AsyncFilter: Detecting Poisoning Attacks in Asynchronous Federated Learning

Yufei Kang
University of Toronto
Toronto, Ontario, Canada
yufei.kang@mail.utoronto.ca

Baochun Li
University of Toronto
Toronto, Ontario, Canada
bli@ece.toronto.edu

Abstract

Federated learning has garnered substantial research attention as a privacy-preserving learning paradigm. Nonetheless, its inherently distributed architecture, especially in asynchronous settings, poses vulnerabilities to model poisoning attacks. In such scenarios, malicious clients compromise the integrity of the global model by transmitting manipulated updates. To tackle this issue, previous research efforts like AFLGuard and Zeno++ have made strides but often hinge on the impractical assumption that the server has access to a clean dataset. In this study, we delve into practical solutions tailored for real-world scenarios, minimizing assumptions about the server's capabilities and accommodating varying settings. Specifically, we propose AsyncFilter module, which defends against poisoning attacks in asynchronous federated learning. Functioning as a plug-and-play module on the server, AsyncFilter enhances the learning process by statistically identifying and filtering out poisoned updates during training. On four real-world datasets, AsyncFilter effectively enhances global model accuracy against model poisoning attacks by up to 7%, 20%, 16% and 39% respectively. Through extensive evaluation, AsyncFilter demonstrates robust capabilities in detecting and mitigating model poisoning attacks in various scenarios encompassing diverse data and system heterogeneity, as well as varying attacker presence.

CCS Concepts: • **Computing methodologies** → **Distributed artificial intelligence**.

Keywords: Federated Learning, Cloud Computing

ACM Reference Format:
Yufei Kang and Baochun Li. 2024. AsyncFilter: Detecting Poisoning Attacks in Asynchronous Federated Learning. In *25th International Middleware Conference (MIDDLEWARE '24), December 2–6, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3652892.3700787

MIDDLEWARE '24, December 2–6, 2024, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-0623-3/24/12…$15.00
https://doi.org/10.1145/3652892.3700787

1 Introduction

As one of the emerging distributed machine learning paradigms, federated learning (FL) [14] has attracted widespread research attention, thanks to its merits in preserving data privacy. In federated learning, a central server coordinates training among edge devices, referred to as clients, to build a shared global model without directly accessing datasets. At each communication round, clients train the model locally and commit the model updates to the server. The server then aggregates the received updates and updates the global model. This repeats until the model converges. Due to its potential promise of protecting privacy of client data, FL has been deployed in a variety of real-world applications such as Google's next-word prediction Gboard, drug discovery project Melloddy [16] and Apple's Siri for automatic speech recognition [17].

Yet, most existing mechanisms on federated learning failed to scale efficiently beyond a few hundred clients with the presence of extremely slow clients, called stragglers. This is because they assumed fully synchronized aggregation of client updates and the server always waits for stragglers before aggregation. To alleviate the negative impact of stragglers on model convergence, asynchronous federated learning (AFL) was proposed [24]. In asynchronous design, the server is enabled to perform aggregation as long as a portion of the clients have reported, which helps reduce the time it takes to complete each round of aggregation and accelerates model convergence.

Owing to its inherently distributed framework, asynchronous federated learning is susceptible to model poisoning attacks. In these attacks, malicious attackers compromise the integrity of the global model and reduce its accuracy by transmitting manipulated model updates to the central server throughout the federated learning cycle.

Two predominant strategies exist for executing such poisoning attacks in federated learning: targeted and untargeted poisoning. Targeted attacks are aimed at achieving a specific malicious objective. Untargeted attacks, on the other hand, aim to generally degrade the performance of the federated

learning model without any specific target in mind. Our emphasis is chiefly placed on untargeted poisoning, given its threat to real-world implementations.

However, detecting poisoning attackers in asynchronous federated learning presents a myriad of challenges. From one viewpoint, data heterogeneity—stemming from the decentralized nature of federated learning—introduces substantial variability in model updates. This variability makes it hard to differentiate malicious updates, crafted by poisoning attackers, from genuine updates originating from data that are not independently and identically distributed (non-IID). The challenge is further accentuated by another inherent trait of asynchronous mechanisms: staleness. Stale updates are a common occurrence in asynchronous federated learning, as clients are permitted to submit local updates based on outdated versions of the global model. Moreover, the degree of staleness varies across clients due to differences in their local systems, further amplifying the heterogeneity of the updates. Specifically, model updates are not just reflections of diverse data, but also of different initial model states.

The combined challenges of data heterogeneity and staleness significantly complicate the task of detecting poisoning attempts, making it a particularly arduous endeavor. Proposing an efficient detection methodology that can navigate these complexities is essential to ensuring the security and integrity of asynchronous federated learning.

Existing methodologies addressing poisoning attacks in federated learning concentrate on synchronous Byzantine-robust systems. Byzantine-robust aggregation method, such as Krum [4], Trimmed-Mean and Median [26] and FLtrust [5], targets at learning an accurate model despite the malicious or Byzantine behaviors of some clients. Besides, some studies, for instance FLDetector [27], have focused on the preemptive detection of poisoned updates before their aggregation. Although various defense methods have shown some success in synchronous federated learning, there is a notable scarcity of solutions explored in the realm of asynchronous federated learning.

To date, only two studies, namely AFLGuard [9] and Zeno++ [25], have delved into asynchronous federated learning. These studies operate under the assumption that the server possesses a clean dataset, which it uses to evaluate local updates against clean updates from this dataset. However, this assumption is often unrealistic, as enabling the server to collect sensitive client data prior to learning can infringe on client privacy.

In this study, we explore the solution to model poisoning problems in asynchronous federated learning under a real-world environment. Specifically, we bridge the research gap by proposing an AsyncFilter module that actively detects and filters out malicious updates before server aggregation. Instead of completely relying on pre-collected clean and IID distributed datasets as in prior studies, AsyncFilter analyzes the received local model updates and identifies abnormal clients statistically. Specifically speaking, AsyncFilter makes predictions for clients with different staleness levels and calculates suspicious score based on the prediction. Furthermore, the method uses a 3-means clustering method to identify and reject potentially poisoned updates meanwhile tolerating updates trained from non-IID clients. With AsyncFilter, the robustness of the asynchronous federated learning process against such attacks has been improved, especially in non-IID data scenarios.

To rigorously evaluate the effectiveness and robustness of AsyncFilter, we conducted extensive empirical evaluations using four real-world datasets: MNIST, FashionMNIST, CIFAR-10 and CINIC-10. Demonstrated by the experimental results, AsyncFilter effectively enhances global model accuracy against model poisoning attacks by up to 7%, 20%, 16% and 39% on each dataset. Further, to validate the robustness of AsyncFilter, we subjected it to various testing scenarios encompassing diverse data and system heterogeneity, different server staleness limits, as well as varying numbers of attackers, and achieved consistent accuracy improvement. The results from all these experiments affirm the effectiveness of AsyncFilter in different challenging contexts.

Our contributions are highlighted as follows. First, we pioneer the study of defending against poisoning attacks in asynchronous federated learning under the realistic condition where the server has no access to any dataset. Second, to address this challenge, we propose AsyncFilter, a novel method designed to detect and filter out malicious model updates, thereby enhancing system security. A distinct advantage of this method is its modularity, acting as a "plug and play" component that can function seamlessly alongside secure aggregation techniques. Additionally, through extensive experiments, we demonstrate the efficacy and robustness of AsyncFilter in thwarting model poisoning under a variety of settings.

This paper is organized as follows: The next section provides the background of our work, including examples of related works and their limitations. Section 3 discusses the problem formulation and outlines the assumptions for the problem setting. Section 4 illustrates our approach with an example and introduces the system design for AsyncFilter. Section 5 details the experimental settings and presents comprehensive results. Finally, Section 6 concludes this paper.

2 Background

Federated learning is a distributed machine learning approach that allows multiple clients to train a shared machine learning model, without the need for all clients to share their raw data to a central server. Commonly, federated learning works synchronously.

In synchronous federated learning, the clients iteratively train a global model with the coordination of a cloud server.

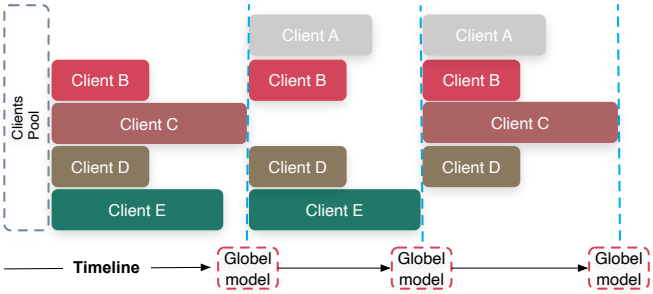

Figure 1. Synchronous federated learning workflow.

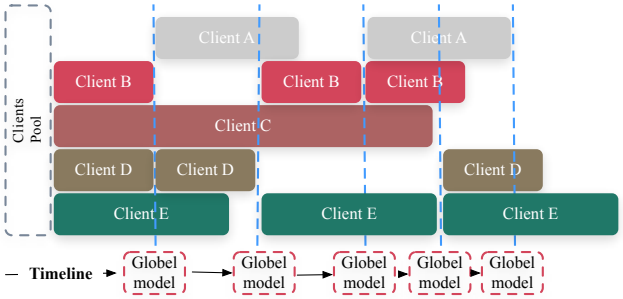

Figure 2. Asynchronous federated learning workflow.

In each iteration, each client optimizes a local model on its own data, and then periodically sends the model updates to the central server. Synchronous server waits for reports from multiple clients before using them to update the global model. This iterative process continues until the global model converges, as shown in Fig. 1. However, if some slow clients are sampled in a round, both the server and the clients having arrived must wait for these slow reports. This delay in aggregation slows down the entire federated learning process. As a solution, asynchronous federated learning has been proposed.

2.1 Asynchronous Federated Learning

Compared to traditional synchronous federated learning's weakness in handling stragglers, this asynchronous approach allows the server to perform aggregation as long as partial clients' reports arrive and therefore accelerates convergence in a scalable setting.

Optimization objective. Consider an asynchronous federated learning system involving a set of $\mathcal{N} = 1, \ldots, N$ clients, coordinated by a central server. Each client i has n_i local training data samples $(x_{i,1}, \ldots, x_{i,n_i})$, and the total number of training data samples across N devices is $n_{total} = \sum_{i=1}^{N} n_i$.

Define $f(\cdot)$ as the loss function where $f(\omega; x_{i,j})$ indicates how the machine learning model parameter ω performs on the input $x_{i,j}$, which is the j-th data sample of client i. Thus, the local loss function $F_i(\omega)$ of client i can be defined as

$$F_i(\omega) := \frac{1}{n_i} \sum_{j=1}^{n_i} f(\omega; x_{i,j}). \tag{1}$$

Same in synchronous FedAvg, denote p_i as the aggregation weight of the i-th client such that $\sum_{i=1}^{N} p_i = 1$. Then, by denoting $F(\omega)$ as the global loss function, the goal of asynchronous federated learning is to find the solution ω of the following optimization problem

$$\min_{\omega} F(\omega) := \sum_{i=1}^{N} p_i F_i(\omega). \tag{2}$$

Here, denoting t as the index of an asynchronous federated learning round, we describe one round (*e.g.*, the t-th) proceeding as follows: The server uniformly samples a subset of K clients at random (*i.e.*, $K = \|\mathcal{K}\|$ and $\mathcal{K} \subseteq \mathcal{N}$) and broadcasts the latest model ω^t to the sampled clients. Each sampled client i sets the local model as $\omega_i^{t,0} = \omega^t$ and runs E steps of local optimization on (1) to compute local update model $\omega_i^{t,E}$. After finishing local training, the sampled client sends the updated model back to the server.

When the server receives Ω reports from sampled K clients ($\Omega \leq K$), it performs aggregation with weight p_i and compute a new global model by

$$\omega^{t+1} = \sum_{i}^{\Omega} p_i \omega_i. \tag{3}$$

Notably, ω_i does not have to be local update based on the latest round $\omega_i^{t,E}$. It can be $\omega_i^{t-\tau_i,E}$ that was optimized on a global model in a much earlier round, as staleness $\tau_i < t$ is allowed. The above process repeats for many rounds until the global loss converges.

With such an asynchronous mode of operation as the essential idea, FedBuff [15] is proposed. In FedBuff, the server randomly samples clients at each round as FedAvg but introduces a buffer to store local updates and only aggregates when the buffer size reaches a certain aggregation goal. Due to its simplicity, FedBuff and its variants with different weighting methods are commonly used.

2.2 Poisoning Attacks to Federated Learning

Federated learning's distributed nature makes it vulnerable to poisoning attacks. Attackers can corrupt the global model either by introducing fake clients or compromising genuine ones, leading to a decline in the model's testing accuracy. Poisoning attacks in this context can be categorized into targeted and untargeted types. Targeted attacks are designed to manipulate the global model to incorrectly predict specific labels for chosen inputs, while leaving its performance on other inputs unaffected. Conversely, untargeted attacks disrupt the global model more broadly, causing it to

perform poorly across a wide range of indiscriminate testing examples, thus rendering it largely dysfunctional. Our paper primarily addresses untargeted attacks, given their widespread and significant impact. Below, we review the untargeted attacks.

Untargeted attacks. Untargeted poisoning attacks are typically calibrated with a delicate balance: they are potent enough to cause significant accuracy loss, yet subtle enough to elude detection by the server. In line with this strategy, the Little-Is-Enough (LIE) attack is proposed in [3] that subtly introduces minor noise increments to each dimension of the average benign gradients, thereby poisoning the global model. In [18], Min-Sum and Min-Max attacks are designed. In Min-Sum attack, the sum of squared distances of malicious gradients from all benign gradients is upper bounded by the sum of squared distances of any benign gradient from the other benign gradients. In Min-Max attacks, malicious gradients are generated such that its maximum distance from any other gradient is upper bounded by the maximum distance between any two benign gradients. The Gradient Deviation (GD) attack, as detailed in [8], manipulates the model updates from malicious clients in such a way that the global model update is directed oppositely to the gradient's direction. In this study, we include these four poisoning attacks to evaluate our defense method.

2.3 Secure Federated Learning

Secure synchronous federated learning. Secure federated learning that aims to counteract poisoning attacks predominantly focuses on synchronous FL environments. These Byzantine-robust methods in synchronous FL are known for their use of robust aggregation rules. The core idea behind these rules is to accurately learn the model, even when some clients exhibit malicious or Byzantine behaviors. For example, the Krum aggregation rule [4] is designed to select the model update that demonstrates the smallest cumulative distance to its $n-m-2$ neighbors, where n and m represent the total and malicious clients, respectively. Additionally, methods such as Trimmed-Mean and Median [26] aim to calculate the mean or median value for each dimension from the gathered model updates. To reduce heterogeneity, a Bucketing scheme is proposed as a pre-aggregation module [11]. Similarly, Byz-VR-MARINA is designed in combination with variance reduction and compression [10]. Meanwhile, FLtrust [5] takes a different approach, enhancing trust through the utilization of a clean dataset stored on the server, which is then used to assess the trustworthiness of each client. Additionally, instead of taking advantage of server dataset, Nearest neighbor mixing (NNM) [2] averages each input with a subset of their nearest neighbors.

Other defenses for synchronous Federated Learning involve detecting malicious clients. To protect FL from poisoning attacks, various detection methods have been proposed. These methods typically view poisoned model updates as statistical outliers among benign model updates. In [13], a spectral anomaly detection model is trained on a clean dataset before initiating federated learning. The state-of-the-art FLDetector [27] detects malicious clients by measuring the consistency between the client's model update and the server's predicted model update based on historical model updates, all without needing a clean dataset.

However, these synchronous methods are not applicable for an asynchronous context because they rely on the homogeneity of model updates, which is not the case with asynchronous updates.

Secure asynchronous federated learning. To protect asynchronous federated learning, Zeno++ [25] and AFLGuard [9] are proposed, assuming that the server holds a clean dataset prior to learning. In Zeno++, the server uses this trusted dataset to filter clients' model updates. The server computes a model update based on the trusted dataset, then calculates the cosine similarity between each client's model update and the server's. If the similarity is positive, the server normalizes the client's model update. AFLGuard employs a similar technique, using a clean model update to compare each local model update. However, a client's model update is considered benign only if it does not significantly deviate from the server's model update in both direction and magnitude.

These studies operate under the assumption that the server possesses a clean dataset, which it uses to evaluate local updates against clean updates from this dataset. However, this assumption is hard to be satisfied with in reality, as enabling the server to collect sensitive client data prior to learning can infringe on client privacy. Addressing a gap in existing research, our innovative AsyncFilter demonstrates robust defense against poisoning attacks, ensuring enhanced security and reliability in diverse environments.

3 Problem Formulation

3.1 Threat Model

In this study, we adopt the attack model outlined in prior research. This model involves an attacker exerting control over several malicious clients. These clients may be either attacker-fabricated or compromised legitimate ones, yet they still possess data from the same distribution as benign clients. It is important to note, however, that in this scenario, the server remains uncompromised by the attacker. During each iteration of the federated learning training process, these malicious clients are capable of transmitting arbitrary local model updates to the server. Generally, an attacker in such a system possesses limited knowledge, which typically includes: the local training data and model updates from the malicious clients, the loss function used, and the learning rate.

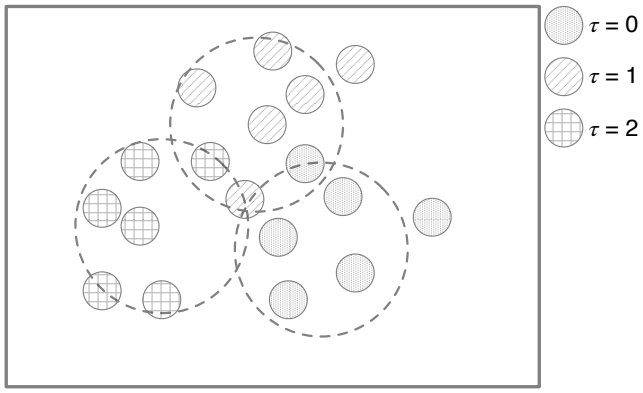

Figure 3. t-SNE diagram of local updates on MNIST (IID).

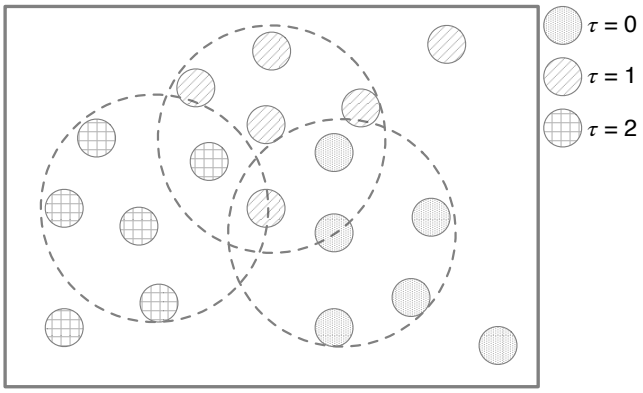

Figure 4. t-SNE diagram of local updates on MNIST (Non-IID).

3.2 Defense Goal

Our objective is to develop an asynchronous federated learning approach that meets two key criteria: First, in non-adversarial environments, our method aims to match or exceed the accuracy of FedBuff. This means that when operating with all benign clients, our approach should facilitate the learning of a global model with a level of accuracy comparable to that achieved by FedBuff. Additionally, the method is designed to be resilient against a range of poisoning attacks, including both existing and adaptive strategies, in adversarial settings.

3.3 Server's Capability and Knowledge

Contrasting with prior research, such as those outlined in [25] and [9], which assume the availability of a clean dataset, our approach operates under a different premise. In our model, the server is unaware of the local data distribution and consequently lacks access to any datasets. This approach is both reasonable and practical, particularly in real-world applications. Allowing a server to gather data samples mirroring the clients' distribution often compromises client privacy and can discourage participation. Our method acknowledges and addresses these privacy concerns by eliminating the need for server access to client-distributed data.

4 AsyncFilter: System Design

4.1 Challenges

Identifying poisoning attackers in asynchronous federated learning is fraught with complexities. **On one hand**, the inherent data heterogeneity of federated learning, due to its decentralized structure, leads to significant variation in model updates. This variation blurs the line between malicious updates, intentionally altered by attackers, and legitimate updates from clients with non-IID data. **On the other hand**, the characteristic staleness of asynchronous systems further complicates matters. Clients in such a system are allowed to send updates based on outdated versions of the global model, increasing the diversity of updates.

Therefore, local updates reflect not only the heterogeneous data but also different states of the model at the start of the local optimization process. The duality of these two factors — data heterogeneity and staleness — turns the process of detecting poisoning attacks into a highly challenging endeavor.

4.2 Observation and Motivation

Experimental settings. To investigate the effects of staleness and data heterogeneity on local updates, our study delves into local updates reported from participating clients. Specifically, we employed the LeNet-5 [1] model, training it with PyTorch on the MNIST dataset, which comprises $60,000$ 32×32 color images distributed across 10 classes. Our asynchronous federated learning framework includes 500 clients, each operating on an individual thread in parallel. At each communication round, when 150 model updates arrive, the central server performs model aggregation and updates the global model. To represent system speed heterogeneity, the processing latency of clients is modeled to follow a Zipf distribution with a parameter s of 1.2. This setup results in a distribution where most devices exhibit high speed, a minority are significantly slower (stragglers), and a moderate number have medium speed. Regarding data heterogeneity, we arrange the local data partitions based on the Dirichlet distribution with a concentration parameter of 0.01. This ensures that data samples are highly non-IID distributed across local devices.

Impact of data heterogeneity and staleness. To analyze the statistical properties of received updates and the influence of data distribution and staleness on them, we utilize t-SNE visualizations [21] for the local updates at each communication round. We label each update sample with its staleness level and use consistent coloring for samples sharing the same level of staleness τ. For clarity, we summarize our observations on both IID and non-IID scenarios from the t-SNE representations in Fig. 3 and Fig. 4 respectively, where

each ball represents a local update sample and the pattern represents its staleness level.

It reveals two key observations. **Firstly**, there is a noticeable dispersion among balls of the same pattern, suggesting that local updates from clients are statistically scattered. This pattern reflects the extent to which data heterogeneity influences the spread of the distribution. **Secondly**, despite the scattering, balls sharing the same pattern tend to cluster around a common center. This observation highlights the significant impact of the staleness level on the central tendency of the distribution. In short, updates sharing a staleness level tend to cluster around a common center, while the extent of non-IID characteristics in local updates influences how far these updates deviate from this central point.

4.3 System Design: AsyncFilter

Building upon these observations, we are ready to introduce AsyncFilter, an innovative algorithm designed to safeguard against model poisoning attacks in asynchronous federated learning. Upfront, as we have been abundantly clear, AsyncFilter is crafted to operate effectively in real-world scenarios, where data heterogeneity and staleness levels present challenges for the central server. Design details are as follows.

Staleness-based clustering. In asynchronous federated learning, the system inherently allows for staleness, meaning that received model updates might be based on outdated global models. From our empirical observations, even within an IID setting, the disparities between model updates derived from varying global model versions eclipse the differences between poisoned and genuine model updates. Consequently, a necessary step is to categorize model updates according to their respective staleness levels.

With AsyncFilter, the server actively monitors the staleness $\tau(\omega_i)$ of each client i and groups the received local updates ω_i accordingly:

$$\mathbb{C} = \{\mathbb{C}_k : \mathbb{C}_k = \{\omega_i | \tau(\omega_i) = \tau_k\}, k = 1, 2, \ldots, m\}, \quad (4)$$

where $\mathbb{C}$ is the set of all groups and $\mathbb{C}_k$ is a subset of $\mathbb{C}$ containing model updates ω_i that have the same staleness τ_k, and m is the maximum staleness enabled by the server..

This ensures that, within each group, the variances introduced by different staleness levels are neutralized.

Moving average estimation. In our methodology, we incorporate a moving average approach to process the incoming model updates in each group. This technique is instrumental in mitigating the variability inherent in model updates received from diverse clients. In asynchronous federated learning, where updates are asynchronously contributed by a multitude of clients with varying data distributions, the aggregation of model updates can introduce significant variance, leading to instability in the learning process. By applying a moving average to these updates, we effectively smooth out abrupt fluctuations and ensure a more stable estimation for each staleness group.

At the t-th round of asynchronous federated learning, the server receives stale updates such as ω_i^{t-1}, ω_{i+1}^{t-2}... which are trained on earlier global models ω_g^{t-2}, ω_g^{t-3} ... Notably, in the server's previous aggregation round (*i.e.*, the $t-1$-th round), we had already gathered local model updates corresponding to the same group as these stale updates. Specifically, the fresh model update ω_k^{t-1} from the $t-1$-th round originates from the same training starting point as the stale update ω_i^{t-1} in the t-th round. To harness the full breadth of these statistical data, we implement a moving average for each staleness group and estimate model updates as follows.

$$\mathcal{MA}(\mathbb{C}_k) = \frac{t}{t+1}\mathcal{MA}(\mathbb{C}_k) + \frac{1}{t+1}\omega_i, \quad (5)$$

where $\mathcal{MA}(\mathbb{C}_k)$ is the moving average estimation for staleness group $\mathbb{C}_k$.

This approach not only enhances the robustness of the model against erratic or noisy updates but also aids in maintaining a consistent learning trajectory across iterations. The moving average operation, in this context, acts as a stabilizing factor, averaging out the disparities among updates and thus fostering a more reliable and gradual integration of knowledge from distributed local data into the global model.

Distance-based suspicious scores. So far we have estimated model updates for each group as an expectation. Intuitively, updates closer to the standard model tend to originate from benign clients. Hence, we propose a distance-based suspicious score for each update to assess its likelihood of being malicious. For each client i in its corresponding staleness group $\mathbb{C}_k$, we first compute the l_2-norm distance between its local update and group estimation as follows,

$$d(\mathcal{MA}_k, \omega_i) = \sqrt{(\mathcal{MA}_k - \omega_i)^2}. \quad (6)$$

The underlying intuition is that, even in the presence of a non-IID data distribution, impactful poisoned model updates tend to deviate more substantially from the group's estimation. If a subtle attacker makes only minimal modifications to the model updates, and such alterations render the model closer to the average than those trained from non-IID datasets, then this is not regarded as a successful attack, given its negligible impact on the global model.

Furthermore, we calculate the suspicious score for client i by normalization as follows.

$$score_i = \frac{d(\mathcal{MA}_k, \omega_i)}{\sqrt{\sum_{k=1}^{m} d(\mathcal{MA}_k, \omega_i)^2}}, \quad (7)$$

where $0 \leq score_i \leq 1$ and a higher $score_i$ denotes a higher probability of poisoned updates.

Attacker identification. In our approach, we employ a 3-means clustering method to process the suspicious scores, specifically chosen over a 2-means method to allow a greater degree of tolerance for model updates originating from non-IID local data. Within the three identified groups, the one with the highest average score is labeled as the attacker

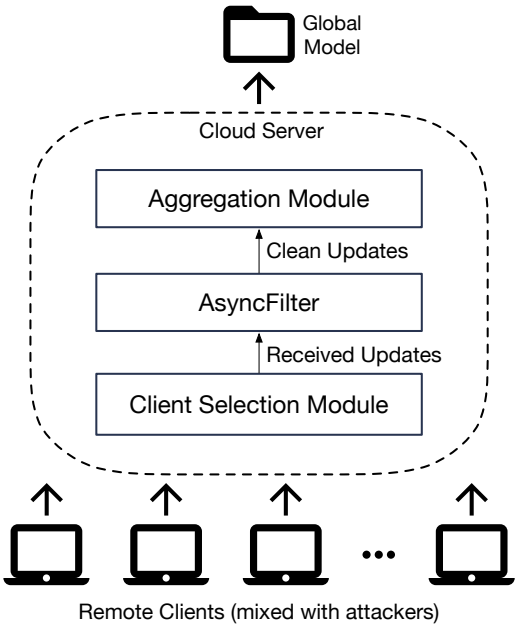

Figure 5. Asynchronous federated learning with AsyncFilter.

group, and updates from this group are subsequently excluded from server aggregation. Conversely, the group with the lowest average score is deemed to be honest and its updates are accepted for aggregation. The middle group, potentially comprising less effective attackers and honest clients with non-IID data, is permitted to contribute to the aggregation at a later stage. This is because our experimental findings indicate that incorporating updates from a few weak attackers typically does not significantly harm the global model. However, excluding updates from honest clients can result in a noticeable loss of accuracy.

This tripartite clustering approach enables a more nuanced filtering mechanism. It not only helps in identifying and excluding true outliers that could damage the model but also ensures that updates reflecting legitimate, albeit diverse, data characteristics are not unjustly discarded. Consequently, this method strikes a balance between maintaining the integrity of the global model and embracing the valuable diversity of client data, ultimately leading to a more robust and representative federated learning model.

4.4 Workflow of AsyncFilter

So far, we have introduced the technical details of AsyncFilter step by step and now we propose the AsyncFilter module for asynchronous federated learning. We summarize the AsyncFilter in Alg. 1. AsyncFilter works in three key steps: first, it categorizes model updates based on their staleness levels; second, for each staleness group, it employs a moving average and normalization for each staleness group to calculate a suspicious score for each client; finally, the method uses a

Algorithm 1 AsyncFilter

1: **Input:** Received model updates ω_i
2: **Output:** Genuine model updates ω_{clean}
3: **for** model updates $\omega_i, i = 1, \ldots, N$ with staleness τ_k **do**
4: Update group estimation $\mathbb{C}_k$ with formula (4).
5: **end for**
6: **for** group $\mathbb{C}_k$ $(k = 1, \ldots, m)$ **do**
7: Calculate suspicious score with formula (5) and (6).
8: Identify poisoned updates via 3-means method.
9: **end for**

3-means clustering approach to identify malicious updates while tolerating non-IID clients.

Notably, the AsyncFilter can be seamlessly integrated into all asynchronous federated learning systems as a pluggable component without the need for additional sensitive information, as shown in Fig. 5. For instance, integrated with AsyncFilter, asynchronous federated learning operates as follows: in each communication round of asynchronous federated learning, the server selects clients following the customized rule and selected clients perform local optimization; once the local training is complete, the client sends the updated model back to the server; when the number of arrived clients reaches the minimum aggregation bound, the server starts to detect and filter out suspicious ones with the AsyncFilter module; after removing abnormal updates, the server aggregates the updates following its aggregation rule and moves into the next round.

4.5 Theoretical Analysis on Suspicious Scores

Theoretically, we compare the suspicious scores of benign and malicious clients. We first make the assumptions following [15] and [22] and then achieve Theorem1.

Assumption 1. *(Intra-cluster similarity): For all clients $i \in \mathcal{S}$ with local dataset x_i, and some constant $A > 0$,*

$$\|\nabla f_i(x_i) - \nabla \bar{f}_i(x_i)\|^2 \leq A^2 \|\nabla \bar{f}_i(x_i)\|^2 \tag{8}$$

where $\bar{f}_i(x_i) = \frac{1}{N} \sum_{i \in \mathcal{S}} f_i(x_i)$

Assumption 2. *(Bounded local and global variance): for all clients $i \in \mathcal{S}$,*

$$\sigma_{l,min}^2 \leq \mathbb{E}_{\zeta_i | i}[\|g_i(\omega; \zeta_i) - \nabla F_i(\omega)\|^2] \leq \sigma_{l,max}^2 \tag{9}$$

$$\frac{1}{N} \sum_{i=1}^{N} \|\nabla F_i(\omega) - \nabla f(\omega)\|^2 \leq \sigma_{g,max}^2 \tag{10}$$

With these assumptions, we have

Theorem 1. *Suppose FedAvg is used as the aggregation rule, the clients' local training datasets are non-IID distributed. Suppose the malicious clients perform an untargeted model poisoning attack, e.g. the GD attack, in each iteration by reversing the*

true model updates as the poisoning ones, i.e., each malicious client j sends $-\delta_j^t$ to the server in iteration t. Then we have the expected suspicious score of a benign client is smaller than that of a malicious attacker. Formally, with $A \leq \sqrt{2 + \frac{\sigma_{l,min}^2}{\sigma_{g,max}}}$, we have the following inequality:

$$\mathbb{E}(score_i^t) \leq \mathbb{E}(score_j^t), \forall i \in \mathcal{B}, \forall j \in \mathcal{M}, \tag{11}$$

where the expectation $\mathbb{E}$ is taken with respect to the randomness in the local optimization, $\mathcal{B}$ is the set of honest clients, and $\mathcal{M}$ is the set of malicious attackers.

Proof sketch:

$$\begin{aligned}
&\mathbb{E}[score_j^t] - \mathbb{E}[score_i^t] \\
=&\mathbb{E}_j\|\omega_t - \omega_j\|^2 - \mathbb{E}_i\|\omega_t - \omega_i\|^2 \\
=&\mathbb{E}_j\|\omega t - (\omega_{t-1} - \delta_j)\|^2 - \mathbb{E}_i\|\omega_t - (\omega_{t-1} + \delta_i)\|^2 \\
=&\mathbb{E}_j\|\omega_t - \omega_{t-1} + \delta_j)\|^2 - \mathbb{E}_i\|\omega t - \omega_{t-1} - \delta_i\|^2 \\
=&\mathbb{E}_j\|\delta_t + \delta_j)\|^2 - \mathbb{E}_i\|\delta_t - \delta_i\|^2 \\
=&\mathbb{E}_j\|\bar{\nabla}F(\omega_t) + \nabla F(\omega_j)\|^2 - \mathbb{E}_i\|\bar{\nabla}F(\omega_t) - \nabla F(\omega_i)\|^2 \\
\geq&\mathbb{E}_j\|\bar{\nabla}F(\omega_t) + \nabla F(\omega_j)\|^2 - A^2\bar{\nabla}F(\omega_t) \\
\geq&\mathbb{E}_j\|\bar{\nabla}F(\omega_t)^2\| + \mathbb{E}_i\|\bar{\nabla}F(\omega_j)^2\| - A^2\bar{\nabla}F(\omega_t) \\
=&2\bar{\nabla}F(\omega_t)^2 + \sigma_{l,min} - A^2\bar{\nabla}F(\omega_t) \\
\geq&0
\end{aligned} \tag{12}$$

5 Experimental Results

In this section, we evaluate the effectiveness and robustness of AsyncFilter through extensive empirical experiments. Our goal is to confirm that AsyncFilter performs robustly against various attacks, different numbers of attackers, different server staleness limits, as well as varying levels of data and system heterogeneity.

5.1 Experimental Setup

Datasets and models. In order to provide a comprehensive and fair evaluation, we ran experiments on four real-world datasets:

MNIST [7]: MNIST is a 10-class digit image classification dataset, which consists of 60,000 training examples and 10,000 testing examples. For MNIST dataset, we train a LeNet-5 model.

FashionMNIST [23]: FashionMNIST is a 10-class fashion image classification dataset, which has a predefined training set of 60,000 fashion images and a testing set of 10,000 fashion images. For FashionMNIST dataset, we train a LeNet-5 model.

CIFAR-10 [12] : CIFAR-10 is a 10-class color image classification dataset consisting of predefined 50,000 training examples and 10,000 testing examples. For CIFAR-10 dataset, we train a VGG-16 [19] model.

Table 1. Configuration parameters.

Dataset	MNIST	FashionMNIST	CINIC-10	CIFAR-10
Model	Lenet-5	VGG-16	VGG-16	VGG-16
Partition size	10^3	$2 * 10^3$	$1 * 10^4$	$1 * 10^4$
Local epochs	5	5	5	5
Batch size	32	32	128	128
Optimizer	SGD	SGD	Adam	Adam
Learning rate	0.01	0.01	0.01	0.01
Momentum	0.9	0.9	0	0

CINIC-10 [6]: CINIC-10 is an augmented extension of CIFAR-10. It contains images from CIFAR-10 and a selection of ImageNet database images. Its training and testing dataset contain 90,000 image samples each. For CINIC-10 dataset, we train a VGG-16 model as well.

Parameters. We summarize all configuration parameters at Table 1. For training tasks on MNIST and FashionMNIST, we used SGD as the local optimizer. For training tasks on CINIC-10 and CIFAR-10, we applied Adam optimizer. Notably, to guarantee model convergence, we assigned larger partition sizes to clients for large image datasets such as CIFAR-10 and CINIC-10.

Platform. We conducted all of our experiments on Plato [20], an open-source framework designed for scalable federated learning research. The Plato framework uses object-oriented subclassing, leveraging Python 3's ABC library and Data Classes. Additionally, the framework supports defining callback classes and customizing trainers by providing a list of custom callback classes. With its ability to scale to a large number of clients and its extensibility to accommodate a wide variety of datasets, models, and FL algorithms, Plato is an ideal tool for federated learning research. The framework abstracts away the underlying ML drivers using convenient APIs, making it agnostic to deep learning frameworks such as *TensorFlow*, *PyTorch*, and *MindSpore*.

Plato facilitates client-server communication via industry-standard WebSockets. The server can either run on the same GPU-enabled physical machine as its clients for emulation research testbeds or be deployed in a cloud datacenter. Plato supports heterogeneity in both client local time and local data distribution during sampling. The framework allows for specifying random seeds for random number generators and protecting random number generation from third-party frameworks using random.getstate() and random.setstate(). With these mechanisms in place, Plato enables fully reproducible experiments through its reproducible mode, where the same set of clients and data samples are selected across runs.

AFL setting. We ran all experiments using plato framework on a server, with 3 NVIDIA RTX A4500 GPUs with 20GB of CUDA memory. By default, in each run, 100 clients

participate and all of them are selected by the server sampler at each round. The minimum aggregation bound for asynchronous federated learning is 40 so that when 40% of the selected clients arrive, the server performs aggregation and updates the global model. By default, the server tolerated local updates with a staleness of less than 20.

For system heterogeneity, we configure clients' processing latency to follow the Zipf distribution with parameter s of 1.2. Zipf distribution models a practical scenario with various device speeds such that the majority of devices are fast and a few devices are stragglers, while a medium number of devices are with middle-of-the-road speed. With $s > 1$, it satisfies convergence of the generalized harmonic series.

To additionally introduce data heterogeneity, we set up the local data distribution for each client. As four datasets such as MNIST, FashionMNIST, CIFAR-10 and CINIC-10 are collected as a centralized dataset, we sample local data partition following the Dirichlet distribution with a concentration parameter of 0.1. In Dirichlet distribution, with $\alpha \leq 1$, the samples will be highly concentrated in a few labels, and all the rest labels will have almost no samples, while with $\alpha > 1$, the samples will be dispersed almost equally among all the labels.

Attack setting. By default, we randomly sample 20 out of 100 of the clients as malicious ones and mix them with benign clients. At each communication round, the server selects clients from the whole clients pool. This means that the number of attackers in each round is changing and unknown to the server. For a fair evaluation of untargeted model poisoning attacks, we have selected four prominent attack methods for consideration: LIE, Min-Max, Min-Sum, and GD attacks. For details on these attacks, please refer to Section 2.2.

Baselines. So far, in the realm of asynchronous federated learning, no specific countermeasures against attackers have been proposed. We incorporated the baseline scheme FedBuff, which does not actively counteract attackers. Additionally, FedBuff serves as a crucial comparison point to ensure that our approach safeguards the global model when operating solely with benign clients. We also included FLDetector [27], a state-of-the-art detection method in synchronous federated learning. FLDetector identifies malicious clients by contrasting client model updates with server-predicted updates, which are generated from historical data. By including FLDetector in our empirical evaluation, we aim to underscore the necessity of developing a detection method specifically tailored to the asynchronous federated learning environment.

Performance metrics. In our experiments, we used the test accuracy of the final global model to evaluate the defense methods. Typically, attacks compromise the global model by reducing its test accuracy, as they misguide the model into making incorrect predictions. An AFL defense method is deemed more robust against such attacks if its global models manage to maintain higher test accuracy even when under attack.

Table 2. AsyncFilter defends against attacks on MNIST.

	The Achieved Accuracy				
Attack	GD	LIE	Min-Max	Min-Sum	No attack
FedBuff	86.6%	**96.9**%	89.0%	**97.4**%	97.0%
FLDetector	82.9%	93.6%	84.9%	95.7%	95.1%
AsyncFilter	**93.0**%	95.6%	**93.9**%	97.3%	**97.2**%

Table 3. AsyncFilter defends against attacks on FashionMNIST.

	The Achieved Accuracy				
Attack	GD	LIE	Min-Max	Min-Sum	No attack
FedBuff	72.2%	**86.2**%	77.4%	65.9%	**86.5**%
FLDetector	69.1%	82.2%	71.1%	83.83%	82.5%
AsyncFilter	**79.1**%	83.1%	**81.0**%	**86.12**%	85.3%

Table 4. AsyncFilter defends against attacks on CIFAR-10.

	The Achieved Accuracy				
Attack	GD	LIE	Min-Max	Min-Sum	No attack
FedBuff	70.3%	52.0%	**84.7**%	85.2%	83.9%
FLDetector	75.3%	48.5%	79.4%	85.6%	81.2%
AsyncFilter	**76.2**%	**60.2**%	83.8%	**85.6**%	**84.8**%

Table 5. AsyncFilter defends against attacks on CINIC-10.

	The Achieved Accuracy(%)				
Attack	GD	LIE	Min-Max	Min-Sum	No attack
FedBuff	10%	26.3%	17.3%	51.3%	**56.0**%
FLDetector	46.3%	10.3%	42.0%	50.5%	53.4%
AsyncFilter	**49.2**%	**53.2**%	**56.8**%	**52.3**%	53.4%

5.2 AsyncFilter Defends Attackers Effectively

We first evaluate the performance of AsyncFilter against model poisoning attacks on four real-world datasets and summarize the results in Table 2 - 5. The test accuracy of the final global model with different methods under different attacks are as shown. The "No attack" scenario depicts an ideal setting where all participants are honest, and no malicious attackers are present.

Initially, it is important to note that **AsyncFilter demonstrates both efficacy and safety in non-adversarial environments**. In scenarios devoid of malicious clients, AsyncFilter maintains a global model accuracy comparable to that of FedBuff. On four real-world datasets such as MNIST, FashionMNIST, CIFAR-10 and CINIC-10, the final global model accuracy without attacks are respectively 97.23%, 85.31%, 84.84% and 53.37% for AsyncFilter and 97.04%, 86.51%, 83.91% and 56.02% for FedBuff, while the accuracy are degraded to 95.16%, 82.5%, 81.23% and 53.47% with FLDetector.

Additionally, in Table 2 - 5, our analysis reveals that **AsyncFilter method effectively defends against model poisoning attacks on all four datasets, particularly against strong poisoning attacks**. Table 2 shows that on MNIST dataset, GD and Min-Max attacks are two strong ones that cause great model accuracy reduction of 10%. AsyncFilter improves the model accuracy to 93.04% and 93.97% respectively. For less effective attacks LIE and Min-Sum that incur accuracy reduction around 1%, both AsyncFilter and FedBuff preserve model accuracy. However, with FLDetector, due to its unconsciousness of staleness, it incurs more accuracy loss instead of compensation under all attacks. Table 3 shows that on FashionMNIST dataset, GD, Min-Sum and Min-Max attacks are strong and cause accuracy loss from 10% to 20%. AsyncFilter defends against them and improves accuracy to 79.11%, 81.00% and 86.12%.

Moving to more advanced datasets such as the CIFAR-10 dataset and CINIC-10 dataset, results are as shown in Table 4 and Table 5 respectively. For CIFAR-10, GD and LIE attacks reduce accuracy from 83.91% to 70.36% and 52.08%. AsyncFilter defends against them and improves accuracy by 6% and 8%, outperforming FLDetector. For less effective attacks, AsyncFilter has similar performance with FedBuff. On CINIC-10 dataset, all attacks incur great accuracy loss, and FedBuff even diverges under GD attacks. AsyncFilter exhibits superior defense capabilities against these attacks, including GD, Min-Max, and Min-Sum attacks. This performance is markedly better than that of the existing comparison methods.

In conclusion, the experimental results strongly support the effectiveness of our AsyncFilter method in defending against poisoning attacks in asynchronous federated learning contexts. This method not only successfully mitigates the impact of highly effective attacks but also maintains robust performance in the face of less effective ones, underscoring its adaptability and efficiency as a defense mechanism in diverse attack scenarios.

5.3 AsyncFilter Is Robust Against Data Heterogeneity

In real-world scenarios, local data often exhibits a non-IID distribution across remote clients, with the central server lacking direct access to this local data distribution. Therefore, robustness against unknown local data heterogeneity is crucial for any federated learning method. To assess AsyncFilter's robustness against data heterogeneity, we formulated a setting with varying data distribution for CINIC-10 and FashionMNIST datasets. We adjusted the Dirichlet distribution parameters from 0.1 to 0.05, and 0.01 respectively and therefore incurring more non-IID local data than previous environment. The results are detailed in Tables 6 and 7.

Table 6. AsyncFilter is robust against data heterogeneity on CINIC-10.

	The Achieved Accuracy(%)			
Attack	GD	LIE	Min-Max	Min-Sum
FedBuff	30.7%	10.4%	44.2%	43.1%
FLDetector	**46.3**%	14.3%	40.3%	46.3%
AsyncFilter	41.0%	**49.3**%	**47.2**%	**48.8**%

Table 7. AsyncFilter is robust against data heterogeneity on FashionMNIST.

	The Achieved Accuracy(%)			
Attack	GD	LIE	Min-Max	Min-Sum
FedBuff	10%	**63.4**%	31.8%	**73.7**%
FLDetector	24.2%	47.9%	37.8%	65.8%
AsyncFilter	**30.7**%	60.4%	**41.6**%	69.0%

The experimental findings affirm **AsyncFilter's effectiveness and resilience in managing data heterogeneity within asynchronous federated learning environments**. In experiments conducted on the CINIC-10 dataset with a Dirichlet parameter of 0.05, all attacks resulted in substantial accuracy degradation. Notably, the model fails to converge under the LIE attack. While FLDetector effectively counters the GD attack, AsyncFilter demonstrates superior performance in mitigating the LIE, Min-Max, and Min-Sum attacks. When tested on the FashionMNIST dataset with a Dirichlet parameter set to 0.01, the extreme non-IID distribution inherently led to significant accuracy reduction. The introduction of model poisoning further exacerbates the issue. As illustrated in Table 7, the GD attack inflictes considerable damage, to the extent that the model is unable to achieve convergence. Both AsyncFilter and FLDetector are effective in their defense against attacks. Specifically, AsyncFilter leads to a significant improvement in accuracy by 20%, surpassing the 14% enhancement achieved by FLDetector. In the case of the Min-Max attack, which notably diminishes model accuracy, AsyncFilter successfully boosts accuracy by 10%. For other attacks like LIE and Min-Sum, AsyncFilter maintains accuracy levels akin to the best existing strategy, which involves no intervention.

Table 8. AsyncFilter is robust against doubled attackers on CINIC-10.

	The Achieved Accuracy(%)			
Attack	GD	LIE	Min-Max	Min-Sum
FedBuff	10%	10.0%	10.0%	**51.7**%
FLDetector	29.2%	10.3%	**50.3**%	50.0%
AsyncFilter	**38.1**%	**34.5**%	46.9%	46.9%

Table 9. AsyncFilter is robust against doubled attackers on FashionMNIST.

	The Achieved Accuracy(%)			
Attack	GD	LIE	Min-Max	Min-Sum
FedBuff	10%	**85.3**%	72, 7%	73.1%
FLDetector	19.9%	81.3%	69.1%	82.7%
AsyncFilter	**31.3**%	83.1%	**78.9**%	**85.0**%

5.4 AsyncFilter Is Robust Against Different Number of Attackers

In practical scenarios, attackers can control multiple clients and integrate these controlled devices into a federated learning round. Therefore, it's essential for federated learning methods to be resilient against an unknown number of attackers. To test AsyncFilter's robustness in such conditions, we altered the number of total attackers from 20 to 40, effectively doubling their presence. The outcomes of this modification on CINIC-10 and FashionMNIST are detailed in Table 8 and 9.

On the CINIC-10 dataset, as detailed in Table 8, the presence of GD, LIE, and Min-Max attackers, particularly when their numbers are doubled, significantly disrupts the federated learning process, leading to model divergence. Analyzing the first two columns of the table, it's evident that FLDetector is unable to effectively counter GD and LIE attacks, with the latter causing the model to fail to converge. In contrast, AsyncFilter markedly enhances model accuracy, elevating it from 10% to 38% and 34.53% for GD and LIE attacks, respectively. In the case of the Min-Max and Min-Sum attacks, AsyncFilter attains an accuracy level comparable to the best ones there. On the FashionMNIST dataset, as demonstrated in Table 9, the GD attack emerges as the most significant threat, beating FLDetector. However, AsyncFilter effectively counters this attack, achieving a model accuracy of 31.3%. Furthermore, AsyncFilter exhibits superior performance compared to its counterparts in defending against Min-Max and Min-Sum attacks. In the case of the less potent LIE attack on this dataset, AsyncFilter's performance is on par with other methods. In short, **AsyncFilter consistently exhibits superior defensive capabilities against varying numbers of attackers**.

Table 10. AsyncFilter is robust against speed heterogeneity on FashionMNIST.

	The Achieved Accuracy(%)			
Methods	GD	LIE	Min-Max	Min-Sum
FedBuff	83.7%	85.5%	80.9%	84.5%
FLDetector	80.1%	83.9%	69.0%	81.7%
AsyncFilter	**83.8**%	**85.5**%	**83.1**%	**85.1**%

5.5 AsyncFilter Is Robust Against System Speed Heterogeneity

In practical federated learning scenarios, remote client devices often display a lack of uniformity, with notable variations in their training speeds and communication rates. Consequently, it becomes imperative for federated learning methods to exhibit robustness against such diverse system speed heterogeneity. To evaluate AsyncFilter's effectiveness in addressing this heterogeneity, we implemented a varied experimental setup for the FashionMNIST dataset. This involved adjusting the Zipf distribution parameter s from 1.2 to 2.5. With 2.5, the speed distribution is more skewed, indicating a few devices vary fast, and the rest are much slower. This change emphasizes the impact of speed divergence across remote clients in federated learning. The results of this assessment are presented in Tables 10.

Demonstrated by the results, AsyncFilter defends against all four considered attacks and exhibits a superior performance compared to its counterparts. Specifically, when countering Min-Max attacks, AsyncFilter enhances the final accuracy by 3%, whereas FLDetector experiences an accuracy decline. These results underscore **AsyncFilter's proficiency in effectively neutralizing poisoning attacks within environments characterized by system heterogeneity** in asynchronous federated learning.

5.6 AsyncFilter Is Robust Against Varying Staleness Limits

In realistic scenarios, the central server imposes different staleness limits on updates; specifically, when the staleness of an update exceeds this limit, the server rejects the update instead of aggregating it. To evaluate the robustness of AsyncFilter under varying staleness limits, we conducted a parameter study.

We carried out experiments using the LeNet-5 model on the FashionMNIST dataset under critical attacks, such as the GD attack and the LIE attack. We varied the maximum staleness limits across the values $\{5, 10, 15, 20\}$. To reduce the impact of randomness, each experiment was repeated

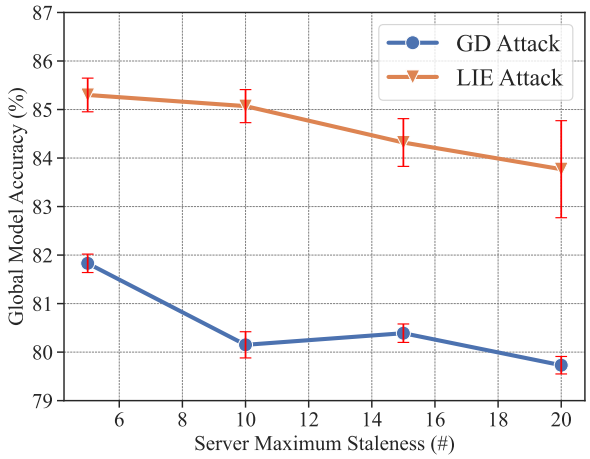

Figure 6. As the staleness limit increases, global model accuracy decreases, reflecting the negative impact of stale updates on convergence. Despite this, AsyncFilter maintains a stable accuracy under both attacks.

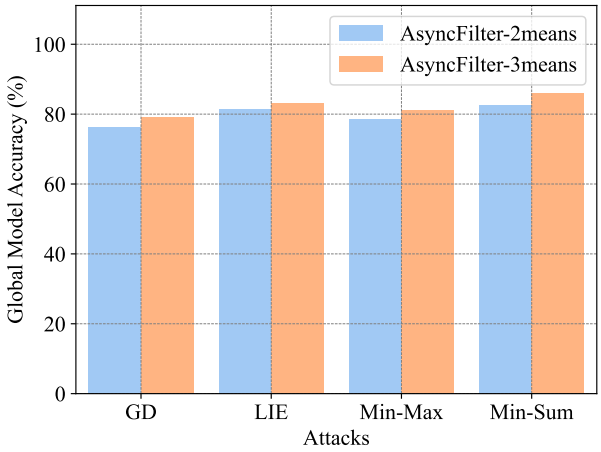

Figure 7. AsyncFilter-3means demonstrates advantages over AsyncFilter-2means. It tolerates updates trained from non-IID Data, unlike AsyncFilter-2means which often excessively rejects such updates. This results in improved accuracy.

three times with different random seeds to ensure statistical significance. The results are shown in Figure 6, with solid markers representing the mean and error bars indicating the standard deviation.

As shown in the line plot, the global model accuracy decreases as the staleness limit increases, which is expected since stale updates hinder model convergence. Notably, AsyncFilter demonstrates stable performance under both attacks, maintaining a final global model accuracy over 84% and 80%, respectively, even as the staleness limit ranges from 5 to 20. **This result highlights the effectiveness and robustness of AsyncFilter against varying server staleness limits.**

5.7 AsyncFilter Tolerates Updates Trained From Non-IID Data

In AsyncFilter, we employ the k-means clustering method to group clients. A key insight here is the use of a 3-means approach rather than the traditional 2-means. This strategy allows for a more nuanced classification, avoiding a strict dichotomy between suspicious and genuine clients. Specifically, AsyncFilter outputs three groups, identifying the one with the highest score as comprising malicious attackers.

To demonstrate the superiority of the 3-means over the conventional 2-means in this context, we conducted an experiment using the LeNet-5 model on the FashionMNIST dataset. The experiment compares AsyncFilter with 3-means as a grouping method (AsyncFilter-3means) and AsyncFilter with 2-means (AsyncFilter-2means). To simulate data heterogeneity, we set the Dirichlet distribution parameter as 0.1, reflecting a real-world-like, yet not excessively rare, non-IID distribution.

The outcomes of defending against Gradient Descent (GD), LIE, Min-Max, and Min-Sum attacks are presented in Figure 7. The results clearly indicate that AsyncFilter-3means outperforms AsyncFilter-2means in terms of global model accuracy across all the mentioned attack types. This is because AsyncFilter-2means tends to excessively reject legitimate updates trained from non-IID data, consequently leading to a decline in accuracy. **This finding underscores the effectiveness of the 3-means approach within AsyncFilter in the presence of data heterogeneity**.

6 Concluding Remarks

In this study, we explore the crucial issue of model poisoning attacks in asynchronous federated learning, particularly in practical, real-world scenarios where the server's capabilities are minimized due to privacy concerns. We introduce AsyncFilter, a robust solution to combat these threats. Designed as a "plug-and-play" module for a federated server, AsyncFilter protects the learning process by statistically identifying and eliminating poisoned updates throughout the training. Through comprehensive evaluations across four diverse real-world datasets, AsyncFilter empirically demonstrates its efficacy against various sophisticated model poisoning scenarios encompassing diverse data and system heterogeneity as well as varying magnitudes of attacker presence.

AsyncFilter effectively mitigates a major vulnerability in asynchronous federated learning, thereby contributing to the development of safer and more reliable systems in this domain. This study, we hope, will inspire ongoing innovation and research in this crucial field of distributed machine learning.

References

[1] 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[2] Youssef Allouah, Sadegh Farhadkhani, Rachid Guerraoui, Nirupam Gupta, Rafaël Pinot, and John Stephan. 2023. Fixing by mixing: A recipe for optimal byzantine ml under heterogeneity. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1232–1300.

[3] Gilad Baruch, Moran Baruch, and Yoav Goldberg. 2019. A little is enough: Circumventing defenses for distributed learning. In *Proc. Advances in Neural Information Processing Systems (NuerIPS)*, Vol. 32.

[4] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. *Proc. Advances in neural information processing systems (NeurIPS)* 30 (2017).

[5] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2020. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *Proc. Network and Distributed System Security (NDSS) Symposium* (2020).

[6] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. 2018. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505* (2018).

[7] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[8] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *Proc. 29th USENIX security symposium (USENIX Security 20)*. 1605–1622.

[9] Minghong Fang, Jia Liu, Neil Zhenqiang Gong, and Elizabeth S Bentley. 2022. AFLGuard: Byzantine-robust Asynchronous Federated Learning. In *Proc. the 38th Annual Computer Security Applications Conference*. 632–646.

[10] Eduard Gorbunov, Samuel Horváth, Peter Richtárik, and Gauthier Gidel. 2022. Variance reduction is an antidote to byzantines: Better rates, weaker assumptions and communication compression as a cherry on the top. *arXiv preprint arXiv:2206.00529* (2022).

[11] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. 2020. Byzantine-robust learning on heterogeneous datasets via bucketing. *arXiv preprint arXiv:2006.09365* (2020).

[12] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[13] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. 2020. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211* (2020).

[14] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication Efficient Learning of Deep Networks from Decentralized Data. In *Proc. Int'l Conference on Artificial Intelligence and Statistics (AISTATS)*.

[15] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated Learning with Buffered Asynchronous Aggregation. In *Proc. Int'l Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 3581–3607.

[16] Martijn Oldenhof, Gergely Ács, Balázs Pejó, Ansgar Schuffenhauer, Nicholas Holway, Noé Sturm, Arne Dieckmann, Oliver Fortmeier, Eric Boniface, Clément Mayer, et al. 2023. Industry-scale orchestrated federated learning for drug discovery. In *Proc. the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 37. 15576–15584.

[17] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, et al. 2021. Federated Evaluation and Tuning for On-device Personalization: System design and applications. *arXiv preprint arXiv:2102.08503* (2021).

[18] Virat Shejwalkar and Amir Houmansadr. 2021. Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning. In *Proc. Network and Distributed Systems Security (NDSS) Symposium (NDSS)*.

[19] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[20] Ningxin Su and Baochun Li. 2022. How Asynchronous can Federated Learning Be?. In *Proc. 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–11.

[21] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

[22] Mariel Werner, Lie He, Sai Praneeth Karimireddy, Michael Jordan, and Martin Jaggi. 2023. Provably Personalized and Robust Federated Learning. *arXiv preprint arXiv:2306.08393*.

[23] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[24] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous Federated Optimization. In *Proc. NeurIPS Workshop on Optimization for Machine Learning (OPT)*.

[25] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2020. Zeno++: Robust fully asynchronous SGD. In *Proc. International Conference on Machine Learning (ICML)*. PMLR, 10495–10503.

[26] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proc. International Conference on Machine Learning (ICML)*. PMLR, 5650–5659.

[27] Zaixi Zhang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2022. FLDetector: Defending federated learning against model poisoning attacks via detecting malicious clients. In *Proc. the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2545–2555.