Moreover, in the centralized setting, Custom MobileNet achieved an accuracy of 0.511, close to random guessing in a binary classification scenario, suggesting limited learning effectiveness under this configuration. Regarding the results achieved with Dataset B, it introduced severe levels of difficulty. The most striking finding is the metric collapse observed in the centralized InceptionV3 model. Despite a nominal accuracy of 0.751 and a respectable AUC of 0.841, precision and recall fell to 0. The Federated approach, however, demonstrated a remarkable capacity to mitigate this failure. In the federated setting, InceptionV3 and Custom MobileNet achieved accuracies of 0.918 and 0.962, respectively, with precision and recall scores closely matching. This success suggests that federated learning may inherently address class imbalance.



(a) Accuracy performance metrics.



(b) AUC performance metrics.

Fig. 19 Comparison of Accuracy and AUC performance metrics achieved by the Federated (Distributed) and Centralized learning approaches across the evaluated datasets and models

6 Conclusion and future work

In this paper, we proposed a malware detection method leveraging Federated Machine Learning. Specifically, we employed two datasets of Android applications converted into images using a state-of-the-art approach, transforming the smali code into ASCII characters and then into gray-scale images. This approach was adopted on a first dataset named "Dataset A" composed of two classes, and a second dataset named "Dataset B" consisting of four classes, such as FakeInst, Fusob, Mecor, and Trusted. After concluding the preparatory phase, we exploited two Convolutional Neural Networks, such as InceptionV3 and a custom version of MobileNet, to train and test decentralized and centralized models. To do that, we employed the grid search algorithm to identify the best hyperparameters. The obtained results allowed us to identify the model trained using the Custom MobileNet network model as the best model on the binary dataset, and we applied two CAM algorithms to perform explainability (*i.e.,* Grad-CAM and Score-CAM). Finally, the Structural Similarity Index Measure was employed to objectively quantify the fidelity and spatial consistency of the generated heatmaps, ensuring the reliability of the visual explanations. To the best of our knowledge, this article is the first to incorporate explainability into image-based malware detection within Federated Machine Learning using images retrieved from smali code.

In future studies, we plan to perform a deeper analysis to extract the smali code from the highlighted areas. Using this approach, it is possible to better understand the specific code segments the model identifies as significant for distinguishing between malware and trusted applications. Moreover, we intend to use other CAM algorithms and compare the outcomes to provide a better, robust, and comprehensive evaluation of the model's interpretability. In conclusion, another essential step will be to apply CAM algorithms individually on each device in Federated Machine Learning to investigate how the model's learning process evolves across different devices.

Acknowledgements This work has been partially supported by EU DUCA, EU CyberSecPro, SYNAPSE, PTR 25-27 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.

Author Contributions Conceptualization: G.C., F.M. (Fabio Martinelli), A.S. and F.M. (Francesco Mercaldo) Methodology: G.C. and F.M. (Francesco Mercaldo) Software: G.C. and F.M. (Francesco Mercaldo) Validation: G.C., F.M. (Francesco Mercaldo) and A.S. Investigation: G.C. and F.M. (Francesco Mercaldo) writing—original draft preparation: G.C., F.M. (Francesco Mercaldo) writing—review and editing: G.C., F.M. (Fabio Martinelli), A.S., and F.M. (Francesco Mercaldo) supervision: F.M. (Fabio Martinelli), A.S. funding acquisition: F.M. (Fabio Martinelli) All authors have read and agreed to the published version of the manuscript.

Funding Open access funding provided by Scuola IMT Alti Studi Lucca within the CRUI-CARE Agreement.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Bharati, S., Mondal, M., Podder, P., Prasath, V.: Federated learning: applications, challenges and future directions. Int. J. Hybrid Intell. Syst. **18**, 19–35 (2022)
2. Canfora, G., Medvet, E., Mercaldo, F., Visaggio, C.A.: Detecting android malware using sequences of system calls. In: Proceedings of the 3rd international workshop on software development lifecycle for mobile, pp. 13–20 (2015)
3. Ciaramella, G., Martinelli, F., Mercaldo, F., Mori, P., Santone, A.: A federated machine learning method for malicious smart contract detection. In: 2025 7th International Conference on Blockchain Computing and Applications (BCCA), IEEE. pp. 459–466 (2025a)
4. Ciaramella, G., Martinelli, F., Peluso, C., Santone, A., Mercaldo, F.: A method for real-world privacy-preserving android malware detection through federated machine learning. Inform. Software Technol. 107892 (2025b)
5. Dhade, P., Shirke, P.: Federated learning for healthcare: a comprehensive review. Eng Proceedings **59**, 230 (2024)
6. Du, Z., Wu, C., Yoshinaga, T., Yau, K.L.A., Ji, Y., Li, J.: Federated learning for vehicular internet of things: recent advances and open issues. IEEE Open J. Comput. Soc. **1**, 45–61 (2020)
7. Fang, Y., Gao, Y., Jing, F., Zhang, L.: Android malware familial classification based on dex file section features. IEEE Access **8**, 10614–10627 (2020)

8. Gálvez, R., Moonsamy, V., Diaz, C.: Less is more: a privacy-respecting android malware classifier using federated learning. arXiv preprint arXiv:2007.08319 (2020)
9. Hemalatha, J., Roseline, S.A., Geetha, S., Kadry, S., Damaševičius, R.: An efficient densenet-based deep learning model for malware detection. Entropy **23** (2021). https://www.mdpi.com/1099-4300/23/3/344, https://doi.org/10.3390/e23030344
10. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE/CVF international conference on computer vision, pp. 1314–1324 (2019)
11. Iadarola, G., Casolare, R., Martinelli, F., Mercaldo, F., Peluso, C., Santone, A.: A semi-automated explainability-driven approach for malware analysis through deep learning. In: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE. pp. 1–8 (2021)
12. İbrahim, M., Issa, B., Jasser, M.B.: A method for automatic android malware detection based on static analysis and deep learning. IEEE Access **10**, 117334–117352 (2022)
13. Jiang, C., Yin, K., Xia, C., Huang, W.: Fedhgcdroid: an adaptive multi-dimensional federated learning for privacy-preserving android malware classification. Entropy **24**, 919 (2022)
14. Jiang, T., Shen, G., Guo, C., Cui, Y., Xie, B.: Bfls: blockchain and federated learning for sharing threat detection models as cyber threat intelligence. Comput. Netw. **224**, 109604 (2023)
15. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. Foundations Trends® Mach. Learn. **14**, 1–210 (2021)
16. Li, Q., Diao, Y., Chen, Q., He, B.: Federated learning on non-iid data silos: an experimental study. In: 2022 IEEE 38th international conference on data engineering (ICDE), IEEE. pp. 965–978 (2022)
17. Li, Y., Jang, J., Hu, X., Ou, X.: Android malware clustering through malicious payload mining. In: Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18–20, 2017, Proceedings, Springer. pp. 192–214 (2017)
18. Lim, W.Y.B., Luong, N.C., Hoang, D.T., Jiao, Y., Liang, Y.C., Yang, Q., Niyato, D., Miao, C.: Federated learning in mobile edge networks: a comprehensive survey. IEEE Commun. Surveys Tutorials **22**, 2031–2063 (2020)
19. Lin, K.Y., Huang, W.R.: Using federated learning on malware classification. In: 2020 22nd international conference on advanced communication technology (ICACT), IEEE. pp. 585–589 (2020)
20. Ma, X., Zhu, J., Lin, Z., Chen, S., Qin, Y.: A state-of-the-art survey on solving non-iid data in federated learning. Futur. Gener. Comput. Syst. **135**, 244–258 (2022)
21. Mayhoub, S., Shami, M.T.: A review of client selection methods in federated learning. Archives Computat. Methods Eng. **31**, 1129–1152 (2024)
22. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, PMLR. pp. 1273–1282 (2017)
23. Mercaldo, F., Ciaramella, G., Santone, A., Martinelli, F.: 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, pp. 1–10 (2023)
24. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: 2019 IEEE symposium on security and privacy (SP), IEEE. pp. 739–753 (2019)
25. Nguyen, D.C., Ding, M., Pathirana, P.N., Seneviratne, A., Li, J., Poor, H.V.: Federated learning for internet of things: a comprehensive survey. IEEE Commun. Surveys Tutorials **23**, 1622–1658 (2021)
26. Niro, F., Di Renzo, M., Agnello, P., Petyx, M., Ciaramella, G., Martinelli, F., Cesarelli, M., Santone, A., Mercaldo, F.: A privacy-preserving method for explainable multiple sclerosis detection through federated machine learning. In: International Conference on Image Analysis and Processing, Springer. pp. 29–40 (2025)
27. Rathore, H., Agarwal, S., Sahay, S.K., Sewak, M.: 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, Springer. pp. 402–411 (2018)
28. Rey, V., Sánchez, P.M.S., Celdrán, A.H., Bovet, G.: Federated learning for malware detection in iot devices. Comput. Netw. **204**, 108693 (2022)
29. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE international conference on computer vision, pp. 618–626 (2017)
30. Sewak, M., Sahay, S.K., Rathore, H.: Comparison of deep learning and the classical machine learning algorithm for the malware detection. In: 2018 19th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD), IEEE. pp. 293–296 (2018)
31. Sharma, Y.K., Tomar, D.S., Pateriya, R., Bhandari, S.: Mosdroid: obfuscation-resilient android malware detection using multisets of encoded opcode sequences. Comput. Security **152**, 104379 (2025)
32. Shen, S., Zhu, T., Wu, D., Wang, W., Zhou, W.: From distributed machine learning to federated learning: in the view of data privacy and security. Concurrency and Computation: Practice and Experience **34**, e6002 (2022)
33. Sun, G., Cong, Y., Dong, J., Wang, Q., Lyu, L., Liu, J.: Data poisoning attacks on federated machine learning. IEEE Internet Things J. **9**, 11365–11375 (2021)
34. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826 (2016)
35. Taheri, R., Shojafar, M., Alazab, M., Tafazolli, R.: Fed-iiot: a robust federated malware detection architecture in industrial iot. IEEE Trans. Industr. Inf. **17**, 8442–8452 (2020)
36. Tang, M., Qian, Q.: Dynamic api call sequence visualisation for malware classification. IET Inf. Secur. **13**, 367–377 (2019)
37. Wang, H., Wang, Z., Du, M., Yang, F., Zhang, Z., Ding, S., Mardziel, P., Hu, X.: Score-cam: score-weighted visual explanations for convolutional neural networks. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, pp. 24–25 (2020)
38. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**, 600–612 (2004)
39. Yadav, R., Kumaran, U., Meena, V., Wei, L., Feng, G., Elaziz, M.A.: Federated deep learning for malware detection and data protection in edge-enabled iomt. Clust. Comput. **28**, 757 (2025)
40. Yapici, M.M.: A novel image based approach for mobile android malware detection and classification. Knowl.-Based Syst. **323**, 113855 (2025)
41. Zhan, Y., Zhang, J., Hong, Z., Wu, L., Li, P., Guo, S.: A survey of incentive mechanism design for federated learning. IEEE Trans. Emerg. Top. Comput. **10**, 1035–1044 (2021)
42. Zhu, H., Xu, J., Liu, S., Jin, Y.: Federated learning on non-iid data: a survey. Neurocomputing **465**, 371–390 (2021)