M2FD: Mobile malware federated detection under concept drift

A. Augello, A. De Paola, G. Lo Re

Computers & Security

DOI: https://doi.org/10.1016/j.cose.2025.104361

M2FD: Mobile Malware Federated Detection Under Concept Drift

Andrea Augello* Alessandra De Paola Giuseppe Lo Re

Department of Engineering, University of Palermo, Italy

Abstract

The ubiquitous diffusion of mobile devices requires the availability of effective malware detection solutions to ensure user security and privacy. The dynamic nature of the mobile ecosystem, characterized by data distribution changes, poses significant challenges to the development of effective malware detection systems. Additionally, collecting up-to-date information for training machine learning models in a centralized fashion is costly, time-consuming, and privacy-invasive. To address these shortcomings, this paper presents a novel federated learning system for collaborative mobile malware detection. M2FD leverages the collective intelligence of the user community to collect valuable contributions to the detection system while preserving user privacy. Additionally, M2FD incorporates robust concept drift detection mechanisms and model retraining strategies to ensure the adaptability of the system to changing data distributions. By effectively handling concept drift, M2FD guarantees a high ability to detect malware, with 85% accuracy and 84% F1-score, even in presence of evolving attack strategies, thus avoiding the need for frequent model retraining, reducing the retraining frequency by up to 84%, so reducing the computational burden on clients. An extensive experimental evaluation performed on KronoDroid, an open-source real-world dataset, proves the effectiveness of M2FD in detecting concept drift, minimizing model updates, and achieving high accuracy in mobile malware detection.

Keywords: Federated Learning, Mobile Malware, Concept Drift, Privacy Preserving Computing.

*Corresponding author, e-mail address: andrea.augello01@unipa.it

1 Introduction

Mobile operating systems nowadays run over billions of hardware devices, including smartphones, tablets, and IoT devices [57], with projections estimating more than 30.2 billion devices by 2030 [34]. The extensive mobile app ecosystem, while contributing to their popularity, also makes mobile devices attractive targets for malware developers seeking to steal sensitive information, compromise user privacy, and cause financial loss. Mobile platforms face the regular emergence of new malicious apps and attack vectors [61, 21]. These malicious applications constitute a significant threat to mobile device users, undermining the security and integrity of their devices and data.

The detection of mobile malware has been the subject of extensive research and, while automated detection systems are remarkably effective in the short term, it has been shown that most approaches are undermined by the faulty assumption that malware characteristics are constant over time [44]. In realistic scenarios, instead, the performance of detection models degrades over time due to the presence of the phenomenon known as *concept drift*. Concept drift describes the situation where the statistical properties of the data change over time in an arbitrary way [41] and makes the task of detecting mobile malware particularly challenging. The presence of concept drift in this domain is not surprising, as malware developers are constantly evolving their attack strategies to evade detection [11], and the landscape of mobile applications is continuously changing [69], with new applications continuously being released and older ones being abandoned. This drift can negatively impact the performance of malware detection systems, as the datasets used to train the detection system may not reflect the current data distribution, necessitating the development of adaptive techniques that can handle concept drift.

Adaptively trained machine learning models can evolve with the data, making them more suitable to mitigate mobile malware attacks. However, most traditional techniques to train adaptive models pose a severe scalability burden as they require centralized storage and expert labeling of data, which is both costly and time-consuming. The difficulty of obtaining labeled data is exacerbated by the fact that concept drift quickly renders the training data outdated [43]. Furthermore, data is distributed across geographically dispersed devices, making the centralized collection and storage a difficult task; as it also poses privacy concerns for the data owners [17].

To mitigate the large overheads of centralized data collection and labeling, crowdsourcing has emerged as a promising approach for malware detection, leveraging the collective intelligence of users in order to obtain valuable data and labels [62]. Despite its potential, crowdsourcing also raises serious user acceptance issues due to their reluctance to share with a central server which applications they have installed. The authors of [53] have shown that a simple snapshot of the installed applications on a device can reveal sensitive information about the user, such as their location, interests, religious beliefs, marital status, medical conditions, and habits.

Federated learning (FL) has emerged as a promising solution to address the privacy preservation and scalability issues in mobile and IoT malware detection [17, 52]. FL is a distributed machine learning technique that trains a model across multiple decentralized clients, each holding its local dataset, without requiring exchanges of sensitive data [1]. By enabling collaborative model training across multiple devices without exposing their data, FL preserves user privacy. Clients train local models on their data samples and exchange only parameters (either model weights or computed gradients) with centralized a server; the server aggregates the received models [35] to produce a global model which is sent back to the clients for further training. FL can thus train a global model during multiple communication rounds without direct access to the private training data produced and collected by the users.

Limited research is currently available on the use of federated learning for mobile malware detection, and the few existing methods are not designed to handle concept drift. More research is necessary to overcome the difficulty of maintaining model performance in a decentralized setting [15]. Existing methods for handling concept drift in malware detection often require centralized datasets and/or constant retraining, imposing a high computational burden on the participating clients and raising privacy concerns. Moreover, these methods struggle to adapt to concept drift quickly, to distinguish noise from actual drift, and to maintain model performance in the presence of significant sudden drift. Further investigation is required to develop robust systems that can effectively solve this problem [66].

Although providing some privacy-related advantages, the current FL approaches available in the literature impose a significant computational burden on these devices as the cost of model training is shifted from the centralized server to participating clients. Such shortcomings result invalidating when energy safeguards are a prioritary goal. Therefore, it is crucial to develop federated learning techniques that minimize the computational burden on individual clients in order to ensure their continuous participation, thus guaranteeing the overall effectiveness of the FL process [55].

This paper presents M2FD (Mobile Malware Federated Detection), a system for collaborative mobile malware detection designed to consider the distributed nature of the data and the difficulties posed by concept drift and ever-changing local client datasets, without compromising user privacy. Under this demanding scenario, this paper contributes a novel system designed to address the complexities associated with malware detection in a dynamic and distributed environment, integrating federated learning, concept drift detection, and adaptive model retraining.

M2FD aims to optimize the detection accuracy by adapting to the evolving distribution of applications on individual devices by incorporating robust concept drift detection mechanisms, while minimizing the computational burden on participating clients. A significant contribution of the approach described here is the effective handling of concept drift in the mobile ecosystem which would otherwise introduce difficulties in maintaining high detection accuracy over time. This adaptive capability is crucial for sustaining high-performance malware detection in presence of evolving attack strategies.

The detection algorithm strives to optimize detection accuracy while respecting the computational constraints of individual client devices. Recognizing the limited computational capabilities of mobile devices, M2FD is designed to distribute the computational burden efficiently. By increasing the time between model retraining without significant loss in accuracy, M2FD ensures that resource-constrained devices, such as smartphones, can participate effectively in the federated learning process without significant disruption to their primary tasks. The main contributions of this paper are:

- M2FD overcomes the need of updated centralized datasets by crowdsourcing data from users, preserving their privacy through the use of federated learning;
- M2FD effectively adapts to concept drift in the mobile malware detection domain, maintaining stable detection accuracy over time;
- by effectively recognizing concept drift, M2FD reduces the need for frequent model retraining, minimizing the computational burden on the clients.

To assess the effectiveness of the detection algorithm, extensive experiments were performed using open-source real-world datasets containing mobile applications published over several years (KronoDroid [29] and Malware Hunter 100k [12]). Additionally, its applicability to other domains was evaluated using the EMBER Windows malware dataset [7]. This experimental evaluation confirms the ability of M2FD to detect concept drift, reduce the computational burden on the clients, and achieve high accuracy in mobile malware detection. Furthermore, M2FD proves to be resilient to noisy training labels, a common obstacle to real-world deployment of machine learning models. This increases its reliability and effectiveness in scenarios lacking centralized authority, where data quality may vary without a clear shared criterion for labeling.

The remainder of the paper is organized as follows. Section 2 provides an overview of the related works. Section 3 outlines the M2FD architecture and introduces the client model and problem formulation. Section 4 details the M2FD algorithm. Section 5 presents the experimental evaluation. Finally, section 6 concludes the paper.

2 Related works

This section provides an overview of the current mobile malware detection techniques, analyzes the issues caused by concept drift, and discusses the existing methods for detecting and mitigating such issues in the considered domain.

2.1 Malware detection in mobile devices

The landscape of malware detection is marked by the constant evolution of malicious techniques to avoid identification. Traditional signature-based methods, while lightweight, are vulnerable to evasion by malware developers through simple modifications [50]. Malware developers can prevent signature-based detection by modifying their software, employing polymorphic techniques, and obfuscating their code through compression [3].

In response, the field has seen a burst of research focusing on machine learning approaches for automated malware detection [39]. These techniques span from traditional machine learning algorithms such as Naive Bayes Classifiers [54], Support Vector Machines (SVM) [33] and Random Forests [74], to deep learning approaches such as Convolutional Neural Networks (CNN) [36] and Recurrent Neural Networks (RNN) [67].

Machine learning techniques for mobile malware detection can utilize both static and dynamic features for their classifications [6, 23]. Methodologies based on static analysis extract features from the application's code and other metadata information, such as that included in the manifest file of Android applications [75], or extract more complex features such as the app control flow graph [38, 5]. Dynamic analysis techniques monitor the application's behavior at runtime [10], e.g., by analyzing the system calls it makes, the generated network traffic [32] and the memory contents [65]. Both types of features are often used together to improve the detection accuracy [60]. Despite the widespread use of machine learning in malware detection for mobile devices, the authors in [28] have shown that concept drift negatively affects the performance of machine learning models for malware detection for both analysis approaches.

2.2 Concept drift in mobile malware

Concept drift is a phenomenon in machine learning involving arbitrary changes in the statistical properties of a target domain over time [41]. This shift in the data statistical properties can be caused by alterations in hidden variables that cannot be directly measured, for instance, the discovery of new vulnerabilities or the introduction of new malware lineages [4]. Concept drift can be categorized into three classes: drift in the input data distribution, drift in the conditional probability of the output given the input, and a mixture of both. In mobile malware detection, both kinds of drift can occur. The statistical distribution of the application features can change due to evolving trends in application usage and functionality. Furthermore, in the specific case of mobile applications, the conditional probability of their malicious nature given the characterizing features is also subject to their continuous effort of detection avoidance. If concept drift is not taken into account when developing and evaluating automated malware detection systems, the performance of the models in the wild will be noticeably worse than during development [4].

These changes can lead to a classification accuracy decrease. The research effort when dealing with these types of drift focuses on how to minimize the drop in accuracy and achieve the fastest recovery rate during the concept transformation process [40]. Despite the significant impact of concept drift on machine learning models, there is a lack of a unified methodology to deal with it. The development of a universally applicable solution is difficult due to the complex nature of concept drift, which can occur in various forms and at different times [9]. Additionally, concept drift can be gradual, where the change occurs slowly over time, or sudden, where the change occurs abruptly. The simultaneous presence of both types of drifts further hinders the development of a universal solution.

The presence of concept drift makes mobile malware detection a particularly complex task, as discussed in [70] and [8], also because this scenario is subject to both the natural incremental drift of benign applications and the drift of malicious applications attempting to avoid detection, which can be sudden and significant [30], as illustrated in Figure 1. The drift of benign applications is gradual, and it is caused by the natural evolution of the mobile ecosystem, with limited negative impact on malware detection models. Malicious applications, on the other hand, exhibit sudden drift, both due to the evolution of existing malware families, and the introduction of new malware lineages, which can significantly impact the performance of malware detection models [19]. Thus, in M2FD, periodic retraining of the model is performed to match the new data distribution and maintaining the model performance, as will be detailed in Section 4. Many other domains, such as Windows executables, are also subject to constant evolution and are thus affected by concept drift. The mobile context is particularly challenging, as trends shift swiftly [58] and mobile devices have limited computational resources to update the models frequently to keep up with faster development cycles [25]. The authors of [18] proved that training an Android malware classifier on one year's worth of data resulted in a 23 percentage point drop in F1-score after just six months of concept drift. Even advanced dynamic behavioral analysis tools like MaMaDroid are not immune to a decline in performance over time [45].

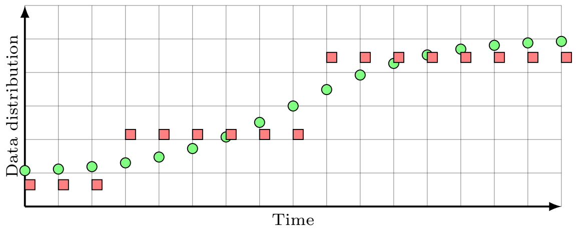


Figure 1: Concept drift in malware detection. Green circles and red squares represent benign and malicious software, respectively. Benign software is subject to gradual incremental drift caused by the evolving trends in usage and functionality, while malicious software try to change their data distribution to evade identification, mostly leading to sudden drift.

2.3 Concept drift detection and mitigation

Several methods and techniques have been developed to detect and prevent concept drift. These include error rate-based drift detection algorithms, such as that proposed by the authors in [26], which monitor the performance of the learning system and assume drift when the error rate exceeds a predefined threshold, and data distribution drift detection methods that focus on detecting the occurrence of a drift by comparing the distribution of the data at different time points, as that proposed by the authors in [49].

To mitigate the impact of concept drift in mobile malware detection, researchers have proposed different techniques, such as using adaptive classifiers [16, 68]. Despite these efforts, concept drift remains an open research field in mobile malware detection, mainly addressed by periodically updating the classifiers if concept drift is detected [22].

Continuous training of a classifier model to react to concept drift is a complex task, as it requires a constant flow of new data to assess drift and labels for the new data to retrain the model [59]. To address this issue, the authors of [72] have proposed a method to delay the need for model retraining by slowing down the performance degradation of malware detection models leveraging the semantic relationships between API calls. However, their approach cannot detect drift when it happens, and still requires a centralized collection of data to assess API relationships and train the model. Reducing the need for human analysis effort remains a key challenge in centralized settings [18].

In this work, federated learning is used to easily collect data and labels from users: users only need to interact with their devices normally and label the new applications they install, without additional data collection effort, bypassing the need for costly periodic operations to gather new data in a central server.

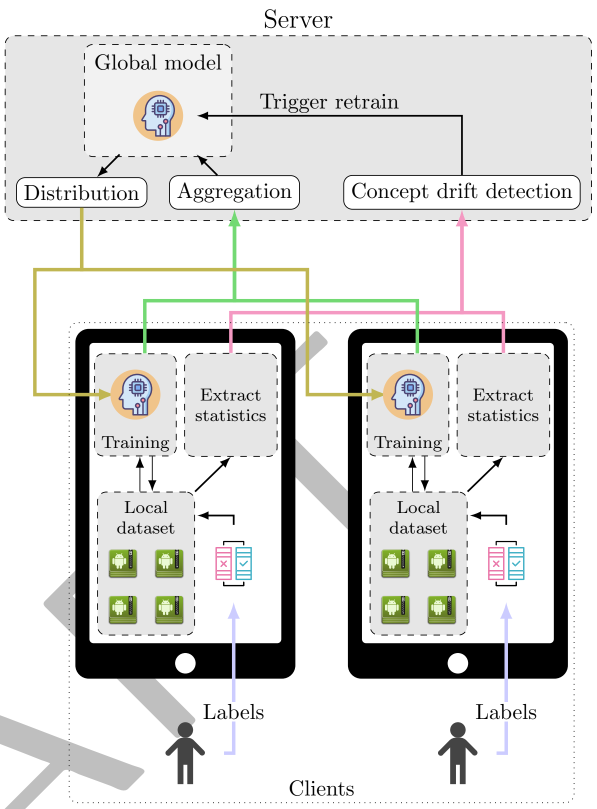


Figure 2: M2FD architecture. The system comprises a server and a set of mobile clients. The server holds the global model and communicates with the clients to distribute the model and detect concept drift. The clients train local models on their local datasets and send the model parameters to the server. Users are required to label their local dataset.

3 M2FD architecture and models

M2FD is designed to leverage federated learning to train a shared model using the collective intelligence of the user community while preserving their privacy. Its architecture comprises two main components, one of which is on end users' mobile devices, and the other on a cloud server, as shown in Figure 2.

Whenever an application is installed, its behavior is monitored by M2FD in order to extract a set of features. The features extracted from each app are obtained through a hybrid analysis. First, static features are extracted from the application's manifest file. These features are, for instance, the requested permissions, the declared activities, and the requested services. The extraction of static features is an automated process that does not require user intervention. Multiple tools are available to extract static features from Android applications, such as Androguard [24] and DexHunter [73] that only require the application's APK file as input. During the app execution, dynamic features are recorded, such as the system calls made by the application. These features can be extracted using external tools such as Android Debug Bridge as done by the authors of [29], or by developing M2FD as an application with root access that can directly access the system log files, as done by the authors of [56]. It is worth noticing that this fine-grained application monitoring can raise privacy concerns, thus federated learning is critical to ensure that data never leaves the device and protect user privacy.

At any time, the user can assign a label to one of his applications, indicating whether it is benign or malicious. By interacting with the application over time, the user can vary the degree of certainty of the assigned label, in response to greater or lesser confidence in his assessment, or can change the label entirely. As in other crowdsourced mobile security works [46, 13], the user interaction with the application is a crucial step in the process, as the crowd-sourced labeling of applications by users is the only source of knowledge for the models. The labels provided by the user constitute a local, private dataset together with the features automatically extracted from the applications. The obtained dataset is stored locally and never shared, ensuring user privacy.

The server holds the parameters of a neural network model and shares them with the clients. Each client uses their dataset to train a copy of the model and then sends the updated model parameters back to the server. The server aggregates the model parameters sent by the clients to update the global model and distributes it back to the clients.

Periodically, the server communicates with the clients to detect concept drift and determine if the local models should be retrained to maintain the global model's accuracy in the presence of concept drift, as will be detailed in Section 4.

The dynamic nature of application installations, uninstallations, and mislabeling introduces complexity to the problem. The primary goal is the optimization of the global model's accuracy on a distributed dataset while considering individual client constraints and the presence of concept drift. The server faces the dual goals of maximizing accuracy and minimizing client-side computational burdens.

3.1 Client model

Each mobile device hosts a client C_i, where i represents the index of the device. Each client possesses a local dataset D_i containing $|D_i|$ samples. These samples correspond to applications previously installed on the device and their user-determined labels, forming the basis for subsequent actions within fixed-length time periods.

3.1.1 Dataset updates

During each time period T, every client C_i undergoes a dynamic process of installing new applications. The number of new applications installed within the time period T in the i-th device is denoted as $k_{i,T}$ and is modeled as a random variable following a Poisson distribution with parameter λ. The Poisson distribution is a suitable choice for modeling the number of new applications installed by each client, as it is commonly used to model the number of events occurring in a fixed interval of time, and it is in accordance with existing studies on the popularity of mobile applications [37] and their usage patterns [51].

The parameter λ in the Poisson distribution is a measure of the average number of new applications installed by each client during a time period T and is assumed to be constant across all clients for all time periods for simplicity. Thus, in a given time period, the probability that client C_i installs k new applications is given by Eq. 1:

$$\Pr(k_{i,T} = k) = \frac{\lambda^k e^{-\lambda}}{k!}. \tag{1}$$

The set of new applications installed by each client, represented as A_i, is drawn from a global pool of applications $\mathcal{D}_T$ independently for each client. Thus, the probability of an application being installed by the i-th client at time T is $\frac{\lambda}{|\mathcal{D}_T|}$. Since an application can be independently installed by multiple users, the expected number of clients installing a given application at time T is $\mathbb{E}[\eta_T] = \sum_{i=1}^{n} \frac{\lambda}{|\mathcal{D}_T|} = n\frac{\lambda}{|\mathcal{D}_T|}$, and the probability of at least one client installing a given application is given by Eq 2:

$$\Pr(\eta_T > 0) = 1 - \prod_{i=1}^{n}\left(1 - \frac{\lambda}{|\mathcal{D}_T|}\right) = 1 - \left(1 - \frac{\lambda}{|D_i|}\right)^t. \tag{2}$$

This assumption of equal probability of selection is a pragmatic compromise necessitated by the constraints of limited public datasets. App stores contain millions of applications with different popularity levels. However, since public datasets are only a subsample of all the existing applications, the same popularity distribution cannot be replicated. Notably, empirical evidence in the context of Android applications suggests a significant concentration of downloads, with a small fraction of applications garnering the majority of user attention and installations [48]. Thus, in our model, we consider the global pool to consist of this subset of very popular applications that roughly have the same probability of being installed by each user, making this assumption realistic.

3.1.2 Removal of old applications

Since the storage capacity of mobile devices is limited, to install new applications each client may need free up storage space by uninstalling a set of existing applications. This involves the random selection of $k_{i,T}$ applications from the set D_i of those installed on the i-th device before T, each chosen with equal probability.

Since each application in D_i has the same probability of being removed from the local dataset of the $i-th$ client at each time period, this probability only depends on the size of the dataset and the number of newly installed applications. Thus, the removal probability can be described as a Bernoulli trial with success (uninstallation) probability given by the ratio $\frac{k_{i,T}}{|D_i|}$. The expected value for said ratio is $\mathbb{E}\left[\frac{k_{i,T}}{|D_i|}\right] = \frac{\lambda}{|D_i|}$.

Let Y be the number of time periods an application remains installed on the device. Y is defined by observing a parallel sequence of independent Bernoulli variables $\mathbf{X} = (X_1, X_2, \dots)$, stopping at the first X_j that is equal to 1, and counting the number of X_j's that are equal to 0.

$$\Pr(Y \geq t|\mathbf{X}) = \prod_{j=1}^{t} \Pr(X_j = 0) = \prod_{j=1}^{t} \left(1 - \frac{k_{i,j}}{|D_i|}\right).$$

Since the variables are independent, the expectation can be expressed as follows:

$$\begin{aligned}\Pr(Y \geq t) &= \mathbb{E}\left[\Pr(Y \geq t|\mathbf{X})\right] = \prod_{j=1}^{t} (1 - \mathbb{E}[X_j]) \\ &= \prod_{j=1}^{t} \left(1 - \frac{\mathbb{E}[k_{i,j}]}{|D_i|}\right) = \prod_{j=1}^{t} \left(1 - \frac{\lambda}{|D_i|}\right),\end{aligned}$$

which can be simplified as $\Pr(Y \geq t) = \left(1 - \frac{\lambda}{|D_i|}\right)^t$. Thus, the probability of an application in the set D_i remaining installed after at least t time periods is a geometric random variable with parameter $\frac{\lambda}{|D_i|}$. Once the $k_{i,T}$ applications have been selected for uninstallation, they are removed from the dataset D_i.

3.1.3 Application labeling

The mobile device's user is required to assess whether the applications in their local dataset are benign or malicious and labeling them accordingly. The labeling process is performed manually by the users, who are presented with the list of their installed applications and are asked to label each application as either benign or malicious. Since the users are not expected to have in-depth domain knowledge, it would be unrealistic to expect them to provide fine-grained labels, such as the specific malware family or type, as this would require a high level of expertise. Instead, the users are asked to provide a binary label, which is sufficient for the purposes of the system. Each user labels the installed applications independently of other users.

Obviously, manual labeling introduces a degree of mislabeling due to users not being security experts. For instance, an application may be incorrectly labeled as malicious simply for malfunctioning. At the same time, some stealthy malware may be able to not be noticed by the client, being labeled as benign. These mistakes are likely to introduce noise in the learning process, thus M2FD is designed to handle high levels of noise in the labels. In this work, we assume that each user provides correct labels at least half of the time. The probability of mislabeling malware as benign is denoted as p_{mal}, and the probability of mislabeling a benign application as malware is denoted as p_{ben}. For simplicity, we assume $p_{mal} = p_{ben} = p$, resulting in a correct labeling probability of $1-p$. Given that multiple clients may independently install the same application, the probability of an application being mislabeled by n clients is calculated as $1 - p^{1-\left(1-\frac{\lambda}{|D_T|}\right)^n}$, with the exponent representing the number of clients that installed the application as calculated in Eq. 2. The data acquisition and annotation process is illustrated in Figure 3.

Over time, a user's perception of an application may change. To reflect this, a user may also change the label assigned to an application. For instance,

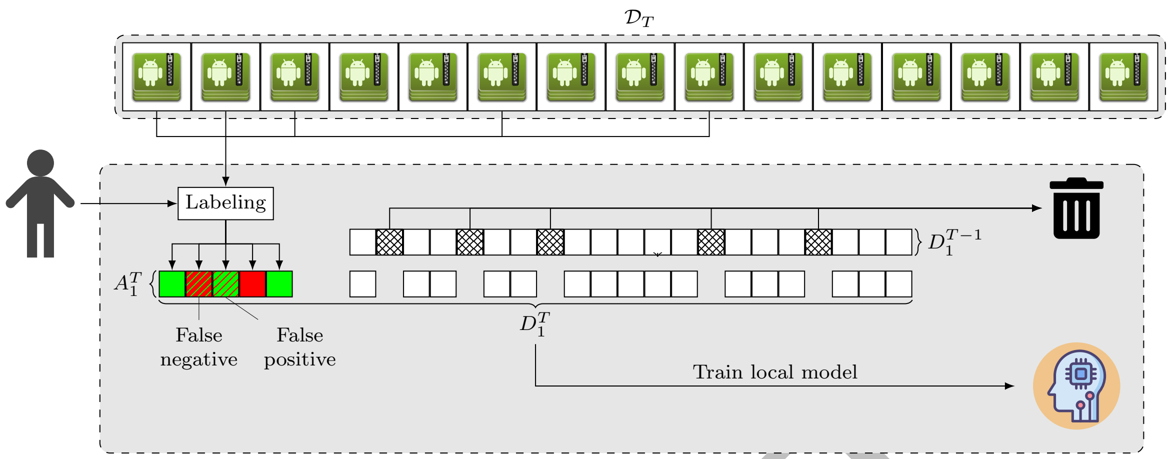


Figure 3: At each time period T, users download new applications from the global pool, add them to their local datasets, and remove old applications. At any time, the user can label an installed application as either benign or malicious. Upon request by the server, when concept drift is detected, the local model is trained on the local dataset and uploaded to the server.

a user might notice that an application previously labeled as benign is now behaving maliciously, or they might re-install again an application that was previously uninstalled because it was considered malicious due to a malfunction. This change in the assigned label is not necessarily an improvement, as the new label may be the incorrect one.

Furthermore, users themeselves may not be completely sure of the label they assign to an application. To account for these uncertain beliefs, M2FD supports the adoption of fuzzy labels, instead of binary ones. For instance, a user might be 70% certain that an application is benign and have a 30% remaining suspicion that it is malicious.

In order to model a varying *certainty* we adopt the Pert probability distribution, which is often used to model expert judgment [27]. To model different user behaviors, different shapes of the *certainty* distribution can be adopted. The specific shape is obtained by setting a *confidence* parameter, named γ, which varies from 0 to 1, and concisely represents the user's attitude to perform fuzzy classifications.

The Pert cumulative distribution function which models the *certainty* is defined by Eq. 3:

$$\alpha = 1 + 4 \cdot \frac{\min(1, 0.5+\gamma) - 1}{\max(0.5, 1-\gamma)},$$
$$\beta = 1 + 4 \cdot \frac{1 - \min(1, 0.5+\gamma)}{\max(0.5, 1-\gamma)},$$
$$\Pr(X \leq u) = \int_{-\infty}^{u} \frac{(x-\max(0.5,\gamma))^{\alpha-1}(1-x)^{\beta-1}\Gamma(\alpha+\beta)}{(1-\max(0.5,\gamma))^{\alpha+\beta-1}\Gamma(\alpha)\Gamma(\beta)}\, dx, \quad (3)$$

where Γ denotes the gamma function and α and β are the shape parameters depending on the *confidence* parameter γ.

Distribution shapes corresponding to different values of γ are shown in Figure 4. A user with $\gamma = 0$ is completely uncertain about his labels, and the most frequent sampled certainty is 50%. Increasing values of γ shift the distribution mode towards 100%, leaving the minimum set as 50%. With $\gamma = 0.5$, the mode and maximum value coincide and are equal to 100%. Further increasing γ results in an increase in the minimum allowed certainty value. Finally, a user with $\gamma = 1$ is fully certain about his labels, thus he only produces binary labels.

3.2 Server goal

The server has two main goals. The first goal is the maximization of the accuracy of the global model on the dataset $\mathcal{D}_T$ at any given time T. Simultaneously,

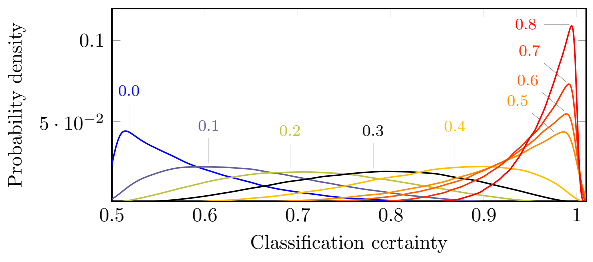


Figure 4: Probability density function of the classification *certainty* with different levels of user *confidence*.

the server strives to minimize the computational burden imposed on individual clients, quantified by the frequency of updates to the global model performed by clients. Furthermore, the server must address concept drift, since it could render the global model obsolete. To this end, the server takes proactive measures to adapt to changing data patterns by retraining the global model on the most recent data from the clients.

Therefore, the server has to manage two conflicting goals: mitigating concept drift by frequently updating the global model and minimizing the computational burden on the clients by limiting the number of times the global model is retrained.

To model the retraining decision, the set of all time periods is denoted as $\mathcal{T}$. Out of these time periods, the subset for which the global model is retrained is referred to as $\mathcal{T}_r \subseteq \mathcal{T}$. For each time period T, the current global model is denoted as θ_T. If T is in $\mathcal{T}r$, then θ_T refers to the global model after retraining; otherwise, θ_T is equal to θ_{T-1}.

In order to reduce the computational burden on the clients, the server aims to determine the smallest set of time periods $\mathcal{T}_r$ for which the global model θ_T is retrained that still minimizes the loss $\mathcal{L}$ of the global model on the dataset $\mathcal{D}_T$ at any given time T. This objective is formalized as the optimization problem in Equation 4.

$$\begin{aligned} \min \quad & |\mathcal{T}_r| \\ \text{such that} \quad & \text{minimize} \sum_{T \in \mathcal{T}} \mathcal{L}(\theta_T, \mathcal{D}_T). \end{aligned} \tag{4}$$

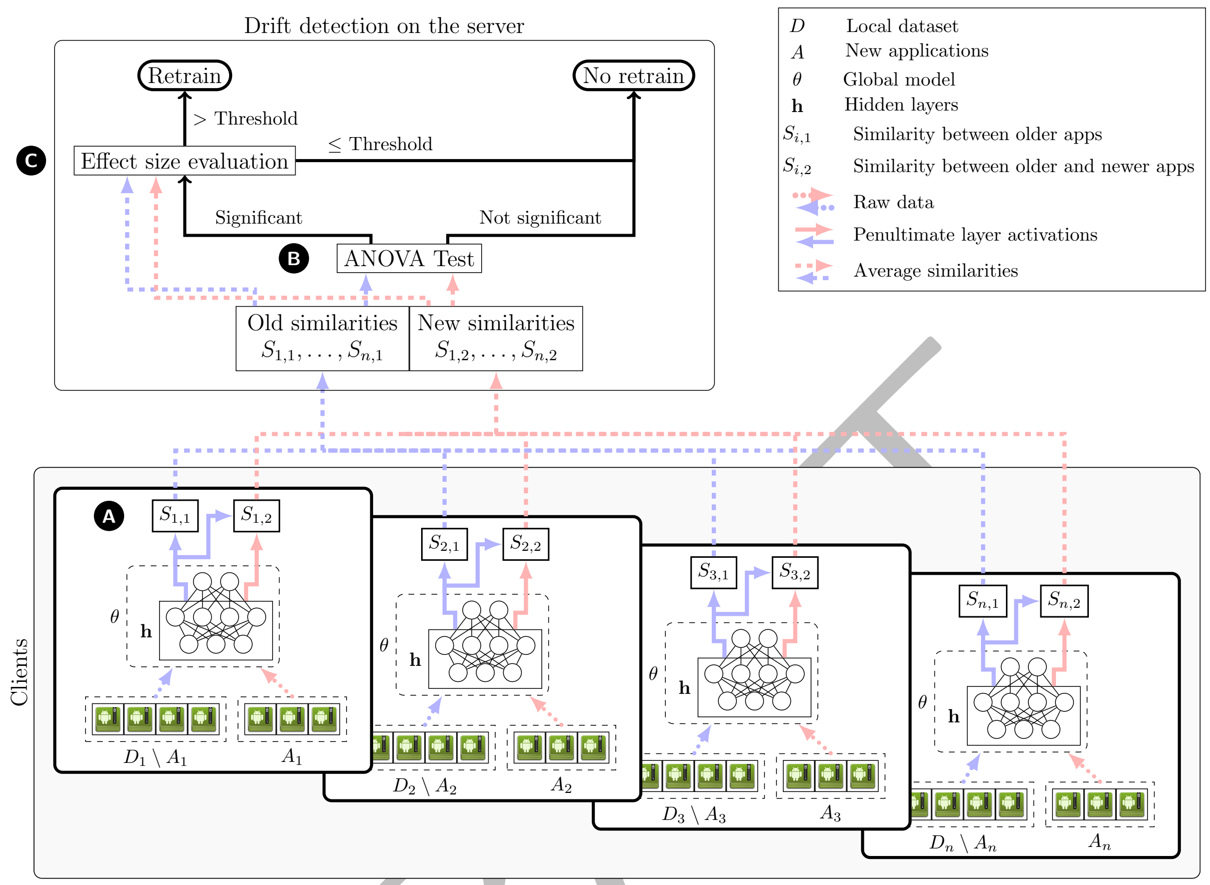
The goal of minimizing the loss function $\mathcal{L}(\theta_T, \mathcal{D}_T)$ is pursued to maximize the accuracy of the global model on the dataset $\mathcal{D}_T$ at any given time T. Thus, overall, the server is tasked to determine the minimum required set of retraining time periods $\mathcal{T}_r$ that are sufficient to maintain the accuracy of the global model over time. The loss minimization objective is commonly used in federated learning [42], and it is achieved by optimizing the utility of the global model, as shown in Equation 5, where $\mathcal{L}(\theta, D_i)$ is the loss function of the global model θ on the local dataset of client C_i at time T:

$$\theta_T = \arg\min_{\theta} \sum_{i=1}^{n} \frac{|D_i^T|}{\sum_{j=1}^{n} |D_j^T|} \mathcal{L}(\theta, D_i). \tag{5}$$

To achieve this goal, the Federated Averaging (FedAvg) algorithm [42] is employed whenever the server determines that retraining is necessary. FedAvg is a well-established approach in federated learning that harnesses the collective intelligence of individual clients while preserving their data privacy. The algorithm operates synchronously over multiple rounds. In each round, clients download the current global model, train it on their local dataset, and subsequently upload their updated model parameters to the server. The server aggregates these parameters to compute a new global model. The global model in each client is updated using Stochastic Gradient Descent (SGD) on their local dataset for a specified number of epochs. The updated local model parameters are then uploaded to the server, where they are aggregated to compute the new global model parameters. This process is iterated for a given number of rounds.

4 M2FD Algorithm

The M2FD algorithm introduces a novel approach to determine the optimal periods for retraining the global model in a federated learning setting, addressing the challenges arisen from the dynamic nature of mobile devices. It effectively handles the inherent complexity introduced by dataset variations and potential mislabeling of applications. M2FD is designed to ensure that the accuracy of the global model is maintained over time, even in presence of concept drift; to this end it considers individual client constraints and minimizes the computational burden on the clients by determining the optimal set of time periods for retraining the global model.



Drift detection on the server
Retrain
No retrain
> Threshold
≤ Threshold
C
Effect size evaluation
Significant
Not significant
B
ANOVA Test
Old similarities $S_{1,1}, \dots, S_{n,1}$
New similarities $S_{1,2}, \dots, S_{n,2}$
D Local dataset
A New applications
θ Global model
$\mathbf{h}$ Hidden layers
$S_{i,1}$ Similarity between older apps
$S_{i,2}$ Similarity between older and newer apps
Raw data
Penultimate layer activations
Average similarities
Clients
A
$S_{1,1}$ $S_{1,2}$ θ $\mathbf{h}$ $D_1 \setminus A_1$ A_1
$S_{2,1}$ $S_{2,2}$ θ $\mathbf{h}$ $D_2 \setminus A_2$ A_2
$S_{3,1}$ $S_{3,2}$ θ $\mathbf{h}$ $D_3 \setminus A_3$ A_3
$S_{n,1}$ $S_{n,2}$ θ $\mathbf{h}$ $D_n \setminus A_n$ A_n

Figure 5: Concept drift detection in M2FD. On the server's request, the clients send aggregate information about the similarity of the samples in their local datasets (A) to the server. The server uses the ANOVA statistical hypothesis test (B) to determine the presence of concept drift, if there is a significant concept drift and its effect size is large (C), the server will decide to retrain the global model.

A high-level overview of the concept drift detection process is shown in Figure 5. First, the global model is used to extract high-level features from the local datasets of the clients. Then, each client autonomously computes the pairwise cosine similarity between the extracted feature vectors of the new samples and the old ones in their local dataset. The average similarity among the older samples and the similarity between the new and old samples are reported to the server. The server uses statistical hypothesis testing to determine whether the new samples are drawn from the same distribution as the old ones. If the new samples are not likely to originate from a different distribution, the server maintains the current global model. Otherwise, the server checks if the new distribution is significantly different from the old one. If there is a big gap between the two distributions exceeding a predetermined threshold, the server communicates with the clients to retrain the global model on the new data.

Federated learning is employed as the underlying mechanism for model updates, enabling collaboration among mobile device users while respecting privacy constraints. Different from the usual federated learning approach, M2FD is designed to dynamically determine the optimal set of time periods $\mathcal{T}_r$ to update the global model, while maintaining the previous model for the remaining periods.As a result, the computational burden on the clients is minimized while ensuring that the global model is up-to-date and robust to concept drift.

As previously stated in Section 3.2, in M2FD, the retraining decision is taken in a centralized manner by the server. However, to conform to the FL paradigm, the server does not have direct access to the client data. Instead, the clients provide aggregate information about the state of their local datasets, in order to enable the server to detect eventual concept drift.

To this aim, each client independently extracts a robust set of features from its local dataset. Since the global model is a neural network with multiple layers, the activations of the upper layers can be used to capture the high-level features of the samples in the dataset without disclosing the raw data. Specifically, the chosen neural network model is a 4-layer fully connected architecture with 128, 64, 16, and 2 neurons in each layer, respectively. A Rectified Linear Unit (ReLU) serves as the activation function, and the output layer employs a softmax function.

Let $\mathbf{h}_{i,T}^{(k)}$ be the activation vector of the k-th sample in the local dataset of client C_i at time T. This activation captures the high-level features learned by the model and serves as a representation of the samples in the dataset in a compressed form. Rather than detecting drift on the raw data, drift is detected on the learned features, which are more robust to noise and more compact [64].

To assess whether the set of new samples A_i, installed by the i-th client, was extracted from the same distribution as the old ones, clients compute the average pairwise cosine similarity between the activation vectors of the new samples.

$$S_{i,1} = \frac{2}{|A_i|(|A_i|-1)} \sum_{k=1}^{|A_i|} \sum_{j=k+1}^{|A_i|} \frac{\mathbf{h}_{i,T}^{(k)} \cdot \mathbf{h}_{i,T}^{(j)}}{\|\mathbf{h}_{i,T}^{(k)}\| \|\mathbf{h}_{i,T}^{(j)}\|}. \quad (6)$$

Additionally, each client calculates the average pairwise cosine similarity between the activation vectors of the new samples and the old ones already in D_i ($D_i \setminus A_i$), i.e., all the samples acquired before T that are still present in the local dataset, as follows:

$$S_{i,2} = \frac{1}{|A_i||D_i \setminus A_i|} \sum_{k=1}^{|A_i|} \sum_{j=|A_i|+1}^{|D_i|} \frac{\mathbf{h}_{i,T}^{(k)} \cdot \mathbf{h}_{i,T}^{(j)}}{\|\mathbf{h}_{i,T}^{(k)}\| \|\mathbf{h}_{i,T}^{(j)}\|}. \quad (7)$$

These similarity measures provide insight into the relationships between the new and existing samples in the local dataset. Each client communicates with the server by reporting the average similarity among the new samples (Eq. 6) and the similarity between the new samples and the old samples in D_i (Eq. 7). This information is crucial for the server to assess the degree of similarity and potential concept drift in the overall dataset, in order to perform an informed decision regarding model retraining. The server's responsibilities encompass statistical hypothesis testing, effect size evaluation, and decision-making based on the obtained results.

To assess the overall similarity of the new samples to the old ones, the server employs statistical hypothesis testing, specifically ANOVA (Alexander-Govern test [2]). This test verifies if two or more independent means, extracted from sample populations with differing sizes and variances, are equal. Briefly, the ANOVA test is computed over the differences between the mean of the groups and the sum of the samples in each group weighted by the inverse of the in-group variances.

The null hypothesis posits that the similarity between the new samples and the old ones is equal to the similarity between the new samples alone. This hypothesis is tested through a paired sample test, providing a robust evaluation of the differences in similarity while accounting for potential client heterogeneity. Since the similarities between the new samples and the old ones are computed on a client-by-client basis, the server performs the ANOVA testing using the clients as the unit of analysis rather than the individual samples. Doing otherwise would introduce dependencies and correlation between the samples, thus invalidating the statistical test. Moreover, by averaging the scores for all the applications in a single client, the client only needs to report two aggregated similarity measures, rather than the similarities of each sample, reducing information leakage (i.e., the server does not know how many new applications each client has installed).

It is worth noting that, for a more granular analysis of concept drift on a client-by-client basis, the server may opt for a two-way ANOVA [71], by testing the interaction between new and old samples for each client. This approach would allow the server to discern variations in similarity specific to individual clients rather than the overall dataset, possibly limiting the number of clients that need to retrain the model. However, to grant enough statistical power to the test, this approach would require the clients to report the similarity of each sample, thus increasing the amount of information leaked to the server. Additionally, the limited number of new samples installed in each time period may not be large enough to provide a robust statistical analysis.

Once the server has performed the statistical hypothesis testing, if the null hypothesis is not rejected with a high degree of confidence, the server concludes that there is no significant difference in similarity between new and old samples, and no retraining is required in the current time period. Instead, if the statistical hypothesis testing reveals a significant difference in similarity, the server proceeds to evaluate the effect size using Cohen's d [20] as in Eq. 8, with $\bar{S}_1$ and $\bar{S}_2$ being the average similarity measures of the new samples and the old ones across all clients, respectively:

$$d(S_{1,1}, S_{1,2}, \ldots, S_{n,1}, S_{n,2}) = \frac{\frac{1}{n}\sum_{i=1}^{n}(S_{i,1} - S_{i,2})}{\sqrt{\sum_{i=1}^{n}\frac{(S_{i,1}-\bar{S}_1)^2+(S_{i,2}-\bar{S}_2)^2}{2n}}}. \tag{8}$$

The effect size provides a quantitative measure of the magnitude of the observed difference. A large effect size (typically considered as $d > 0.8$ [20]) indicates a substantial dissimilarity between new and old samples, suggesting a need for updating the model.

Upon obtaining the results of the statistical analysis and effect size evaluation, the server can determine whether to start model retraining. If the difference in similarity is found to be statistically significant and the effect size is relevant, indicating a notable divergence, the server requests the clients to retrain the global model. These conditions are designed to ensure that the model adapts to evolving data patterns while avoiding unnecessary retraining in the absence of significant concept drift.

By combining robust data representation, rigorous statistical methods, client-specific analysis, and careful consideration of effect sizes, the M2FD algorithm is designed to balance the dual objectives of detecting concept drift and minimizing computational burdens on individual clients, contributing to its robustness and efficiency. The M2FD concept drift detection algorithm is summarized in Algorithm 1.

While this algorithm is designed to address the issues caused by the dynamic nature of mobile devices and their dataset variations, it does not explicitly account for possible incremental drift patterns. Thus, if changes in the data distribution occur too gradually, the system may not be able to detect them. While we acknowledge that this limitation to general-purpose usage of the M2FD exists, studies on mobile malware detection have shown that the drift phenomenon is typically sudden [28] thus making the presented approach adequate for the task at hand.

Algorithm 1 M2FD concept drift detection algorithm

```
Input: Local datasets $D_1, \ldots, D_n$ of $n$ clients, global model $\theta_{T-1}$, time period $T$
Output: Retrain decision based on concept drift
 1: Clients:
 2: for $i = 1 \rightarrow n$ do
 3:     $S_{i,1} \leftarrow 0$
 4:     $S_{i,2} \leftarrow 0$
 5:     for $k = 1 \rightarrow |A_i|$ do
 6:         for $j = k+1 \rightarrow |A_i|$ do
 7:             $S_{i,1} \leftarrow S_{i,1} + \frac{\mathbf{h}_{i,T}^{(k)} \cdot \mathbf{h}_{i,T}^{(j)}}{\|\mathbf{h}_{i,T}^{(k)}\| \|\mathbf{h}_{i,T}^{(j)}\|}$
 8:         for $j = |A_i| + 1 \rightarrow |D_i|$ do
 9:             $S_{i,2} \leftarrow S_{i,2} + \frac{\mathbf{h}_{i,T}^{(k)} \cdot \mathbf{h}_{i,T}^{(j)}}{\|\mathbf{h}_{i,T}^{(k)}\| \|\mathbf{h}_{i,T}^{(j)}\|}$
10:     $S_{i,1} \leftarrow \frac{2}{|A_i|(|A_i|-1)} S_{i,1}$
11:     $S_{i,2} \leftarrow \frac{1}{|A_i||D_i \setminus A_i|} S_{i,2}$
12:     SEND $S_{i,1}$, $S_{i,2}$ to server
13: Server:
14: $\bar{S}_1, \bar{S}_2 \leftarrow \frac{1}{n}\sum_{i=1}^{n} S_{i,1}, \frac{1}{n}\sum_{i=1}^{n} S_{i,2}$
15: $\sigma_1, \sigma_2 \leftarrow \sqrt{\frac{\sum_{i=1}^{n}(S_{i,1}-\bar{S}_1)^2}{n(n-1)}}, \sqrt{\frac{\sum_{i=1}^{n}(S_{i,2}-\bar{S}_2)^2}{n(n-1)}}$
16: $w_1, w_2 \leftarrow \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2(\sigma_1^2+\sigma_2^2)}, \frac{\sigma_1^2\sigma_2^2}{\sigma_2^2(\sigma_1^2+\sigma_2^2)}$
17: $t_1, t_2 \leftarrow \frac{(1-w_1)\bar{S}_1 - w_2\bar{S}_2}{\sigma_1}, \frac{(1-w_2)\bar{S}_2 - w_1\bar{S}_1}{\sigma_2}$
18: $significance \leftarrow p\text{-}value(t_1, t_2)$
19: if $significance < 0.005$ then
20:     $effect\_size \leftarrow d(S_{1,1}, S_{1,2}, \ldots, S_{n,1}, S_{n,2})$
21:     if $effect\_size > 0.8$ then
22:         return Request clients for model retraining
23: return No retraining needed
```

5 Experimental evaluation

To comprehensively evaluate the effectiveness of M2FD, extensive experiments were conducted on the KronoDroid dataset [29], a publicly available real-world dataset spanning from 2008 to 2020.

According to the dataset authors, which underline that the earlier and later years in this dataset have a limited number of samples [28], the experiments are focused on the 7-years period from 2011 to 2018. The KronoDroid dataset, consisting of 28,745 malware samples and 35,256 benign samples, sorted according to their "last modification" date, provides a rich and dynamic environment for assessing the system's performance in Android malware detection over multiple years. Each sample is represented by a set of 289 dynamic features, such as system calls, and 200 static features, such as permissions and metadata. The dataset presents two instances of especially sudden drift, in the last quarter of 2015 and the third quarter of 2016 chunks, linked to abruptly emerging different patterns in permission usage and API calls in the malware samples [31]. The two drifts are uncorrelated, with different sets of features acquiring greater relative importance in the classification of malicious samples.

5.1 Experimental setup

The evaluation process involves dividing the dataset into 3-month chunks, each representing a different time period for comparability with [30]. The initial chunk $\mathcal{D}_0$ is dedicated to initializing the system, with each client sampling their entire applications dataset from the global pool and labeling them.

All the subsequent chunks undergo a "testing-then-training" procedure. Initially, the samples are classified using the current global model to evaluate the accuracy of the model on the new data. Following this classification, clients sample new applications from the global pool and label them. Unless otherwise stated, the clients are assumed to have a confidence parameter $\gamma = 1$. Each client is expected to install an average of $\lambda = 5$ new applications during a time period T. If the server determines the need for model retraining, clients retrain the model on their local datasets and upload the updated parameters to the server.

The federated learning training process is executed using the FedAvg algorithm, involving 5 local epochs and 100 rounds. Local model training is performed using SGD with a learning rate of 0.01 and cross-entropy loss.

Feature selection is a critical aspect of model performance. The input features are chosen from the static and dynamic features of the dataset. Specifically, the selected features are the top 32 features with the highest mutual information across all clients' datasets in the first time period. These features are listed in Table 1. Once selected, these features are used for all subsequent time periods.

The experiments were designed to assess the system's performance in detecting concept drift, limiting model retraining, and achieving high accuracy in malware detection, by evaluating the following performance metrics: accuracy, F1-score, and the number of model updates.

5.2 M2FD performance

In order to obtain a baseline evaluation of M2FD, an initial "optimistic" evaluation has been performed, by involving 100 clients with 0% wrong labels. The M2FD retraining strategy has been compared with other two baseline approaches, namely "Always retrain" and "Only train in the first time period". These two strategies represent the two extremes of the retraining spectrum and are addressed as "Always" and "First" in the following. M2FD assesses the need for retraining every three months for homogeneity with the approach proposed in [30]. Analogously, the "Always" baseline retrains every three months. The results, presented in Figure 6, demonstrate M2FD's ability to detect concept drift, updating the model only five times over the 7-year period while maintaining accuracy comparable to that of the "Always" approach. In Figure 6, instants where the model is updated are circled. Notably, these events coincide with drops in model performance. However, the performance of M2FD quickly converges to the accuracy level of the "Always" strategy after each retraining event, whereas the "First" approach fails to recover.

A more detailed performance analysis of M2FD is presented in Table 2. Two additional baselines, CDA-FedAvg [14] and M2FD-, are introduced for comparison. CDA-FedAvg is chosen as it represents the state-of-the-art approach for concept drift detection in federated learning settings. Unlike M2FD, CDA-FedAvg detects concept drift at the client level,

Feature	**Description**
getuid32	Get user identity
prctl	Manipulate the behavior of the calling thread or process
sigaltstack	Changes signal stack
read	Read from file
write	Write to file
munmap	Unmap files or devices into memory
mprotect	Change protections of memory regions
ioctl	Manipulate device parameters of special files
writev	Write data into multiple buffers
dup	Duplicate a file descriptor
fsync	Synchronize a file's in-core state with storage device
lseek	Reposition read/write file offset
fstatfs64	Get file system statistics
sigaction	Examine and change a signal action
rt_sigprocmask	Examine and change blocked signals
recvfrom	Receive a message from a socket
getsockopt	Get options on sockets
clock_gettime	Retrieve the time of the specified clock clockid.
gettimeofday	Get time
futex	Fast user-space locking
pread	Read from a file descriptor at a given offset
getrlimit	Get resource limits
SYS_306	Commit buffer cache to disk
SYS_310	Transfer data between process address spaces
SYS_315	Set scheduling policy and attributes
SYS_317	Operate on Secure Computing state of the process
SYS_329	Set protection on a region of memory
SYS_336	Set protection on a region of memory and assigns a protection key
ACCESS_WIFI_STATE	Allows applications to access information about Wi-Fi networks
READ_PHONE_STATE	Allows read only access to phone state, including the current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device
READ_SMS	Allows an application to read SMS messages
RECEIVE_BOOT_COMPLETED	Allows an application to receive the boot completed Intent that is broadcast after the system finishes booting

Table 1: The selected features ordered by mutual information.

rather than at the global level, using the certainty of local model predictions as an indicator for possible concept drift, detected through a Cumulative Sum test. The drift detection is also performed every three months. M2FD- is a variant of the algorithm proposed here that performs drift detection on the input features instead of penultimate layer activations. This comparison aims to assess the robustness of relying on higher-level features rather than raw data.

The performance metrics in Table 2 show that M2FD achieves an average accuracy of 0.85 ± 0.06 and an F1-score of 0.84 ± 0.08 updating the model five times over the 7-year period. In comparison, M2FD- and the "Always" baseline exhibit similar accuracy but require 27 updates. The "First" baseline, which trains the model only once, demonstrates a lower accuracy, emphasizing the importance of addressing concept drift. CDA-FedAvg, despite its frequent updates, falls behind M2FD.
The results indicate that input features exhibit greater variability, necessitating model updates at every time period, while penultimate layer activations remain more stable. CDA-FedAvg updates the model almost after every time period, leading to suboptimal performance due to the detection being performed on a client-by-client basis. This approach results in only some clients retraining the model at each update, causing inefficiencies despite potentially higher communication overheads.

In order to monitor API calls, the system needs a high level of control on the device. Since this char-

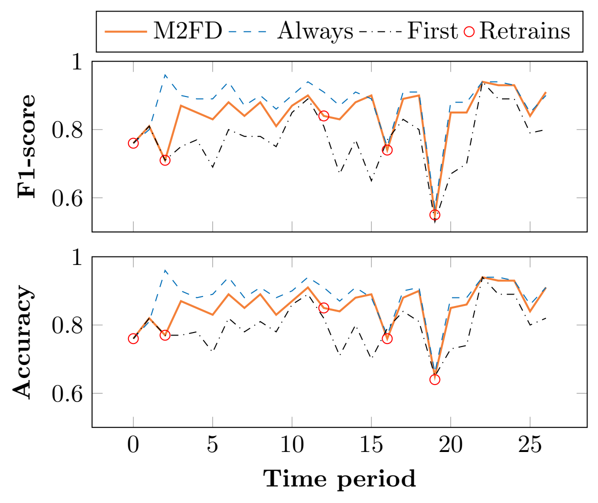


Figure 6: M2FD performance over time. The classifier needs to be trained only 5 times over 7 years (circled in red), while maintaining almost the same accuracy as the "Always" retraining approach.

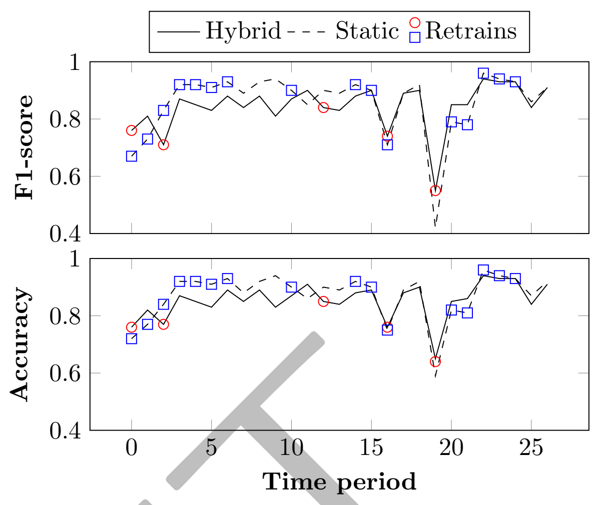


Figure 7: M2FD performance over time with different feature sets. Using only static features ("Static") results in more frequent model updates, as app permissions are more prone to concept drift, but can still maintain overall comparable performance compared to the M2FD version with access both to static and dynamic features ("Hybrid").

Method	**Accuracy**	**F1-Score**	**Updates**
M2FD	0.85 ± 0.06	0.84 ± 0.08	5
M2FD-	0.88 ± 0.06	0.88 ± 0.08	27
Always	0.88 ± 0.06	0.88 ± 0.08	27
First	0.80 ± 0.06	0.77 ± 0.09	1
CDA-FedAvg	0.83 ± 0.06	0.82 ± 0.08	24

Table 2: Average performance and standard deviation of M2FD and the baselines over 7 years. The number of updates is the number of times a new model is distributed to the clients.

acteristic could raise privacy concerns, developers of a M2FD commercial version may choose to implement it without the rights enabling the monitoring of the dynamic behavior of other apps; in such a case, the available feature set would be reduced. Thus, to assess the impact of the available feature set on the system's performance, the performance of M2FD was also evaluated using only static features which can be obtained from the APK file without any superuser access. Compared to training with the full feature set, the system using only static features ("Static") requires more frequent model updates, as shown in Figure 7, since static features are more prone to concept drift. Nevertheless, with an average F1-score of 0.86 ± 0.11, the system can maintain a comparable performance to the full feature set, demonstrating the system's robustness to the feature set used and its potential applicability in real-world scenarios.

To the best of the authors' knowledge, the KronoDroid dataset is the most up-to-date openly available dataset for Android malware detection which contains both static and dynamic features and is annotated with timestamps. However, since the evaluation on KronoDroid is limited to a timeframe spanning from 2011 to 2018, in order to ensure the real-world effectiveness of M2FD, an additional evaluation was performed on the Malware Hunter 100k Dataset [12], a large-scale dataset containing 100,000 benign and malware samples, with a distribution of roughly 10% benign and 90% malware samples containing samples up to 2022. This dataset contains only statically extracted features, so it was not considered for the main evaluations of M2FD but only to ensure that the proposed strategy is still effective on more recent examples of malware. The results of the analysis on this dataset are shown in Figure 8. Similarly to the KronoDroid dataset, never updating the model is not a sustainable long-

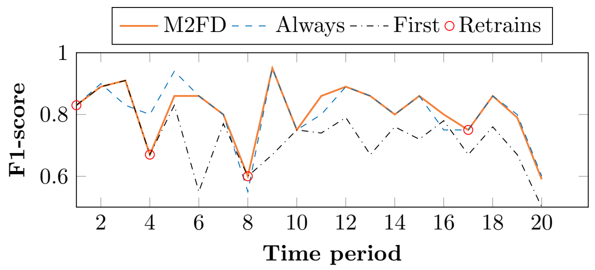


Figure 8: M2FD performance over time on the Malware Hunter 100k dataset [12]. M2FD is able to match the performance of the "Always" baseline, demonstrating its effectiveness on more recent data, while the "First" strategy experiences a significant performance drop.

term strategy as the performance of the "First" strategy drops significantly over time. M2FD, instead, after each performance drop brought by the concept drift, is able to effectively determine the necessity of model retraining, matching the performance of the "Always" baseline with only 4 updates over the analyzed period.

5.3 Impact of client number

The performance of M2FD was then tested with a reduced number of clients, ranging from 5 to 100, to assess the system's robustness to varying client populations. The success of a crowdsourced system depends on its ability to maintain performance with a smaller number of clients, since at the beginning of the system deployment the number of clients may be very limited, and if these initial clients are not satisfied with the system's performance the system may not be able to attract more clients. The results of this analysis are shown in Figure 9.

Independently of the number of clients, training the model only once at the beginning of the deployment ("First" strategy) results in a significantly lower performance compared to the other tested methods. This poor performance is closely followed by the CDA-FedAvg approach, especially when the number of clients is reduced. M2FD, instead, is able to maintain a comparable performance with the setup involving 100 clients even with as few as 5 clients. In the case of 5 clients, M2FD updates the global model with a higher frequency, 12 times out of the 27 time periods, but there is no significant

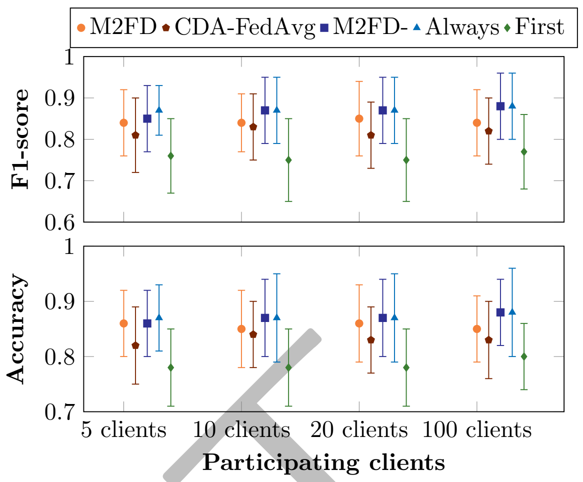


Figure 9: Average performance of the tested methods and standard deviation with varying number of clients.

performance drop, closing the gap with the "Always" approach.

Additionally, we compared the performance of M2FD with the state-of-the-art centralized approach proposed by Guerra-Manzanares et al. in [30]. Is worth noting that the centralized approach uses the entire available dataset to update the model at each time period, while M2FD relies only on the data provided by the clients, obtaining a set of samples which, in scenarios with fewer clients, can cover as few as 25 samples. Another relevant difference is that while the centralized approach proposed in [30] always updates the model, M2FD evaluates from time to time whether this retraining phase is necessary, and therefore updates the model less frequently. Despite the more constraints, M2FD is able to maintain performances comparable to the centralized approach, even with a reduced number of clients and with a limited amount of data, as shown in Figure 10. The performance gap between the two approaches is minimal, with the centralized approach achieving an average F1-score of 0.87 [30] and M2FD achieving an average F1-score of 0.84 with only 5 clients. Since the "Always" baseline, as shown in Table 2, achieves comparable results to Guerra-Manzanares et al., the small drop in performance of M2FD is imputable to the noticeably reduced number of model updates and not to the limited access to training data. Hence, the viability

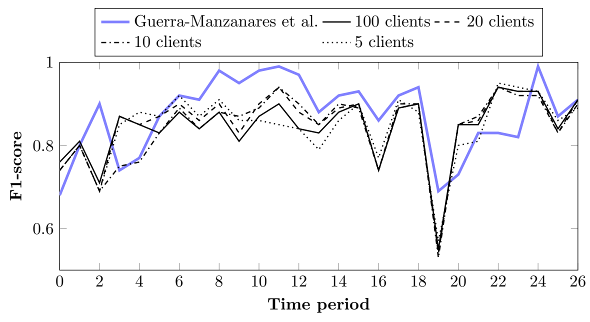


Figure 10: M2FD performance over time varying the number of participating clients. The system is robust to reductions in the number of clients, maintaining good performances with as few as 5 clients. The results are also compared with the state-of-the-art centralized approach [30] which utilizes the entire dataset.

of crowdsourced distributed learning for mobile malware detection is confirmed as a valid alternative to centralized approaches, even with a limited number of participating clients.

5.4 Effect of label noise

Since the clients are not always able to correctly label the applications, the performance of the system was tested with varying levels of label noise with variable numbers of clients. The tested noise levels span from 0% to 40% with a step of 10%. The results are shown in Figure 11. It is worth noting that, with the exception of the "First" strategy and CDA-FedAvg, all the methods are quite robust to label noise, especially as the number of clients increases. By increasing the frequency of model updates, M2FD is able to maintain stable performance even with high levels of label noise. Up to 20% noise level, on average M2FD can still avoid updating the model half of the time, while maintaining a performance comparable to the "Always" baseline. It is worth noting that, with at least 100 clients, M2FD is able to maintain acceptable performances (F1-score> 0.8) even with 40% noise, supporting the applicability of the system in real-world scenarios where non-expert users have a high probability of mistaking malware for benign applications.

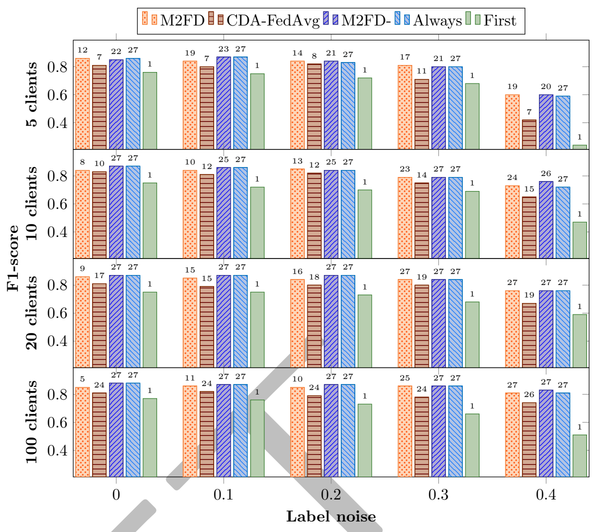


Figure 11: Average performance of the tested methods with varying levels of label noise and number of clients. The number of times the global model is updated is shown above each bar.

5.5 Impact of malware diffusion

The KronoDroid dataset contains roughly equal numbers of malware and benign samples. This balance is not fully representative of the ratio found in in-the-wild scenarios, where the number of benign applications is significantly higher than the number of malwares, which is estimated between 6% and 18.8% [47]. In other contexts, such as Windows software, the malware diffusion is typically more relevant. Thus, to ascertain the wider applicability of the M2FD in realistic scenarios with different malware ratios, the performance of the system was tested by varying the malware ratio from 10% to 50%. The obtained results, summarized in Figure 12, prove that M2FD is robust to changes in the malware ratio, maintaining comparable overall performances in all the considered scenarios. The 0.84 F1-score obtained by M2FD in a scenario with 10% of malware is close to that obtained on the Malware Hunter 100k dataset, (equal to 0.81) as previously described in section 5.2, since this dataset has a similar malware/goodware ratio. Such results confirm the possibility of using M2FD in real-world scenarios.

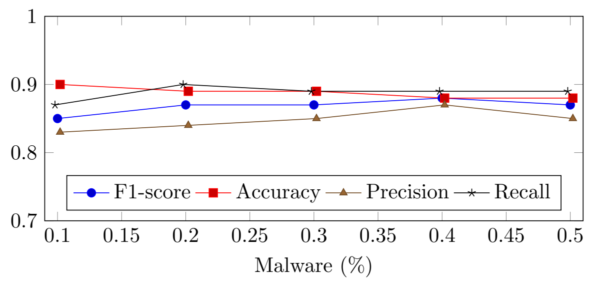


Figure 12: Average performance of M2FD by varying the ratio of malware with respect to the considered applications.

5.6 Influence of user confidence

Finally, to account for the potential variability in user confidence during labeling, the performance of the system was tested with varying value of the *confidence* γ, ranging from 0 to 1.0 in steps of 0.1. These values of γ were tested in scenarios with 100 clients and with different levels of label noise, from 0% to 40% in 10% steps. The results of these evaluations are shown in Figure 13.

The "First" strategy is significantly affected by lower user confidence, losing up to 0.33 points in F1-score when the confidence is reduced from 1.0 to 0.0. CDA-FedAvg is also affected by lower user confidence, because it relies on the certainty of the local model predictions, and therefore lower user confidence leads to more updates. Those updates, however, are not strictly tied to the actual concept drift, as the label fuzziness hides the effect of the drift, and many updates are false positives. M2FD, instead, relies on an intermediate data representation that is more robust to fuzziness and is able to maintain a stable performance even with lower user confidence. With $\gamma > 0.2$, on top of matching the performance of the "Always" strategy, M2FD is able to noticeably reduce the number of updates, especially when the noise level is low. If the confidence level is too low, the system may be subject to more frequent updates, while maintaining comparable performances. At higher noise levels, the impact of user confidence is extremely limited, since noisy labels already have a significantly reduced information content.

5.7 Domain transferability

Although M2FD has been designed to perform malware detection in mobile environments, its architecture and algorithms can be applied to other domains where concept drift is a significant issue and collecting data is a challenging task. In particular, M2FD can be successfully adopted in any scenario where:

- the observed data are characterized by a non-negligible concept drift;
- centralized data collection and annotation are not feasible due to privacy or data availability concerns;
- a central server is available with sufficient computing resources to train a model, even if this happen rarely;
- non-expert users often interact with new apps but are still able to provide better labels than random ones.

Some domains, such as IoT, while meeting the first two requirements, and in many cases the third, may not satisfy the fourth requirement, since most of the data generated by IoT devices is not typically inspected by users. One scenario that meets all the described requirements is the detection of malware within the Windows operating system. Indeed, here, malwares frequently change, collecting samples representative of the current threat landscape is challenging, modern personal computers have adequate computational resources to train a model, and users can often notice if their device is infected by malware belonging to categories such as ransomware, adware, or cryptojackers.

In order to assess the system's performance in the Windows domain, a set of experiment was conducted on the EMBER dataset [7] (composed by the EMBER 2017 v2 and EMBER 2018 subsets), containing 255,499 benign and 148,231 malicious executables. Since the goodware/malware ratio in Windows software is not easily quantified [47], the same ratio as in the original dataset is maintained.

The obtained results are shown in Figure 14. The "First" strategy is mostly stable during the first year, albeit slightly lower than the "Always" strategy. In the second year, the performance drops sharply and keeps decreasing. This behavior confirms the presence of concept drift in the dataset. The sharp drop, which is shared by all the approaches, is consistent

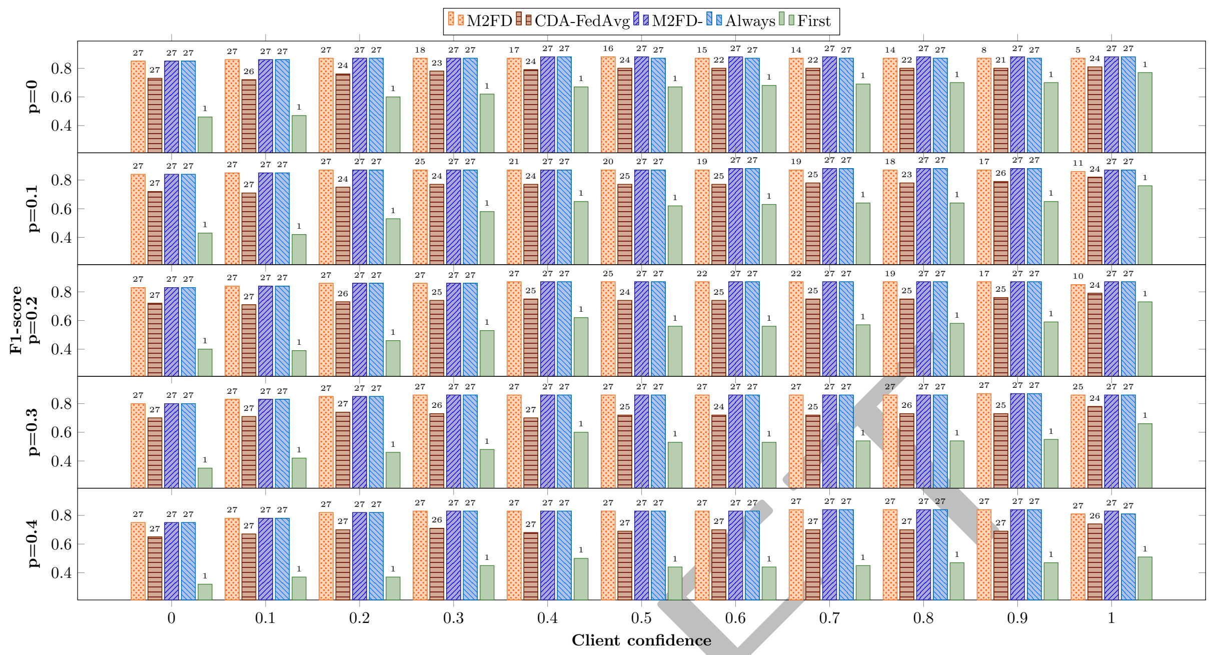


Figure 13: Average performance of the tested methods with varying levels of label noise (p) and user confidence (γ). The number of times the global model is updated is shown above each bar.

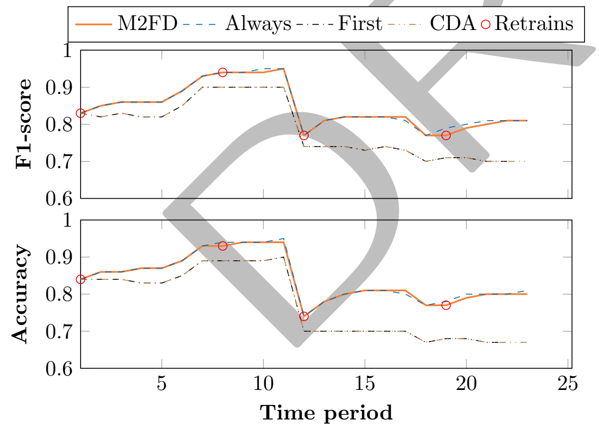


Figure 14: Average performance of the tested methods on the EMBER Windows malware dataset. M2FD only requires 4 updates (circled in red) over the 24 months, maintaining a performance matching the "Always" strategy.

with the observations in [7], whose authors note that the 2018 dataset deliberately contains more elusive malware samples. Despite clear signs of concept drift in this dataset, the CDA approach fails to recognize any drift, resulting is a behavior equivalent to the "First" strategy. M2FD, instead, detects the drift between the 2017 and 2018 datasets, and one additional drift in each period, resulting in a total of 4 updates over the 24 months. Despite training only 1/6 of the time, the accuracy and F1-score of M2FD closely matches those of the "Always" strategy, proving the robustness of M2FD and its potential for domain transferability.

5.8 Discussion and limitations

As shown in the experimental results, M2FD, through a distributed and privacy-preserving approach, is able to detect malware on mobile devices with performance comparable to state-of-the-art centralized approaches.

The system works also when only few clients are participating, and it is robust against varying levels of label noise, user confidence, and malware ratios. As long as the participating clients are reliable sources of information, M2FD can also significantly reduce the number of model retrainings required to keep the system up-to-date. M2FD increases the frequency of the model updates if the labels are very noisy, the confidence level of the users is low, or too few clients are participating to be a representative sample of the application ecosystem.

Nevertheless, if the participating clients cannot provide good enough labels (e.g., for being unable to detect stealthy malware), resulting in $p > 0.5$, the detection model would be provided with more noise than correct information. In this case, it might be necessary to introduce trusted expert users to provide high-quality labels, correcting the labels provided by the non-expert users in a semi-supervised learning setting. If expert users are available, they may also annotate the applications in a more fine-grained way, for instance with malware family labels, enabling multiclass classification. Furthermore, the decentralized approach of M2FD might be vulnerable to adversarial attacks, where a malicious client could intentionally provide incorrect labels to the server to degrade the model's performance. This is an orthogonal issue to the one addressed in this work, and the existing literature on FL security provides several solutions to mitigate this threat [63].

6 Conclusions

Malware detection on mobile devices is a challenging task due to the rapid changes in the cyber threat landscape which quickly render detection models obsolete. A key challenge in developing effective malware detection systems is obtaining labeled up-to-date data to update detection models. To address the data scarcity issue, crowdsourcing can be a valuable solution by leveraging the collective intelligence of mobile device users. However, the crowdsourcing approach must respect the privacy constraints of the user data, who might be unwilling to directly share their applications with a centralized entity. To this end, this paper presents M2FD, a novel malware detection system for mobile devices that leverages Federated Learning to overcome obstacles posed by concept drift and highly dynamic client datasets, without compromising user privacy. This paper first introduces a realistic model of client behavior, which is used to formally represent the installation and removal of applications from mobile devices, as well as the labeling of applications by non-expert users, and then presents a solution to the problem of detecting concept drift to update the detection model. The M2FD algorithm addresses the challenges associated with concept drift in mobile device malware detection while minimizing the computational burden on individual clients originating from frequent model retraining. M2FD leverages high-level robust features to represent the statistical distribution of the input data. The change in distribution over time is monitored by comparing the similarity of the new samples to the old ones through statistical hypothesis testing. If the difference is statistically significant and the effect size is large, the global model is retrained. The experimental results show that M2FD effectively balances the detection of concept drift to maintain satisfactory performances over time, and the minimization of computational burdens on the participating clients, thereby enhancing its overall robustness and efficiency. This work represents a significant step forward in the development of effective crowdsourced mobile malware detection systems, with the potential to be extended to other domains.

Acknowledgements

This work was partially funded by the European Union Next-Generation EU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3) – Progetto "Future Artificial Intelligence - FAIR" PE00000013, CUP J73C24000060007

References

[1] S. Abdulrahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE Internet of Things Journal*, 8(7): 5476–5497, Apr. 2021. ISSN 2327-4662. doi: 10.1109/JIOT.2020.3030072.

[2] R. A. Alexander and D. M. Govern. A new and simpler approximation for ANOVA under variance heterogeneity. *Journal of Educational Statistics*, 19 (2):91–101, 1994. doi: 10.3102/10769986019002091.

[3] E. Alkhateeb, A. Ghorbani, and A. Habibi Lashkari. A survey on run-time packers and mitigation

techniques. *International Journal of Information Security*, pages 1–27, 2023. doi: 10.1007/s10207-023-00759-y.

[4] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. Are your training datasets yet relevant? In F. Piessens, J. Caballero, and N. Bielova, editors, *Engineering Secure Software and Systems*, page 51–67, Cham, 2015. Springer International Publishing. ISBN 978-3-319-15618-7. doi: 10.1007/978-3-319-15618-7_5.

[5] A. Alzaidi, S. Alshehri, and S. M. Buhari. DroidRista: a highly precise static data flow analysis framework for android applications. *International Journal of Information Security*, 19(5): 523–536, 2020. doi: 10.1007/s10207-019-00471-w.

[6] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer. Dldroid: Deep learning based android malware detection using real devices. *Computers & Security*, 89: 101663, 2020. ISSN 0167-4048. doi: 10.1016/j.cose.2019.101663. URL `https://www.sciencedirect.com/science/article/pii/S0167404819300161`.

[7] H. S. Anderson and P. Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[8] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823, San Francisco, CA, 2022. IEEE, IEEE. doi: 10.1109/SP46214.2022.9833659.

[9] F. Bayram, B. S. Ahmed, and A. Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632, 2022. ISSN 0950-7051. doi: 10.1016/j.knosys.2022.108632. URL `https://www.sciencedirect.com/science/article/pii/S0950705122002854`.

[10] M. L. Bernardi, M. Cimitile, D. Distante, F. Martinelli, and F. Mercaldo. Dynamic malware detection and phylogeny analysis using process mining. *International Journal of Information Security*, 18: 257–284, 2019. doi: 10.1007/s10207-018-0415-3.

[11] H. Bostani and V. Moonsamy. Evadedroid: A practical evasion attack on machine learning for black-box android malware detection. *Computers & Security*, 139:103676, 2024. doi: 10.1016/j.cose.2023.103676.

[12] H. Bragança, V. Rocha, L. Barcellos, E. Souto, D. Kreutz, and E. Feitosa. Android malware detection with mh-100k: An innovative dataset for advanced research. *Data in Brief*, 51:109750, 2023. doi: 10.1016/j.dib.2023.109750.

[13] P. Burda, L. Allodi, and N. Zannone. Don't forget the human: a crowdsourced approach to automate response and containment against spear phishing attacks. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 471–476. IEEE, 2020. doi: 10.1109/EuroSPW51379.2020.00069.

[14] F. E. Casado, D. Lema, M. F. Criado, R. Iglesias, C. V. Regueiro, and S. Barro. Concept drift detection and adaptation for federated and continual learning. *Multimedia Tools and Applications*, pages 1–23, 2022. doi: 10.1007/s11042-021-11219-x.

[15] N. Cassavia, L. Caviglione, M. Guarascio, A. Liguori, G. Manco, and M. Zuppelli. A federated approach for detecting data hidden in icons of mobile applications delivered via web and multiple stores. *Social Network Analysis and Mining*, 13 (1):114, 2023. doi: 10.1007/s13278-023-01121-9.

[16] F. Ceschin, M. Botacin, H. M. Gomes, F. Pinagé, L. S. Oliveira, and A. Grégio. Fast & Furious: On the modelling of malware detection as an evolving data stream. *Expert Systems with Applications*, 212:118590, 2023. ISSN 0957-4174. doi: 10.1016/j.eswa.2022.118590.

[17] A. Chaudhuri, A. Nandi, and B. Pradhan. A dynamic weighted federated learning for android malware classification. In R. Kumar, A. K. Verma, T. K. Sharma, O. P. Verma, and S. Sharma, editors, *Soft Computing: Theories and Applications*, pages 147–159, Singapore, 2023. Springer Nature Singapore. ISBN 978-981-19-9858-4. doi: 10.1007/978-981-19-9858-4_13.

[18] Y. Chen, Z. Ding, and D. Wagner. Continuous Learning for Android Malware Detection. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1127–1144, Anaheim, CA, Aug. 2023. USENIX Association. ISBN 978-1-939133-37-3. URL `https://www.usenix.org/conference/usenixsecurity23/presentation/chen-yizheng`.

[19] T. Chow, Z. Kan, L. Linhardt, L. Cavallaro, D. Arp, and F. Pierazzi. Drift forensics of malware classifiers. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 197–207, 2023. doi: 10.1145/3605764.362391.

[20] J. Cohen. *Statistical power analysis for the behavioral sciences*. Academic press, 2013. doi: 10.4324/9780203771587.

[21] F. Concone, A. De Paola, G. Lo Re, and M. Morana. Twitter analysis for real-time malware discovery. In *2017 AEIT International Annual Conference (2017 AEIT)*, Cagliari, Italy, sep 2017. doi: 10.23919/AEIT.2017.8240551.

[22] A. A. Darem, F. A. Ghaleb, A. A. Al-Hashmi, J. H. Abawajy, S. M. Alanazi, and A. Y. Al-Rezami. An adaptive behavioral-based incremental batch learning malware variants detection model using concept drift detection and sequential deep learning. *IEEE Access*, 9:97180–97196, 2021. doi: 10.1109/ACCESS.2021.3093366.

[23] A. De Paola, S. Gaglio, G. Lo Re, and M. Morana. A hybrid system for malware detection on big data. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 45–50, 4 2018. doi: 10.1109/INFCOMW.2018.8406963.

[24] A. Desnos and G. Gueguen. Androguard documentation. *Obtenido de Androguard*, 2018. URL `https://androguard.readthedocs.io/en/latest/`.

[25] K. Divya and V. K. Kumar. Comparative analysis of smart phone operating systems android, apple ios and windows. *International Journal of Scientific Engineering and Applied Science (IJSEAS)*, 2(2):432–439, 2016. URL `https://ijseas.com/volume2/v2i2/ijseas20160253.pdf`.

[26] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*, pages 286–295. Springer, 2004. doi: 10.1007/978-3-540-28645-5_29.

[27] B. Gładysz. Fuzzy-probabilistic pert. *Annals of Operations Research*, 258:437–452, 2017. doi: 10.1007/s10479-016-2315-0.

[28] A. Guerra-Manzanares and H. Bahsi. On the relativity of time: Implications and challenges of data drift on long-term effective android malware detection. *Computers & Security*, 122:102835, 2022. doi: 10.1016/j.cose.2022.102835.

[29] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm. KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization. *Computers & Security*, 110:102399, 2021. ISSN 0167-4048. doi: 10.1016/j.cose.2021.102399.

[30] A. Guerra-Manzanares, M. Luckner, and H. Bahsi. Android malware concept drift using system calls: Detection, characterization and challenges. *Expert Systems with Applications*, 206:117200, Nov 2022. ISSN 0957-4174. doi: 10.1016/j.eswa.2022.117200.

[31] A. Guerra-Manzanares, H. Bahsi, and M. Luckner. Leveraging the first line of defense: a study on the evolution and usage of android security permissions for enhanced android malware detection. *Journal of Computer Virology and Hacking Techniques*, 19 (1):65–96, Mar. 2023. ISSN 2263-8733. doi: 10.1007/s11416-022-00432-3.

[32] S. Hamzenejadi, M. Ghazvini, and S. Hosseini. Mobile botnet detection: a comprehensive survey. *International Journal of Information Security*, 22(1): 137–175, 2023. doi: 10.1007/s10207-022-00624-4.

[33] H. Han, S. Lim, K. Suh, S. Park, S.-j. Cho, and M. Park. Enhanced android malware detection: An SVM-based machine learning approach. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 75–81. IEEE, 2020. doi: 10.1109/BigComp48618.2020.00-96.

[34] F. Jejdling, K. Kalliorinne, S. Thorsell, A. Maharaj, E. Lartey, F. Khan, H. Baur, G. Blennerud, F. Burstedt, W. Chaisatien, M. Karikytö, A.-M. Kåstedt, P. Lindberg, M. Martinsson, R. Hemi Mataira, L. Mattila, A. Mehta, F. Müller, R. Shekhar Pandey, and L. Sand. Ericsson mobility report june 2022. 2022. URL `https://www.ericsson.com/49d3a0/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-june-2022.pdf`.

[35] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016. doi: 10.48550/arXiv.1610.05492.

[36] K. Kong, Z. Zhang, Z.-Y. Yang, and Z. Zhang. FCSCNN: Feature centralized Siamese CNN-based android malware identification. *Computers & Security*, 112:102514, 2022. doi: 10.1016/j.cose.2021.102514.

[37] G. Lee and T. S. Raghu. Determinants of mobile apps' success: Evidence from the app store

market. *Journal of Management Information Systems*, 31(2):133–170, 2014. doi: 10.2753/MIS0742-1222310206.

[38] P. Liu, W. Wang, X. Luo, H. Wang, and C. Liu. NSDroid: efficient multi-classification of android malware using neighborhood signature in local function call graphs. *International Journal of Information Security*, 20:59–71, 2021. doi: 10.1007/s10207-020-00489-5.

[39] Z. Liu, R. Wang, N. Japkowicz, D. Tang, W. Zhang, and J. Zhao. Research on unsupervised feature learning for android malware detection based on restricted boltzmann machines. *Future Generation Computer Systems*, 120:91–108, 2021. doi: 10.1016/j.future.2021.02.015.

[40] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019. doi: 10.1109/TKDE.2018.2876857.

[41] N. Lu, G. Zhang, and J. Lu. Concept drift detection via competence models. *Artificial Intelligence*, 209:11–28, 2014. doi: 10.1016/j.artint.2014.01.001.

[42] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. URL `https://proceedings.mlr.press/v54/mcmahan17a.html`.

[43] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso. Efficient concept drift handling for batch android malware detection models. *Pervasive and Mobile Computing*, 96:101849, 2023. ISSN 1574-1192. doi: 10.1016/j.pmcj.2023.101849. URL `https://www.sciencedirect.com/science/article/pii/S1574119223001074`.

[44] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):157–175, 2017. doi: 10.1109/TETCI.2017.2699220.

[45] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Trans. Priv. Secur.*, 22(2), apr 2019. ISSN 2471-2566. doi: 10.1145/3313391. URL `https://doi.org/10.1145/3313391`.

[46] D. Papamartzivanos, D. Damopoulos, and G. Kambourakis. A cloud-based architecture to crowdsource mobile app privacy leaks. In *Proceedings of the 18th panhellenic conference on informatics*, pages 1–6, 2014. doi: 10.1145/2645791.2645799.

[47] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX security symposium (USENIX Security 19)*, pages 729–746, 2019. URL `https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury`.

[48] T. Petsas, A. Papadogiannakis, M. Polychronakis, E. P. Markatos, and T. Karagiannis. Measurement, modeling, and analysis of the mobile app ecosystem. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(2):1–33, 2017. doi: 10.1145/2993419.

[49] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944, 2015. doi: 10.1145/2783258.2783359.

[50] H. Razeghi Borojerdi and M. Abadi. Malhunter: Automatic generation of multiple behavioral signatures for polymorphic malware detection. In *ICCKE 2013*, pages 430–436, Mashhad, Iran, 2013. IEEE. doi: 10.1109/ICCKE.2013.6682867.

[51] O. Rutz, A. Aravindakshan, and O. Rubel. Measuring and forecasting mobile game app engagement. *International Journal of Research in Marketing*, 36(2):185–199, 2019. doi: 10.1016/j.ijresmar.2019.01.002.

[52] X. Sáez-de Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza. Clustered federated learning architecture for network anomaly detection in large scale heterogeneous iot networks. *Computers & Security*, 131:103299, 2023. doi: 10.1016/j.cose.2023.103299.

[53] S. Seneviratne, A. Seneviratne, P. Mohapatra, and A. Mahanti. Predicting user traits from a snapshot of apps installed on a smartphone. *ACM SIGMOBILE Mobile Computing and Communications Review*, 18(2):1–8, 2014. doi: 10.1145/2636242.2636244.

[54] F. Shang, Y. Li, X. Deng, and D. He. Android malware detection method based on naive bayes

and permission correlation algorithm. *Cluster Computing*, 21(1):955–966, 2018. doi: 10.1007/s10586-017-0981-6.

[55] Y. Shanmugarasa, H.-y. Paik, S. S. Kanhere, and L. Zhu. A systematic review of federated learning from clients' perspective: challenges and solutions. *Artificial Intelligence Review*, pages 1–55, 2023. doi: 10.1007/s10462-023-10563-8.

[56] A. Silva and J. Simmonds. Behaviordroid: monitoring android applications. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16, page 19–20, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341783. doi: 10.1145/2897073.2897121.

[57] S. G. Stats. Mobile Operating System Market Share Worldwide, 2024. URL `https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202401-202410`.

[58] C. Steglich, S. Marczak, L. P. Guerra, L. H. Mosmann, M. Perin, F. Figueira Filho, and C. de Souza. Revisiting the mobile software ecosystems literature. In *2019 IEEE/ACM 7th International Workshop on Software Engineering for Systems-of-Systems (SESoS) and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES)*, pages 50–57, 2019. doi: 10.1109/SESoS/WDES.2019.00015.

[59] A. L. Suárez-Cetrulo, D. Quintana, and A. Cervantes. A survey on machine learning for recurring concept drifting data streams. *Expert Systems with Applications*, 213:118934, 2023. doi: 10.1016/j.eswa.2022.118934.

[60] K. Sugunan, T. Gireesh Kumar, and K. Dhanya. Static and dynamic analysis for android malware detection. In *Advances in Big Data and Cloud Computing*, pages 147–155, Singapore, 2018. Springer, Springer. doi: 10.1007/978-981-10-7200-0_13.

[61] L. Tang, X. Chen, S. Wen, L. Li, M. Grobler, and Y. Xiang. Demystifying the evolution of android malware variants. *IEEE Transactions on Dependable and Secure Computing*, 2023. doi: 10.1109/TDSC.2023.3325912.

[62] Y. Tong, Y. Wang, and D. Shi. Federated learning in the lens of crowdsourcing. *IEEE Data Eng. Bull.*, 43(3):26–36, 2020.

[63] Y. Wan, Y. Qu, W. Ni, Y. Xiang, L. Gao, and E. Hossain. Data and model poisoning backdoor attacks on wireless federated learning, and the defense mechanisms: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 26(3):1861–1897, 2024. doi: 10.1109/COMST.2024.3361451.

[64] N. Wang, Y. Xiao, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou. FLARE: defending federated learning against model poisoning attacks via latent space representations. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 946–958, 2022. doi: 10.1145/3488932.3517395.

[65] R. Wang, J. Gao, and S. Huang. AIHGAT: A novel method of malware detection and homology analysis using assembly instruction heterogeneous graph. *International Journal of Information Security*, pages 1–21, 2023. doi: 10.1007/s10207-023-00699-7.

[66] S. Wares, J. Isaacs, and E. Elyan. Data stream mining: methods and challenges for handling concept drift. *SN Applied Sciences*, 1:1–19, 2019. doi: 10.1007/s42452-019-1433-0.

[67] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah. Android malware detection based on system call sequences and LSTM. *Multimedia Tools and Applications*, 78:3979–3999, 2019. doi: 10.1007/s11042-017-5104-0.

[68] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu. Droidevolver: Self-evolving android malware detection system. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 47–62, Stockholm, Sweden, 2019. IEEE. doi: 10.1109/EuroSP.2019.00014.

[69] J. Yang, H. Li, L. He, T. Xiang, and Y. Jin. Mdadroid: A novel malware detection method by constructing functionality-api mapping. *Computers & Security*, page 104061, 2024. doi: 10.1016/j.cose.2024.104061.

[70] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2327–2344, Vancouver, BC, Canada, Aug. 2021. USENIX Association. ISBN 978-1-939133-24-3. URL `https://www.usenix.org/conference/usenixsecurity21/presentation/yang-limin`.

[71] F. Yates. The analysis of multiple classifications with unequal numbers in the different classes. *Journal of the American Statistical Association*, 29(185): 51–66, 1934. doi: 10.1080/01621459.1934.10502686.

[72] X. Zhang, M. Zhang, Y. Zhang, M. Zhong, X. Zhang, Y. Cao, and M. Yang. Slowing down the aging of learning-based malware detectors with api knowledge. *IEEE Transactions on Dependable and Secure Computing*, 20(2):902–916, 2023. doi: 10.1109/TDSC.2022.3144697.

[73] Y. Zhang, X. Luo, and H. Yin. Dexhunter: Toward extracting hidden code from packed android applications. In *Proc. ESORICS*, 2015.

[74] H.-J. Zhu, T.-H. Jiang, B. Ma, Z.-H. You, W.-L. Shi, and L. Cheng. HEMD: a highly efficient random forest-based malware detection framework for Android. *Neural Computing and Applications*, 30: 3353–3361, 2018. doi: 10.1007/s00521-017-2914-y.

[75] M. Zyout, R. Shatnawi, and H. Najadat. Malware classification approaches utilizing binary and text encoding of permissions. *International Journal of Information Security*, pages 1–26, 2023. doi: 10.1007/s10207-023-00712-z.