Partial Federated Primal-Dual Optimization for Network-Based Android Malware Detection

Mohammad Mashreghi
Department of Electrical & Computer Engineering
College of Engineering, University of Tehran
Tehran, 417935840, Iran
m.mashreghi@ut.ac.ir

Mohammad Hossein Badiei
Department of Electrical & Computer Engineering
College of Engineering, University of Tehran
Tehran, 417935840, Iran
mh.badiei@ut.ac.ir

***Abstract*—The proliferation of Android devices has led to a surge in malware attacks, posing significant threats to user privacy and security. Traditional machine learning approaches for malware detection often require centralized data collection, which raises privacy concerns and may not be scalable given the distributed nature of user data. Federated Learning (FL) offers a decentralized solution by enabling devices to collaboratively train a global model without sharing raw data. However, conventional FL methods like Federated Averaging (FedAvg) face significant limitations in communication efficiency, partial client participation, and non-convex optimization. These challenges result in increased overhead, unstable model updates, and reduced robustness, particularly in real-world applications like malware classification. In this study, we utilize a federated primal-dual optimization approach to enhance communication efficiency and model convergence in FL by leveraging the Alternating Direction Method of Multipliers (ADMM). Unlike traditional optimization methods for FL, ADMM is well-suited for solving non-convex composite optimization problems with non-smooth regularizers, making it robust to data heterogeneity and dynamic client participation. We evaluate our proposed method on widely used benchmark Android malware classification datasets (e.g. Malgenome, Drebin, and Tuandromd). Experimental results demonstrate that our algorithm outperforms traditional FL approaches (e.g. FedAvg and FedProx) in terms of accuracy, F1 score, AUC score, and false positive rate (FPR).**

***Index Terms*—Android malware detection, Partial federated learning, Primal-dual optimization framework, Distributed deep learning, Android cybersecurity.**

I. Introduction

Android has become one of the most widely used operating systems, making it a prime target for web-based malware attacks, with Android malware now accounting for over 98% of mobile threats and increasing significantly in recent years [1]. Unlike traditional computers, Android devices offer greater flexibility in app installation, introducing security risks that threaten both users and organizations [2]. While machine learning (ML) and deep learning (DL) have proven effective for malware detection, traditional approaches struggle with scalability due to the decentralized nature of user data and privacy concerns related to sensitive information such as location and online identifiers [3]. Addressing these challenges requires more efficient and privacy-preserving solutions for Android malware detection.

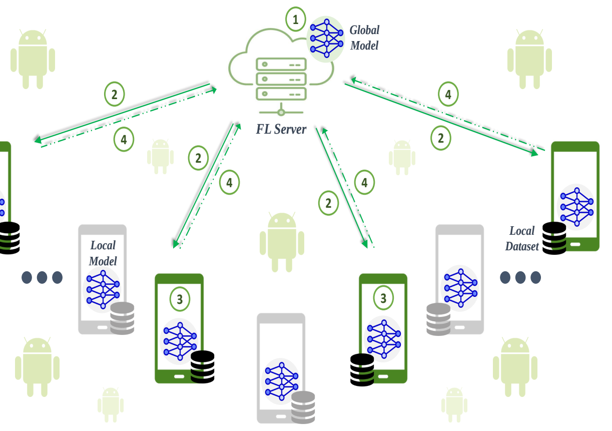


Fig. 1: FL follows a four-step process: ① the server initializes (for the first round) or aggregates (for subsequent rounds) and shares global model weights, ② these weights are distributed to active clients, ③ clients locally train the model on their private data, and ④ updated model parameters are sent back to the server, where they are aggregated to update the global model, repeating iteratively until convergence or optimal weights are achieved.

Federated Learning (FL) has gained significant attention as a decentralized approach for securing distributed environments, particularly in malware detection across Android and IoT networks [4], [5]. As shown in Fig. 1, by allowing multiple devices to collaboratively train a global model without sharing raw data, FL enhances privacy while utilizing the computational power of edge devices. This client-server framework enables model updates to be exchanged between participating devices and a central coordinator, ensuring privacy-preserving learning across heterogeneous edge devices such as smartphones [6], [7].

However, FL introduces several challenges distinct from conventional distributed learning. Communication bottlenecks emerge in large-scale networks or under bandwidth constraints, limiting efficient information exchange, particularly in devices with limited network capacity such as smartphones [8]. Additionally, FL faces challenges in achieving model convergence, which can negatively impact accuracy [9]. The diverse nature of edge devices, each with different processing capacities and connectivity conditions, can slow down global updates—a problem often referred to as the "straggler" effect in FL [10]. Moreover, due to fluctuating availability, only a subset of clients may participate in each training round, making it essential to design algorithms that are resilient to partial participation and asynchronous updates [11]. Overcoming these challenges is key to ensuring scalable and effective federated cybersecurity solutions.

Several methodologies have been proposed to improve the robustness of federated malware classification [12], [13]. Some approaches incorporate adversarial-resistant mechanisms, such as generative adversarial networks (GANs) or worst-case robust optimization techniques, to mitigate poisoning attacks that could compromise the integrity of the global model [14], [15]. However, these adversarial mechanisms present significant challenges in resource-constrained environments. Such GAN-based approaches often aim to improve model robustness by simulating adversarial scenarios; however, their limited fidelity in generating diverse perturbations, unstable training, and high computational demands hinder their effectiveness in FL-based malware detection [16], [17]. Worst-case robust optimization, while improving resilience, tends to rely on overly conservative assumptions, which can reduce detection accuracy. Additionally, these gradient-based approaches impose significant computational overhead, further straining resource-constrained devices and increasing training complexity in FL-based malware detection [18].

In addition, Byzantine-resilient aggregation techniques, such as Krum and Median filtering, have also been employed to prevent malicious updates from degrading model performance [19], [20]. However, these techniques face notable limitations in FL, as Krum's high computational complexity makes it inefficient for large-scale networks, while Median-based methods struggle with accurately filtering out adversarial updates in high-dimensional spaces [21]. Additionally, geometric median aggregation imposes significant computational overhead, prolonging training time and limiting scalability [22].

Additionally, deep neural networks, particularly convolutional neural networks (CNNs), are widely used in FL-based malware detection due to their ability to learn hierarchical representations, but their computational complexity poses challenges in resource-constrained federated environments, necessitating optimization techniques to mitigate overhead while maintaining high performance [13], [23]–[25]. In these frameworks, the models learn hierarchical features, making them effective for identifying malware patterns. By utilizing permission-based feature extraction and optimized feature selection, they enhance classification accuracy [26], [27]. However, CNNs face significant challenges in FL-based malware detection due to their high computational complexity, making them impractical for resource-limited devices. Their reliance on large, diverse datasets is problematic in federated environments, where data is distributed and often imbalanced, affecting model performance. Additionally, their lack of interpretability and the need for frequent updates to adapt to evolving malware strains further complicate deployment in bandwidth-constrained FL networks.

Moreover, LiM is another FL-based malware classification framework that keeps app data local and leverages a secure semi-supervised ensemble to enhance classification accuracy while preserving user privacy [28]. However, key challenges of this method include the inability to classify malware at the family level, as it does not account for the specific characteristics of malicious apps installed by clients. Additionally, its evaluation is limited to controlled experiments with a static set of users and predefined parameters, which may not fully capture real-world deployment complexities.

These advancements demonstrate the potential of FL-based frameworks in securing Android ecosystems against sophisticated cyber threats. However, several challenges persist in implementing FL-based cybersecurity frameworks for Android malware detection. Conventional optimization techniques often struggle with convergence inefficiencies in decentralized learning, particularly under constrained computational resources and limited communication budgets. The high frequency of model synchronization increases latency and energy consumption, making real-time threat detection challenging in resource-limited environments. Additionally, existing approaches face difficulties in balancing local and global objectives during training, leading to suboptimal parameter updates that degrade model performance. Furthermore, current methods lack adaptability to dynamic threat landscapes, requiring frequent retraining that is computationally expensive and impractical for edge devices. Addressing these challenges necessitates more efficient optimization frameworks capable of handling communication constraints, accelerating convergence, and enabling adaptive learning in federated cybersecurity applications.

In this work, we extend the primal-dual optimization framework to FL-based malware detection, leveraging its inherent advantages in adaptive optimization and efficient convergence. Our proposed method, built upon the FedADMM paradigm, integrates dual variables to regulate local training, effectively mitigating client drift without requiring extensive hyperparameter tuning. Compared to traditional FL approaches such as FedAvg and FedProx, our method achieves superior convergence rates with fewer communication rounds while maintaining a lower false positive rate (FPR) and a higher area under the curve (AUC), demonstrating its robustness in malware classification. Furthermore, our framework exhibits enhanced scalability, as increasing the number of participating clients leads to more accurate global model updates, surpassing alternative methods in both efficiency and predictive performance. Ultimately, these results underscore the potential of primal-dual optimization in constructing resilient and adaptive FL-based cybersecurity solutions.

The rest of the article is structured as follows. Section II provides the background. Section III outlines the proposed methodology. Section IV presents simulation results and discussions. Finally, Section V concludes with key takeaways.

II. Background

Before presenting the proposed methods, we first outline the structure of distributed Android applications to provide the necessary foundation for understanding the framework introduced in this article.

Android Applications. These are packaged as APK files, containing essential components such as the AndroidManifest.xml (defining runtime permissions and configurations), classes.dex (holding compiled code and execution logic), and resource eiles (UI definitions and multimedia content). In FL-based malware detection, analyzing these structured components enables efficient feature extraction while preserving privacy, facilitating decentralized learning without exposing raw application data.

Federated Learning. In FL, multiple clients, denoted as $i \in \{1, 2, \dots, n\}$, collaborate to train a global model while maintaining the privacy of their local datasets $\mathcal{D}_i$. The global optimization objective is to minimize the loss function $\mathcal{L}$ across all data distributions:

$$\min_{\bar{x}} \sum_{i=1}^{n} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \mathbb{E}\Big[\mathcal{L}\big(h_{\bar{x}}\big)\Big],$$

where $\bar{x}$ represents the global model parameters. In FedAvg, each client i independently updates its local model x_i based on its data and sends the model parameters to the central server. The server aggregates the updates by averaging:

$$\bar{x}^k = \frac{1}{n}\sum_{i=1}^{n} x_i^k.$$

This process ensures that the global model benefits from diverse data while preserving privacy.

In FedProx, the goal is to mitigate the effects of data heterogeneity by adding a proximal term to the local objective function. The local update for client i is adjusted by a regularization term:

$$x_i^k = \arg\min_{x_i}\Big\{\mathcal{L}\big(h_{x_i}\big) + \frac{\mu}{2}\,\|x_i - \bar{x}^{k-1}\|^2\Big\},$$

where μ controls the strength of regularization, helping the local model stay closer to the global model parameters. After local updates, the server aggregates the parameters as in FedAvg. This modification helps mitigate client drift and improves model convergence, leading to more stable and efficient global model training.

Malware Detection with FL. FL facilitates decentralized malware detection by enabling collaborative model training across distributed devices while maintaining data locality and privacy. It processes high-dimensional feature sets, including API call sequences, dynamic execution traces, and system logs, to detect anomalous behavior indicative of malware. However, FL introduces challenges such as client drift due to inconsistent local updates and substantial communication overhead from frequent model aggregation.

III. Methodology: Partial Federated Primal-Dual Optimization for Android Malware Detection

In this section, we describe how to apply the Federated ADMM (FedADMM) algorithm to detect Android malware in a FL setting. FedADMM is a primal-dual framework allowing partial client participation, which makes it especially suitable for scenarios where not all Android devices are available or willing to participate at each training round.

A. System Overview

To begin with, we consider a network of n Android devices, each holding local data that can be used to train a malware detection model. This local data may include features such as system calls, permissions, network usage patterns, or other behavioral indicators of installed applications. Our goal is to train a global classifier x that accurately distinguishes malicious apps from benign ones while preserving user privacy and minimizing communication overhead.

In FL, each device $i \in [n]$ has its own local dataset $\mathcal{D}_i$. We define a local loss function $f_i(\cdot)$ that measures how well the model x_i fits the local data $\mathcal{D}_i$. In many malware detection tasks, f_i could be a cross-entropy loss with regularization terms to control overfitting. In addition, we may include a global regularization or penalty term $g(\cdot)$ that encodes constraints or priors shared across all devices (e.g., weight decay). The overall objective is to collaboratively minimize the sum of local losses and the global term g, subject to the constraint that the global model x and local models x_i remain consistent.

B. Augmented Lagrangian Formulation

FedADMM uses an augmented Lagrangian to handle the coupling between local models x_i and the global model x. Specifically, for each client i, we define:

$$\mathcal{L}_i\big(x_i^k, x^k, z_i\big) = f_i\big(x_i^k\big) + g\big(x^k\big) + \langle z_i,\, x_i^k - x^k\rangle + \frac{\eta}{2}\,\big\|x_i^k - x^k\big\|^2,$$

where z_i is the dual variable associated with the constraint $x_i = x$, and $\eta > 0$ is a penalty parameter that controls the strength of the coupling. By penalizing the difference $\|x_i^k - x^k\|$ and introducing the dual variable z_i, FedADMM iteratively encourages local models to align with the global model while allowing partial client participation at each round.

C. Optimization Framework

We now present the FedADMM procedure step-by-step for Android malware detection. Algorithm 1 matches the exact structure we use in our framework. Below, we explain each stage of the algorithm and how it applies to our malware detection scenario:

1) **Initialization:** The server initializes a global model x^0 (e.g., a neural network or other classifier) and sets $\bar{x}^0 = x^0$. Each Android device i initializes its local model x_i^0 to the same value x^0. The dual variables z_i^0 are set to zero. We also set parameters: a penalty multiplier γ, the maximum number of communication rounds K, and any local convergence tolerances $\epsilon_{i,0}$.

Algorithm 1 Federated ADMM Algorithm (FedADMM)

```
1: **Initialize** $x^0$, $\gamma > 0$, $K$, and tolerances $\epsilon_{i,0}(i \in [n])$
2: Initialize the server with $\bar{x}^0 = x^0$
3: Initialize all clients with $z_i^0 = 0$ and $x_i^0 = \hat{x}^0 = x^0$
4: for $k = 0, 1, \ldots, K$ do
5:     Randomly sample $S_k \subseteq [n]$ with size $S$
6:     $\triangleright$ Client side:
7:     for each client $i \in S_k$ do
8:         Receive $\bar{x}^k$ from the server
9:         $x_i^{k+1} \approx \arg\min_{x_i} \mathcal{L}_i\big(x_i, \bar{x}^k, z_i^k\big)$
10:        $z_i^{k+1} = z_i^k + \gamma\left(x_i^{k+1} - \bar{x}^k\right)$    // Dual update
11:        $\hat{x_i}^{k+1} = x_i^{k+1} - \frac{1}{\eta} z_i^{k+1}$
12:        Send $\hat{x_i}^{k+1}$ to the server
13:    end for
14:    $\triangleright$ Server side:
15:    $\widetilde{x}^{k+1} = \frac{1}{|S_k|} \sum_{i \in S_k} x_i^{k+1}$
16:    $\bar{x}^{k+1} = \mathrm{prox}_{\gamma/n}\big(\widetilde{x}^{k+1}\big)$
17: end for
```

2) **Partial Participation:** At the start of round k, the server randomly selects a subset $S_k \subseteq [n]$ of size S. Only the selected devices will participate in this round. This step is crucial in real-world Android environments, where devices may have limited battery or intermittent network connectivity.
3) **Local Computation:** Each selected device $i \in S_k$ receives the current global model $\bar{x}^k$ from the server. Locally, device i solves (or approximates) the subproblem

$$\begin{aligned} x_i^{k+1} &= \arg\min_{x_i} \mathcal{L}_i(x_i, \bar{x}^k, z_i^k) \\ &\approx \arg\min_{x_i} \Big[f_i(x_i) + g(\bar{x}^k) \\ &\quad + \langle z_i^k, x_i - \bar{x}^k \rangle + \frac{\eta}{2}\|x_i - \bar{x}^k\|^2 \Big]. \end{aligned}$$

In practice, $f_i(\cdot)$ is the local malware detection loss on device i (e.g., cross-entropy over benign/malicious labels). This step may be performed by a few epochs of gradient descent or another suitable optimization method, given the device's computational constraints.
4) **Dual Update:** The device updates its dual variable z_i to account for the difference between its local model and the global model:

$$z_i^{k+1} = z_i^k + \gamma\left(x_i^{k+1} - \bar{x}^k\right).$$

This step is a key feature of ADMM, helping to balance local and global objectives.
5) **Constructing the Update to Send:** To reduce communication overhead, device i may send a modified update $\hat{x}_i^{k+1}$ back to the server:

$$\hat{x}_i^{k+1} = x_i^{k+1} - \frac{1}{\eta} z_i^{k+1}.$$

This representation encodes both the local model and dual information in a single vector.
6) **Server Aggregation:** The server aggregates the local models from all participating devices $i \in S_k$:

$$\widetilde{x}^{k+1} = \frac{1}{|S_k|} \sum_{i \in S_k} x_i^{k+1}.$$

In a malware detection context, this aggregation forms a new "averaged" model capturing knowledge from all participating devices.
7) **Proximal Update:** Finally, the server applies a proximal operator (depending on γ and possibly g) to the aggregated model:

$$\bar{x}^{k+1} = \mathrm{prox}_{\gamma/n}\big(\widetilde{x}^{k+1}\big),$$

which may include any additional global regularization or constraints (e.g., $g(\cdot)$). This produces the updated global model $\bar{x}^{k+1}$.
8) **Next Round:** The process repeats for K rounds or until convergence criteria are met. Over time, the global model $\bar{x}^k$ improves its ability to classify malicious vs. benign apps, leveraging data from the distributed network of Android devices.

IV. Simulation Results and Discussion

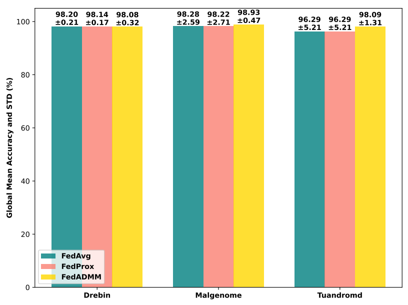
To assess the performance of our proposed method, we carried out extensive simulations across a variety of configurations. These experiments were specifically designed to examine convergence behavior, communication efficiency, and scalability in comparison to established federated learning baselines. The results presented in this section highlight the advantages of our approach and explore its potential implications for federated malware detection.

A. Dataset Description

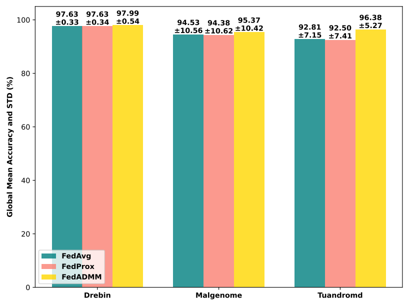
Our evaluation is conducted on three benchmark datasets for Android malware detection, described as follows:
Malgenome. This dataset contains features extracted from 3,799 Android application samples, comprising 2,539 benign apps and 1,260 malware samples, collected from the Android Malware Genome Project [29]. It includes 215 features in total.
Drebin. Sourced from the Drebin Project [30], this dataset features data from 15,036 app samples, with 9,476 classified as benign and 5,560 as Android malware, and consists of 215 features.
Tunadromd. Detailed in [31], this dataset comprises features from 4,465 app samples, including 903 benign apps and 3,565 Android malware samples, and contains 241 features overall.

B. Experimental Setup

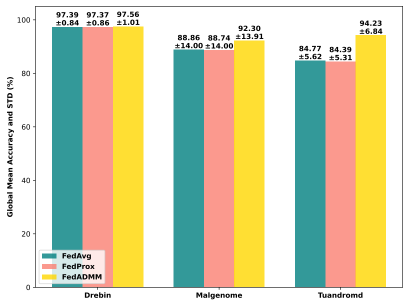
This subsection outlines the experimental framework of our study, including machine configuration, software environment, classifier architecture, optimization strategy, and parameter settings. Details are provided below:
Machine Configuration. The experiments were conducted on a device running Windows 11 Pro Education (OS Build



(a) 5-clients

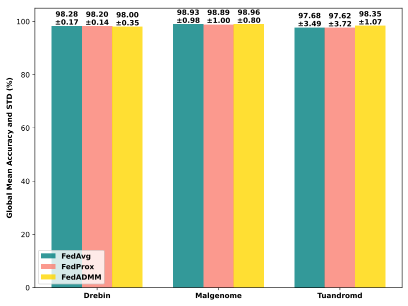


(b) 10-clients

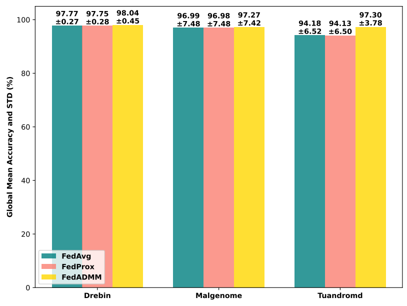


(c) 15-clients

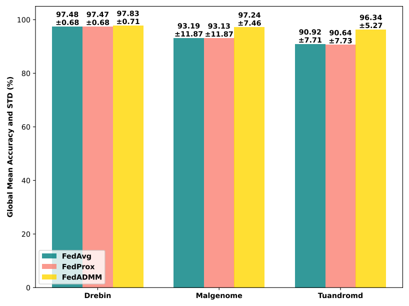
Fig. 2: Performance comparison of FedAvg, FedProx, and FedADMM in terms of model accuracy across the datasets over 10 communication rounds.



(a) 5-clients



(b) 10-clients



(c) 15-clients

Fig. 3: Performance comparison of FedAvg, FedProx, and FedADMM in terms of model accuracy across the datasets over 20 communication rounds.

22631.4890) a 64-bit operating system. The system is powered by a 13th Gen Intel® Core™ i7-13620H processor, operating at 2.40 GHz. It is equipped with 16.0 GB of RAM. This configuration offers robust computational resources suitable for deep learning tasks and federated learning experiments.

Software Development Environment. The FedADMM algorithm was implemented using Python 3.10.11, paired with PyTorch 2.6.0+cpu as the primary deep learning framework for model development and training. The CPU-only version of PyTorch was selected to align with the hardware constraints of the system, ensuring efficient execution without GPU dependencies. Additional libraries, such as NumPy and Pandas, were utilized for data preprocessing and management, ensuring a streamlined development pipeline.

Base Classifier and Optimizer. The base classifier is a 4-layer feed-forward neural network tailored for binary classification (benign vs. malware), selected for two key reasons: (1) to enable fast training and efficient communication between clients in the federated learning setup, minimizing latency during model updates, and (2) to maintain a compact model size suitable for deployment on resource-constrained mobile phone devices. The architecture consists of:

- *First Hidden Layer*: 200 neurons, designed to capture a broad range of features from the input data.
- *Second Hidden Layer*: 100 neurons, refining the learned representations.
- *Third Hidden Layer*: 50 neurons, further compressing the feature space for efficient classification.
- *Output Layer*: A single neuron with a sigmoid activation function, outputting probabilities for the binary classes (benign or malware).

The number of neurons and layers was determined through a trial-and-error approach, as no standardized method exists for automatically optimizing such architectures in this context. The hidden layers employ the ReLU (Rectified Linear Unit) activation function to introduce non-linearity and mitigate vanishing gradient issues, while the sigmoid activation in the output layer is chosen to suit the binary classification task. For training, we adopted the Stochastic Gradient Descent (SGD) optimizer, widely favored in federated learning due to its simplicity, efficiency, and ability to handle distributed updates effectively. The choice of SGD also aligns with its proven performance in similar deep learning applications, and it integrates seamlessly with PyTorch's optimization utilities.

Parameter Setup. The FedADMM model was trained using a partial federated learning approach, where, in each training round, only 80% (0.8) of all available clients are randomly selected to participate and aggregate their updates. This strategy enhances scalability and simulates real-world scenarios where not all clients are available simultaneously. The training process was configured with the following hyperparameters:

- *Batch Size*: 32, balancing computational efficiency and gradient stability.
- *Local Epochs*: 32 fixed for each client in training process.
- *Cross-Validation*: An 80-20 Hold-Out technique was employed, splitting the dataset into 80% for training and 20% for testing. This approach ensures a robust evaluation of model generalization and performance across all datasets.
- *Learning Rate*: Set to 0.0001 for the SGD optimizer, providing a steady convergence rate while avoiding overshooting.

Source Code Availability. The complete source code, along with detailed implementation notes and instructions for replication, is publicly accessible on GitHub at https://github.com/ArgAI-Lab/Partial-FedADMM-Android-Malware-Detection. This repository includes scripts for data preprocessing, model training, and evaluation, enabling full transparency and facilitating further research.

This experimental setup ensures that our FedADMM approach is both computationally feasible and rigorously evaluated, providing a solid foundation for the performance results discussed in subsequent sections.

Performance Metrics. The accuracy, F1-score, AUC, FPR, and specificity are metrics used to evaluate the performance of the classification model. For binary classification, these metrics are calculated in terms of positives and negatives

C. Discussion and Analysis

In this section, we discuss the experimental results obtained by our proposed federated primal dual approach and compare its performance with FedAvg and FedProx under different configurations. Specifically, we evaluated three client scenarios (5, 10, and 15 clients) with clients availability and two round settings (10 and 20 rounds). The average test accuracy of the global model is presented in Figures 2 and 3, while the specificity, F1-score, AUC, and FPR are reported in Tables I and II. The best results are highlighted in bold.

For the Drebin dataset, the FedADMM method has demonstrated similar or higher scores across all metrics (accuracy, specificity, F1 score, AUC score, and FPR) compared to the FedProx and FedAvg approaches. In the first comparison (see Table I and Figure 2a), where the number of rounds is 10 and the number of clients is 5, FedADMM improved specificity by approximately 0.2% and 0.25% compared to FedAvg and FedProx, respectively, while also reducing FPR by 0.09% and 0.016% in that order. For the scenario with 10 clients, the performance of FedADMM is superior to the other two methods in both the tables and figures, showing about a 0.3% increase in accuracy, slightly more than a 1.4% increase in specificity, and a 0.5% improvement in F1 score, along with an almost 1.38% reduction in FPR. In scenarios with 15 clients, the FedADMM approach exhibits trends similar to the 10-client case, albeit with smaller differences in the metric values. For 20 rounds (see Figure 3 and Table II), it can be observed that after 20 rounds the FPR of the FedADMM approach is lower than that of the other two methods. In the case of 10 clients, FedADMM achieves higher specificity and F1 score by 1.3% and 0.39%, respectively. This trend is repeated for the 15-client scenario, where FedADMM consistently produces better scores across the evaluated metrics.

For the Tuandromd dataset, in a 10-round setting with 5 clients, FedADMM achieves approximately 0.8% higher accuracy with a lower standard deviation. Moreover, it improves other metrics by 0.41% in specificity, 0.99% in F1-score, and 0.03% in AUC. In the 10-client setting, the accuracy is 0.84% higher than that of FedAvg, while the F1-score and AUC increase by about 2% and 0.21%, respectively. In the 15-client scenario, similar behavior is observed, with accuracy reaching 9.56% higher than that of the other two methods. For the 20-round setting, the mean test accuracy remains superior, with increases of 0.73%, 3.17%, and 6.34% for the 5-, 10-, and 15-client settings, respectively. In the 5-client scenario, all metrics are outstanding, whereas in the 10-client scenario, only the F1-score and AUC improve (by 1.73% and 0.2%, respectively), and in the 15-client scenario, the F1-score is 3.14% higher than that of the other methods.

For the Malgenome dataset, the FedADMM algorithm demonstrates robust performance across various settings. In the 10-round setting, FedADMM achieves a 0.65% increase in accuracy, a 3.45% improvement in specificity, and a 1.29% increase in F1-score, while the FPR decreases significantly from 4.40% to 0.93%. In the 10-client setting, accuracy is more than 0.84% higher compared to the other methods, with specificity improving by 3.51%, F1-score increasing by 1.83%, and FPR decreasing by 3.51%. In the 15-client scenario, although all metrics tend to decline across methods, the relative difference becomes more pronounced. Specifically, FedADMM shows a 2.46% higher accuracy, an 11.23% improvement in specificity, an increase in F1-score from 0.7142 to 0.7872, and an 11.35% reduction in FPR compared to the alternatives. For the 20-round setting, overall accuracy increases by 0.03% and specificity by 1.29%. In the 10-client configuration, FedADMM's accuracy is 0.28% higher than that of the other two methods, with specificity improving by 2.16% and F1-score by 0.48%. In the 15-client setting, accuracy increases by approximately 4.07%, specificity by about 13.01%, the F1-score improves by 11.05%, and AUC increases by 0.18%.

Overall, it is evident that as the number of clients increases, FedADMM overcomes the challenge more effectively. It also demonstrates excellent performance on the Malgenome dataset. Unlike FedAvg—which only updates parameters based on the client's dataset before sharing them with the server—FedADMM requires clients to update their network parameters using a dual variable, a process that demands more processing power and memory. Consequently, FedADMM achieves superior results across these datasets and exhibits its best performance in various scenarios and rounds.

TABLE I: Global Models Results Across Datasets and Algorithms for 10 rounds

	No. of	FedAvg				FedProx				FedAdmm			
	Clients	**Specificity**	**F1-score**	**AUC**	**FPR**	**Specificity**	**F1-score**	**AUC**	**FPR**	**Specificity**	**F1-score**	**AUC**	**FPR**
Drebin	5	0.9729 ± 0.0027	0.9755 ± 0.0027	0.9962 ± 0.0003	0.0271 ± 0.0027	0.9712 ± 0.0020	0.9747 ± 0.0021	0.9961 ± 0.0003	0.0288 ± 0.0020	**0.9738 ± 0.0049**	0.9739 ± 0.0041	0.9956 ± 0.0005	**0.0262 ± 0.0049**
	10	0.9608 ± 0.0041	0.9677 ± 0.0043	0.9954 ± 0.0006	0.0392 ± 0.0041	0.9604 ± 0.0043	0.9676 ± 0.0044	0.9954 ± 0.0006	0.0396 ± 0.0043	**0.9746 ± 0.0080**	**0.9728 ± 0.0069**	**0.9958 ± 0.0010**	**0.0254 ± 0.0080**
	15	0.9553 ± 0.0167	0.9642 ± 0.0113	0.9943 ± 0.0023	0.0447 ± 0.0167	0.9544 ± 0.0172	0.9639 ± 0.0115	0.9943 ± 0.0024	0.0456 ± 0.0172	**0.9670 ± 0.0156**	**0.9669 ± 0.0130**	**0.9948 ± 0.0022**	**0.0330 ± 0.0156**
Malgenome	5	0.9560 ± 0.0744	0.9726 ± 0.0422	0.9994 ± 0.0009	0.0440 ± 0.0744	0.9544 ± 0.0777	0.9716 ± 0.0444	0.9994 ± 0.0009	0.0456 ± 0.0777	**0.9907 ± 0.0112**	**0.9845 ± 0.0065**	**0.9995 ± 0.0005**	**0.0093 ± 0.0112**
	10	0.8444 ± 0.2962	0.8662 ± 0.2939	0.9973 ± 0.0053	0.1556 ± 0.2962	0.8402 ± 0.2979	0.8633 ± 0.2943	0.9973 ± 0.0055	0.1598 ± 0.2979	**0.8795 ± 0.2960**	**0.8816 ± 0.2945**	**0.9976 ± 0.0053**	**0.1205 ± 0.2960**
	15	0.6788 ± 0.3938	0.7142 ± 0.3910	0.9937 ± 0.0100	0.3212 ± 0.3938	0.6753 ± 0.3938	0.7114 ± 0.3914	0.9936 ± 0.0102	0.3247 ± 0.3938	**0.7923 ± 0.3962**	**0.7872 ± 0.3936**	**0.9965 ± 0.0080**	**0.2077 ± 0.3962**
Tuandromd	5	0.9839 ± 0.0064	0.9784 ± 0.0268	0.9976 ± 0.0019	0.0161 ± 0.0064	0.9841 ± 0.0063	0.9784 ± 0.0268	0.9976 ± 0.0019	0.0159 ± 0.0063	**0.9882 ± 0.0074**	**0.9883 ± 0.0074**	**0.9979 ± 0.0017**	**0.0118 ± 0.0074**
	10	0.9882 ± 0.0076	0.9590 ± 0.0368	0.9945 ± 0.0043	0.0118 ± 0.0076	0.9882 ± 0.0076	0.9574 ± 0.0380	0.9944 ± 0.0043	0.0118 ± 0.0076	0.9843 ± 0.0097	**0.9789 ± 0.0272**	**0.9965 ± 0.0034**	0.0157 ± 0.0097
	15	0.9993 ± 0.0021	0.9154 ± 0.0284	0.9841 ± 0.0325	0.0007 ± 0.0021	0.9995 ± 0.0016	0.9134 ± 0.0268	0.9839 ± 0.0328	0.0005 ± 0.0016	0.9852 ± 0.0156	**0.9669 ± 0.0353**	**0.9920 ± 0.0127**	0.0148 ± 0.0156

TABLE II: Global Models Results Across Datasets and Algorithms for 20 rounds

	No. of	FedAvg				FedProx				FedAdmm			
	Clients	**Specificity**	**F1-score**	**AUC**	**FPR**	**Specificity**	**F1-score**	**AUC**	**FPR**	**Specificity**	**F1-score**	**AUC**	**FPR**
Drebin	5	0.9734 ± 0.0022	0.9767 ± 0.0023	0.9963 ± 0.0003	0.0266 ± 0.0022	0.9715 ± 0.0018	0.9756 ± 0.0019	0.9962 ± 0.0003	0.0285 ± 0.0018	0.9728 ± 0.0052	0.9729 ± 0.0046	0.9955 ± 0.0006	**0.0272 ± 0.0052**
	10	0.9602 ± 0.0052	0.9695 ± 0.0036	0.9957 ± 0.0006	0.0398 ± 0.0052	0.9594 ± 0.0053	0.9691 ± 0.0038	0.9956 ± 0.0006	0.0406 ± 0.0053	**0.9730 ± 0.0064**	**0.9734 ± 0.0059**	0.9956 ± 0.0007	**0.0270 ± 0.0064**
	15	0.9562 ± 0.0113	0.9655 ± 0.0092	0.9950 ± 0.0017	0.0438 ± 0.0113	0.9559 ± 0.0119	0.9653 ± 0.0092	0.9949 ± 0.0018	0.0441 ± 0.0119	**0.9710 ± 0.0137**	**0.9706 ± 0.0096**	0.9955 ± 0.0018	**0.0290 ± 0.0137**
Malgenome	5	0.9751 ± 0.0295	0.9840 ± 0.0153	0.9996 ± 0.0005	0.0249 ± 0.0295	0.9737 ± 0.0300	0.9834 ± 0.0156	0.9996 ± 0.0005	0.0263 ± 0.0300	**0.9880 ± 0.0229**	**0.9847 ± 0.0120**	0.9996 ± 0.0005	**0.0120 ± 0.0229**
	10	0.9162 ± 0.2151	0.9300 ± 0.2149	0.9983 ± 0.0038	0.0838 ± 0.2151	0.9158 ± 0.2151	0.9298 ± 0.2148	0.9983 ± 0.0039	0.0842 ± 0.2151	**0.9378 ± 0.2165**	**0.9348 ± 0.2148**	**0.9985 ± 0.0047**	**0.0622 ± 0.2165**
	15	0.8052 ± 0.3419	0.8238 ± 0.3471	0.9972 ± 0.0040	0.1948 ± 0.3419	0.8037 ± 0.3419	0.8227 ± 0.3469	0.9971 ± 0.0040	0.1963 ± 0.3419	**0.9353 ± 0.2174**	**0.9343 ± 0.2151**	**0.9990 ± 0.0024**	**0.0647 ± 0.2174**
Tuandromd	5	0.9900 ± 0.0045	0.9863 ± 0.0188	0.9981 ± 0.0018	0.0100 ± 0.0045	0.9900 ± 0.0045	0.9859 ± 0.0199	0.9980 ± 0.0018	0.0100 ± 0.0045	**0.9915 ± 0.0057**	**0.9899 ± 0.0062**	**0.9984 ± 0.0014**	**0.0085 ± 0.0057**
	10	0.9865 ± 0.0074	0.9666 ± 0.0344	0.9952 ± 0.0076	0.0135 ± 0.0074	0.9866 ± 0.0076	0.9663 ± 0.0343	0.9951 ± 0.0079	0.0134 ± 0.0076	0.9858 ± 0.0092	**0.9839 ± 0.0201**	**0.9972 ± 0.0025**	0.0142 ± 0.0092
	15	0.9909 ± 0.0090	0.9489 ± 0.0406	0.9925 ± 0.0115	0.0091 ± 0.0090	0.9911 ± 0.0093	0.9473 ± 0.0407	0.9924 ± 0.0116	0.0089 ± 0.0093	0.9899 ± 0.0119	**0.9787 ± 0.0282**	0.9924 ± 0.0214	0.0101 ± 0.0119

V. Conclusion

In this work, we introduced a primal-dual optimization framework for FL-based malware detection. The proposed method incorporated dual variables to regulate local updates, mitigating client drift and enabling automatic adaptation without requiring extensive hyperparameter tuning. Empirical evaluations demonstrated superior convergence properties, achieving higher accuracy with fewer communication rounds compared to FedAvg and FedProx. The method consistently outperformed baselines by reducing the false positive rate and improving AUC, ensuring enhanced detection reliability. Furthermore, scalability analysis showed that as the number of participating clients increased, the algorithm maintained superior performance gains, reinforcing its effectiveness in large-scale FL deployments. These findings established a principled optimization strategy for malware detection in federated settings which offers a computationally efficient and communication-aware alternative to existing approaches.

References

[1] A. Guerra-Manzanares, "Machine learning for android malware detection: mission accomplished? a comprehensive review of open challenges and future perspectives," *Computers & Security*, vol. 138, p. 103654, 2024.

[2] J. Liu, J. Zeng, F. Pierazzi, L. Cavallaro, and Z. Liang, "Unraveling the key of machine learning solutions for android malware detection," *arXiv preprint arXiv:2402.02953*, 2024.

[3] M. Nobakht, R. Javidan, and A. Pourebrahimi, "Sim-fed: Secure iot malware detection model with federated learning," *Computers and Electrical Engineering*, vol. 116, p. 109139, 2024.

[4] H. Kaur, V. Rani, M. Kumar, M. Sachdeva, A. Mittal, and K. Kumar, "Federated learning: a comprehensive review of recent advances and applications," *Multimedia Tools and Applications*, vol. 83, no. 18, pp. 54 165–54 188, 2024.

[5] D. Torre, A. Chennamaneni, J. Jo, G. Vyas, and B. Sabrsula, "Toward enhancing privacy preservation of a federated learning cnn intrusion detection system in iot: Method and empirical study," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 2, pp. 1–48, 2025.

[6] M. Venkatasubramanian, A. H. Lashkari, and S. Hakak, "Iot malware analysis using federated learning: A comprehensive survey," *IEEe Access*, vol. 11, pp. 5004–5018, 2023.

[7] M. Abdel-Basset, H. Hawash, K. M. Sallam, I. Elgendi, K. Munasinghe, and A. Jamalipour, "Efficient and lightweight convolutional networks for iot malware detection: A federated learning approach," *IEEE Internet of Things Journal*, vol. 10, no. 8, pp. 7164–7173, 2022.

[8] S. Wang, R. Morabito, S. Hosseinalipour, M. Chiang, and C. G. Brinton, "Device sampling and resource optimization for federated learning in cooperative edge networks," *IEEE/ACM Transactions on Networking*, 2024.

[9] D. Thakur, A. Guzzo, and G. Fortino, "Hardware-algorithm co-design of energy efficient federated learning in quantized neural network," *Internet of Things*, vol. 26, p. 101223, 2024.

[10] N. Kumari and P. K. Jana, "Communication efficient federated learning with data offloading in fog-based iot environment," *Future Generation Computer Systems*, vol. 158, pp. 158–166, 2024.

[11] H. Wang and J. Xu, "Friends to help: Saving federated learning from client dropout," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 8896–8900.

[12] V. Rey, P. M. S. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in iot devices," *Computer Networks*, vol. 204, p. 108693, 2022.

[13] W. Fang, J. He, W. Li, X. Lan, Y. Chen, T. Li, J. Huang, and L. Zhang, "Comprehensive android malware detection based on federated learning architecture," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3977–3990, 2023.

[14] M. E. Khoda, T. Imam, J. Kamruzzaman, I. Gondal, and A. Rahman, "Robust malware defense in industrial iot applications using machine learning with selective adversarial samples," *IEEE Transactions on Industry Applications*, vol. 56, no. 4, pp. 4415–4424, 2019.

[15] R. Taheri, M. Shojafar, F. Arabikhan, and A. Gegov, "Unveiling vulnerabilities in deep learning-based malware detection: Differential privacy driven adversarial attacks," *Computers & Security*, vol. 146, p. 104035, 2024.

[16] I. Gulrajani, C. Raffel, and L. Metz, "Towards gan benchmarks which require generalization," *arXiv preprint arXiv:2001.03653*, 2020.

[17] W. Huang, M. Ye, Z. Shi, G. Wan, H. Li, B. Du, and Q. Yang, "Federated learning for generalization, robustness, fairness: A survey and benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[18] L. Lyu, H. Yu, X. Ma, C. Chen, L. Sun, J. Zhao, Q. Yang, and S. Y. Philip, "Privacy and robustness in federated learning: Attacks and defenses," *IEEE transactions on neural networks and learning systems*, 2022.

[19] R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, "Fed-iiot: A robust federated malware detection architecture in industrial iot," *IEEE transactions on industrial informatics*, vol. 17, no. 12, pp. 8442–8452, 2020.

[20] A. Khraisat, A. Alazab, S. Singh, T. Jan, and A. Jr. Gomez, "Survey on federated learning for intrusion detection system: Concept, architectures, aggregation strategies, challenges, and future directions," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1–38, 2024.

[21] L. Li, W. Xu, T. Chen, G. B. Giannakis, and Q. Ling, "Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 1544–1551.

[22] A. Acharya, A. Hashemi, P. Jain, S. Sanghavi, I. S. Dhillon, and U. Topcu, "Robust training in high dimensions via block coordinate geometric median descent," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 11 145–11 168.

[23] C. Jiang, K. Yin, C. Xia, and W. Huang, "Fedhgcdroid: An adaptive multi-dimensional federated learning for privacy-preserving android malware classification," *Entropy*, vol. 24, no. 7, p. 919, 2022.

[24] D. Hamouda, M. A. Ferrag, N. Benhamida, Z. E. Kouahla, and H. Seridi, "Android malware detection based on network analysis and federated learning," in *Cyber Malware: Offensive and Defensive Systems*. Springer, 2023, pp. 23–39.

[25] B. Li, Y. Wu, J. Song, R. Lu, T. Li, and L. Zhao, "Deepfed: Federated deep learning for intrusion detection in industrial cyber–physical systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5615–5624, 2021.

[26] D. Javeed, M. S. Saeed, M. Adil, P. Kumar, and A. Jolfaei, "A federated learning-based zero trust intrusion detection system for internet of things," *Ad Hoc Networks*, vol. 162, p. 103540, 2024.

[27] H. Yang, L. Cheng, and M. C. Chuah, "Deep-learning-based network intrusion detection for scada systems," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 1–7.

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

[29] S. Y. Yerima and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," *IEEE transactions on cybernetics*, vol. 49, no. 2, pp. 453–466, 2018.

[30] Y. Zhang, C. Jiang, B. Yue, J. Wan, and M. Guizani, "Information fusion for edge intelligence: A survey," *Information Fusion*, vol. 81, pp. 171–186, 2022.

[31] P. Borah, D. Bhattacharyya, and J. Kalita, "Malware dataset generation and evaluation," in *2020 IEEE 4th Conference on Information & Communication Technology (CICT)*. IEEE, 2020, pp. 1–6.