# AFLGuard: Byzantine-robust Asynchronous Federated Learning

Minghong Fang
The Ohio State University and Duke University

Jia Liu
The Ohio State University

Neil Zhenqiang Gong
Duke University

Elizabeth S. Bentley
Air Force Research Laboratory



## ABSTRACT

Federated learning (FL) is an emerging machine learning paradigm, in which clients jointly learn a model with the help of a cloud server. A fundamental challenge of FL is that the clients are often heterogeneous, e.g., they have different computing powers, and thus the clients may send model updates to the server with substantially different delays. Asynchronous FL aims to address this challenge by enabling the server to update the model once any client's model update reaches it without waiting for other clients' model updates. However, like synchronous FL, asynchronous FL is also vulnerable to poisoning attacks, in which malicious clients manipulate the model via poisoning their local data and/or model updates sent to the server. Byzantine-robust FL aims to defend against poisoning attacks. In particular, Byzantine-robust FL can learn an accurate model even if some clients are malicious and have Byzantine behaviors. However, most existing studies on Byzantine-robust FL focused on synchronous FL, leaving asynchronous FL largely unexplored. In this work, we bridge this gap by proposing *AFLGuard*, a Byzantine-robust asynchronous FL method. We show that, both theoretically and empirically, AFLGuard is robust against various existing and adaptive poisoning attacks (both untargeted and targeted). Moreover, AFLGuard outperforms existing Byzantine-robust asynchronous FL methods.

## CCS CONCEPTS

• **Security and privacy** → **Systems security**.

## KEYWORDS

Federated Learning, Poisoning Attacks, Byzantine Robustness

**ACM Reference Format:**
Minghong Fang, Jia Liu, Neil Zhenqiang Gong, and Elizabeth S. Bentley. 2022. AFLGuard: Byzantine-robust Asynchronous Federated Learning. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3564625.3567991

## 1 INTRODUCTION

**Background and Motivation:** Federated learning (FL) [27, 32] is an emerging distributed learning framework, which enables clients (e.g., smartphone, IoT device, edge device) to jointly train a *global model* under the coordination of a cloud server. Specifically, the server maintains the global model and each client maintains a *local model*. In each iteration, the server sends the current global model to the clients; a client trains its local model via fine-tuning the global model using its local data, and the client sends the *model update* (i.e., the difference between global model and local model) to the server; and the server aggregates the clients' model updates and uses them to update the global model.

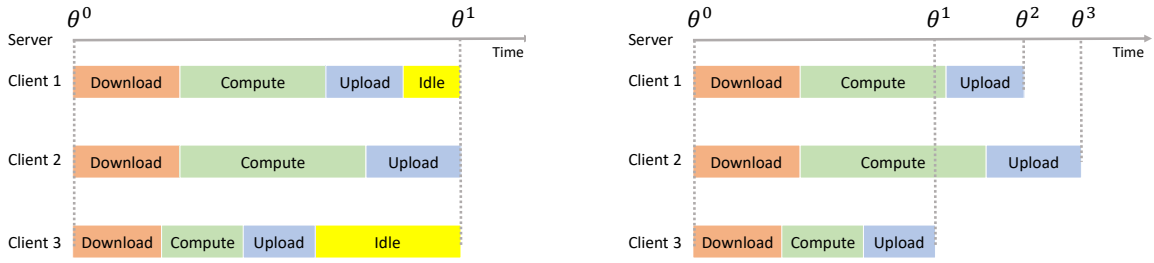
Most existing FL methods are *synchronous* [7, 9, 22, 33, 35, 56]. Specifically, in each iteration of a synchronous FL, the server waits for the model updates from a large number of clients before aggregating them to update the global model. However, synchronous FL faces two key challenges. The first challenge is the so-called *straggler problem*. Specifically, due to clients' unpredictable communication latency and/or heterogeneous computing capabilities, some clients (i.e., stragglers) send their model updates to the server much later than others in each iteration, which substantially delays the update of the global model. Simply ignoring the stragglers' model updates would waste clients' computing resources and hurt accuracy of the global model [44]. The second challenge is that synchronous FL is difficult to implement due to the high complexity in maintaining a perfectly synchronized global common clock.

*Asynchronous FL* aims to address the challenges of synchronous FL. Specifically, in asynchronous FL, the server updates the global model immediately upon receiving a client's model update without waiting for other clients' model updates. Due to the advantages of asynchronous FL, it has been widely incorporated in deep learning frameworks such as TensorFlow [3] and PyTorch [38], as well as deployed by industries, e.g., Meta [26, 36]. However, like synchronous FL, asynchronous FL is also vulnerable to *poisoning attacks* [17, 53, 55], in which malicious clients poison their local data and/or model updates to guide the training process to converge to a bad global model. Specifically, in *untargeted poisoning attacks* [6, 7, 18, 41], the bad global model simply has a large error rate for indiscriminate testing inputs. In *targeted poisoning attacks* [5, 14, 40, 43], the bad global model predicts attacker-chosen label for attacker-chosen testing inputs, but its predictions for other testing inputs are unaffected. For instance, in *backdoor attacks* (one type of targeted poisoning attacks) [5, 37, 43, 49, 51], the attacker-chosen testing inputs are inputs embedded with a backdoor trigger.

Byzantine-robust asynchronous FL aims to defend against poisoning attacks. However, it is highly challenging to design Byzantine-robust asynchronous FL. To date, most existing Byzantine-robust FL methods (e.g., [7, 9, 12, 33, 35, 56]) are designed for synchronous FL. Compared to synchronous FL, the key challenge in designing Byzantine-robust asynchronous FL stems from the fact that noisy model updates are inevitable. Specifically,


*ACSAC '22, December 5–9, 2022, Austin, TX, USA*

ACM ISBN 978-1-4503-9759-9/22/12…$15.00
https://doi.org/10.1145/3564625.3567991




(a) Synchronous FL

(b) Asynchronous FL

**Figure 1: Synchronous vs. asynchronous FL. "Download" means downloading the global model from the server. "Compute" means training a local model. "Upload" means sending the model update to the server.**

when a client sends its model update calculated based on a global model to the server, other clients may have already sent their model updates calculated based on the same global model to the server and thus the global model could have already been updated several times. As a result, delayed model updates are inevitably noisy with respect to the current global model. This asynchrony makes it difficult to distinguish between poisoned model updates from malicious clients and the "noisy" model updates from benign clients.

**Our Work:** In this work, we propose *AFLGuard*, a Byzantine-robust asynchronous FL framework that addresses the aforementioned challenges. In AFLGuard, our key idea to handle the asynchrony complications is to equip the server with a small but clean training dataset, which we call *trusted dataset*. The server (e.g., Meta, Google) can manually collect the trusted dataset for the learning task. When receiving a model update from a client, the server computes a model update (called *server model update*) based on its trusted dataset and the current global model. The server accepts the client's model update only if it does not deviate far from the server model update with respect to both direction and magnitude. In particular, if the magnitude of the difference vector between the client and server model updates is less than a certain fraction of the magnitude of the server model update, then the server uses the client's model update to update the global model. The updated global model is then sent to the client.

Interestingly, we show that this simple intuitive idea of AFLGuard enjoys strong theoretical guarantees. Specifically, under mild assumptions widely adopted by the Byzantine-robust FL community, we prove that the difference between the optimal global model parameters under no malicious clients and the global model parameters learnt by AFLGuard under an arbitrary number of malicious clients can be bounded. We also empirically evaluate AFLGuard and compare it with state-of-the-art Byzantine-robust asynchronous FL methods on a synthetic dataset and five real-world datasets. Our experimental results show that AFLGuard can defend against various existing and adaptive poisoning attacks when a large fraction of clients are malicious. Moreover, AFLGuard substantially outperforms existing Byzantine-robust asynchronous FL methods.

We summarize our main contributions as follows:

- We propose a Byzantine-robust asynchronous FL framework called AFLGuard to defend against poisoning attacks in asynchronous FL.
- We theoretically show that AFLGuard is robust against an arbitrary number of malicious clients under mild assumptions commonly adopted by the Byzantine-robust FL community.
- We conduct extensive experiments to evaluate AFLGuard and compare it with state-of-the-art Byzantine-robust asynchronous FL methods on one synthetic and five real-world datasets.

## 2 PRELIMINARIES AND RELATED WORK

### 2.1 Background on Federated Learning

**Notations:** We use $\|\cdot\|$ to denote $\ell_2$-norm. For any natural number $n$, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$.

**Setup of Federated Learning (FL):** Suppose we have $n$ clients. Client $i$ has a local training dataset $X_i$, where $i = 1, 2, \cdots, n$. For simplicity, we denote by $X = \bigcup_{i=1}^{n} X_i$ the joint training dataset of the $n$ clients. An FL algorithm aims to solve an optimization problem, whose objective function is to find an optimal global model $\boldsymbol{\theta}^*$ that minimizes the expected loss $F(\boldsymbol{\theta})$ as follows:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \Theta} F(\boldsymbol{\theta}) \triangleq \arg\min_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_{x \sim \mathcal{D}} \left[ f(\boldsymbol{\theta}, x) \right], \tag{1}$$

where $\Theta \subseteq \mathbb{R}^d$ is the model-parameter space, $d$ is the dimension of the model-parameter space, $f$ is a loss function that evaluates the discrepancy between an output of a global model and the corresponding ground truth, the expectation $\mathbb{E}$ is taken with respect to the distribution of a training example $x$ (including both feature vector and label), and $\mathcal{D}$ is the training data distribution. In practice, the expectation is often approximated as the average loss of the training examples in the joint training dataset $X$, i.e., $\mathbb{E}_{x \sim \mathcal{D}} \left[ f(\boldsymbol{\theta}, x) \right] \approx \frac{1}{|X|} \sum_{x \in X} f(\boldsymbol{\theta}, x)$.

In FL, the clients iteratively learn a global model with the coordination of a cloud server. In each iteration, synchronous FL waits for the information from multiple clients before using them to update the global model, while asynchronous FL updates the global model once the information from any client reaches it. Fig. 1 illustrates the difference between synchronous FL and asynchronous FL [2].

**Synchronous FL:** Synchronous FL performs three steps in each iteration. In the first step, the server sends the current global model to the clients or a selected subset of them. In the second step, a client trains its local model via fine-tuning the global model using its local training data, and it sends the model update (i.e., the difference between the global model and the local model) to the server.

**Algorithm 1** AsyncSGD.

Server:
Initializes global model $\boldsymbol{\theta}^0 \in \Theta$ and sends it to all clients.
**for** $t = 0, 1, 2, \cdots, T-1$ **do**
  Upon receiving a model update $\boldsymbol{g}_i^{t-\tau_i}$ from client $i$, updates $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \boldsymbol{g}_i^{t-\tau_i}$.
  Sends $\boldsymbol{\theta}^{t+1}$ to client $i$.
**end for**
Client $i$, $i \in [n]$:
**repeat**
  Receives a global model $\boldsymbol{\theta}^t$ from the server.
  Computes stochastic gradient $\boldsymbol{g}_i^t$ based on $\boldsymbol{\theta}^t$ and a random mini-batch of its local training data.
  Sends $\boldsymbol{g}_i^t$ to the server.
**until** Convergence

When all the selected clients have sent their model updates to the server, the server aggregates them and uses the aggregated model update to update the global model in the third step. For instance, in FedAvg [32], the server computes the weighted average of the clients' model updates and uses it to update the global model in the third step.

**Asynchronous FL:** Synchronous FL requires the server to wait for the model updates from multiple clients before updating the global model, which is vulnerable to the straggler problem and delays the training process. In contrast, asynchronous FL updates the global model upon receiving a model update from any client [13, 15, 26, 36, 46, 52, 54]. Specifically, the server initializes the global model and sends it to all clients. Each client trains its local model via fine-tuning the global model based on its local training data, and sends the model update to the server. Upon receiving a model update, the server immediately updates the global model and sends the updated global model back to the client.

Formally, we denote by $\boldsymbol{\theta}^t$ the global model in the $t$th iteration. Moreover, we denote by $\boldsymbol{g}_i^t$ the model update from client $i$ that is calculated based on the global model $\boldsymbol{\theta}^t$. Suppose in the $t$th iteration, the server receives a model update $\boldsymbol{g}_i^{t-\tau_i}$ from client $i$ that is calculated based on an earlier global model $\boldsymbol{\theta}^{t-\tau_i}$ in an earlier iteration $t - \tau_i$, where $\tau_i$ is the delay for the model update. The server updates the global model as follows:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \boldsymbol{g}_i^{t-\tau_i}, \tag{2}$$

where $\eta$ is the global learning rate.

*Asynchronous stochastic gradient descent (AsyncSGD)* [58] is the most popular asynchronous FL method in non-adversarial settings. In AsyncSGD, a client simply fine-tunes the global model using one mini-batch of its local training data to obtain a local model. In other words, a client computes the gradient of the global model with respect to a random mini-batch of its local training data as the model update. Formally, $\boldsymbol{g}_i^t = \frac{1}{|B|} \sum_{x \in B} \nabla f(\boldsymbol{\theta}^t, x)$, where $B$ is a mini-batch randomly sampled from $X_i$. Algorithm 1 shows AsyncSGD, where $T$ is the number of iterations. Note that for simplicity, we assume in all the compared FL methods and our AFLGuard, a client uses such gradient with respect to a random mini-batch of its local training data as the model update.

## 2.2 Byzantine-robust FL

**Poisoning Attacks to FL:** Poisoning attacks have been intensively studied in traditional ML systems, such as recommender systems [19, 21, 31], crowdsourcing systems [20, 34] and anomaly detectors [39]. Due to its distributed nature, FL is also vulnerable to poisoning attacks [5, 6, 10, 18], in which malicious clients poison the global model via carefully manipulating their local training data and/or model updates. The malicious clients can be fake clients injected into the FL system by an attacker or genuine clients compromised by an attacker. Depending on the attack goal, poisoning attacks can be categorized into *untargeted* and *targeted*. In untargeted attacks, a poisoned global model has a large error rate for indiscriminate testing examples, leading to denial-of-service. In targeted attacks, a poisoned global model predicts attacker-chosen labels for attacker-chosen testing inputs, but its predictions for other testing inputs are unaffected.

For instance, *label flipping attack* [56], *Gaussian attack* [7], and *gradient deviation attack* [18] are examples of untargeted attacks. In particular, in the label flipping attack, the malicious clients flip the label $y$ of a local training example to $C - 1 - y$, where $C$ is the total number of labels and the labels are $0, 1, \cdots, C-1$. In the Gaussian attack, the malicious clients draw their model updates from a Gaussian distribution with mean zero and a large standard deviation instead of computing them based on their local training data. In the gradient deviation attack, the model updates from the malicious clients are manipulated such that the global model update follows the reverse of the gradient direction (i.e., the direction where the global model should move without attacks).

Backdoor attack [5, 51] is a popular targeted attack. For instance, in the backdoor attack in [5], each malicious client replicates some of its local training examples; embeds a trigger (e.g., a patch on the right bottom corner of an image) into each replicated training input; and changes their labels to an attacker-chosen one. A malicious client calculates its model update based on its original local training data and the replicated ones. Moreover, the malicious client scales up the model update by a scaling factor before sending it to the server. The poisoned global model would predict the attacker-chosen label for any input embedded with the same trigger, but the predictions for inputs without the trigger are not affected.

**Byzantine-Robust Synchronous FL:** Byzantine-robust FL aims to defend against poisoning attacks. Most existing Byzantine-robust FL methods focus on synchronous FL [7, 9, 56]. Recall that a synchronous FL method has three steps in each iteration. These Byzantine-robust synchronous FL methods adopt robust aggregation rules in the third step. Roughly speaking, the key idea of a robust aggregation rule is to filter out "outlier" model updates before aggregating them to update the global model. For example, the Krum aggregation rule [7] outputs the model update with the minimal sum of distances to its $n-m-2$ neighbors, where $n$ and $m$ are the numbers of total and malicious clients, respectively. Since these methods are designed to aggregate model updates from multiple clients, they are not applicable to asynchronous FL, which updates the global model using one model update. Other defenses for synchronous FL include provably secure defenses to prevent poisoning attacks [11] and methods to detect malicious clients [57].

**Byzantine-Robust Asynchronous FL:** To the best of our knowledge, the works most related to ours are [17, 53, 55]. Specifically, Kardam [17] maintains a Lipschitz coefficient for each client based on its latest model update sent to the server. The server uses a model update from a client to update the global model only if its Lipschitz coefficient is smaller than the median Lipschitz coefficient of all clients. BASGD [55] is a non-conventional asynchronous FL method that uses multiple clients' model updates to update the global model. Specifically, the server holds several buffers and maps each client's model update into one of them. When all buffers are non-empty, the server computes the average of the model updates in each buffer, takes the median or trimmed-mean of the average model updates, and uses it to update the global model. In Zeno++ [53], the server filters clients' model updates based on a trusted dataset. The server computes a server model update based on the trusted dataset. After receiving a model update from any client, the server computes the cosine similarity between the client model update and server model update. If the cosine similarity is positive, then the server normalizes the client model update. Note that FLTrust [9], a synchronous FL method, uses the similar technique as in Zeno++ to filter out malicious information.

**Differences between AFLGuard and Zeno++:** Both our AFLGuard and Zeno++ use a trusted dataset on the server. However, they use it in different ways. Zeno++ simply treats a client's model update as benign if it is positively correlated with the server model update. Due to delays on both client and server sides and the distribution shift between the trusted and clients' training data, the server's and benign clients' model updates may not be positively correlated. In AFLGuard, a client's model update is considered benign if it does not deviate substantially from the server's model update in both direction and magnitude.

## 3 PROBLEM FORMULATION

**Threat Model:** The attacker controls some malicious clients, which could be genuine clients compromised by the attacker or fake clients injected by the attacker. The attacker does not compromise the server. The malicious clients could send arbitrary model updates to the server. The attacker could have different degree of knowledge about the FL system [9, 18], i.e., *partial knowledge* and *full knowledge*. In the partial-knowledge setting, the attacker knows the local training data and model updates on the malicious clients. In the full-knowledge scenario, the attacker has full knowledge of the FL system. In particular, the attacker knows the local training data and model updates on all clients, as well as the FL method and its parameter settings. Note that the attacker in the full-knowledge setting is much stronger than that of partial-knowledge setting [18]. Following [9], we use the full-knowledge attack setting to evaluate the security of our defense in the worst case. In other words, our defense is more secure against weaker attacks.

**Defense Goals:** We aim to design an asynchronous FL method that achieves the following two goals: i) the method should be as accurate as AsyncSGD in *non-adversarial settings*. In other words, when all clients are benign, our method should learn as an accurate global model as AsyncSGD; and ii) the method should be robust against both existing and adaptive poisoning attacks in adversarial settings. Adaptive poisoning attacks refer to attacks that are tailored to the proposed method.

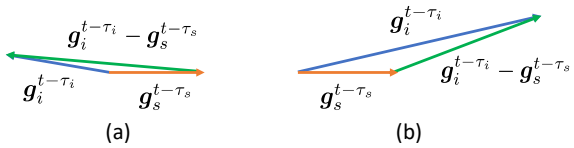



**Figure 2: Illustration of our acceptance criteria. $g_i^{t-\tau_i}$ and $g_s^{t-\tau_s}$ are client model update and server model update, respectively. (a) the direction of $g_i^{t-\tau_i}$ deviates substantially from that of $g_s^{t-\tau_s}$. (b) the magnitude of $g_i^{t-\tau_i}$ deviates substantially from that of $g_s^{t-\tau_s}$. The server rejects the client model update in both cases.**

**Server's Capability and Knowledge:** We assume the server holds a small clean dataset, which we call *trusted dataset*. This assumption is reasonable in practice because it is quite affordable for a service provider to collect and verify a small trusted dataset for the learning task. For instance, Google uses FL for the next word prediction in a virtual keyboard application called Gboard [1]; and Google can collect a trusted dataset from its employees. The trusted dataset does not need to follow the same distribution as the joint training dataset $X$. As our experimental results will show, once the trusted dataset distribution does not deviate substantially from the joint training data distribution, our method is effective. We acknowledge that the trusted dataset should be clean, and our method may not be robust when the trusted dataset is poisoned.

## 4 AFLGUARD

**Intuitions:** The key of our AFLGuard is a criteria to decide whether the server should accept a client's model update to update the global model or not. Ideally, if a model update is from a malicious client performing poisoning attacks, then the server should not use it to update the global model. Our key observation is that, in poisoning attacks, malicious clients often manipulate the directions and/or the magnitudes of their model updates. Therefore, we consider both the direction and magnitude of a client's model update when deciding whether it should be accepted to update the global model or not. Specifically, the server computes a model update (called *server model update*) based on its own trusted dataset. When a client's model update deviates substantially from the server model update with respect to direction and/or magnitude, it is rejected.

**Acceptance Criteria:** Suppose in the $t$th iteration, the server receives a model update $\boldsymbol{g}_i^{t-\tau_i}$ from a client $i \in [n]$, where $\tau_i$ is the delay. Client $i$ calculated the model update $\boldsymbol{g}_i^{t-\tau_i}$ based on the global model $\boldsymbol{\theta}^{t-\tau_i}$, i.e., the server previously sent the global model $\boldsymbol{\theta}^{t-\tau_i}$ to client $i$ in the $(t-\tau_i)$th iteration. Moreover, in the $t$th iteration, the server has a model update $\boldsymbol{g}_s^{t-\tau_s}$ based on its trusted dataset, where $\tau_s$ is the delay (called *server delay*) for the server model update. Specifically, the server trains a local model via fine-tuning the global model $\boldsymbol{\theta}^{t-\tau_s}$ using its trusted dataset, and the model update $\boldsymbol{g}_s^{t-\tau_s}$ is the difference between the global model $\boldsymbol{\theta}^{t-\tau_s}$ and the local model. We note that we assume the server model update can have a delay $\tau_s$, i.e., the server is not required to compute the model

**Algorithm 2** Our AFLGuard.

Server:

Initializes global model $\boldsymbol{\theta}^0 \in \Theta$ and sends it to all clients.
**for** $t = 0, 1, 2, \cdots, T-1$ **do**
  Upon receiving a model update $\boldsymbol{g}_i^{t-\tau_i}$ from a client $i$, retrieves the server model update $\boldsymbol{g}_s^{t-\tau_s}$.
  **if** $\left\|\boldsymbol{g}_i^{t-\tau_i} - \boldsymbol{g}_s^{t-\tau_s}\right\| \leq \lambda \left\|\boldsymbol{g}_s^{t-\tau_s}\right\|$ **then**
    Updates the global model $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \boldsymbol{g}_i^{t-\tau_i}$.
  **else**
    Does not update the global model, i.e., $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t$.
  **end if**
  Sends the global model $\boldsymbol{\theta}^{t+1}$ to client $i$.
**end for**

update using the global model $\boldsymbol{\theta}^t$ in the $t$th iteration. Instead, the server can compute a model update in every $\tau_s$ iterations.

The server accepts $\boldsymbol{g}_i^{t-\tau_i}$ if i) the direction of $\boldsymbol{g}_i^{t-\tau_i}$ does not deviate dramatically from that of $\boldsymbol{g}_s^{t-\tau_s}$ and ii) the magnitude of $\boldsymbol{g}_i^{t-\tau_i}$ is similar to that of $\boldsymbol{g}_s^{t-\tau_s}$. Formally, the server accepts $\boldsymbol{g}_i^{t-\tau_i}$ if the following inequality is satisfied:

$$\left\|\boldsymbol{g}_i^{t-\tau_i} - \boldsymbol{g}_s^{t-\tau_s}\right\| \leq \lambda \left\|\boldsymbol{g}_s^{t-\tau_s}\right\|, \tag{3}$$

where the parameter $\lambda > 0$ can be viewed as a control knob: if $\lambda$ is too small, the server could potentially reject some model updates from benign clients; on the other hand, if $\lambda$ is too large, the server could falsely accept some model updates from malicious clients. Fig. 2 illustrates our acceptance criteria. Once a client's model update is accepted, the server uses it to update the global model based on Eq. (2).

**Algorithm of AFLGuard:** We summarize our AFLGuard algorithm in Algorithm 2. Note that Algorithm 2 only shows the learning procedure of AFLGuard on the server side. The learning procedure on the client side is the same as that of Algorithm 1 and thus we omit it for brevity. In the $t$th iteration, the server decides whether to accept a client's model update or not based on Eq. (3). If yes, the server uses it to update the global model and sends the updated global model back to the client. Otherwise the server does not update the global model and sends the current global model back to the client.

## 5 THEORETICAL SECURITY ANALYSIS

We theoretically analyze the security/robustness of AFLGuard. In particular, we show that the difference between the optimal global model $\boldsymbol{\theta}^*$ under no malicious clients and the global model learnt by AFLGuard with malicious clients can be bounded under some assumptions. We note that simple models like regression can satisfy these assumptions, while more complex models like neural networks may not. Therefore, in the next section, we will empirically evaluate our method on complex neural networks.

For convenience, we define $\boldsymbol{V}$ as the $d$-dimensional unit vector space $\boldsymbol{V} \stackrel{\text{def}}{=} \{\mathbf{v} \in \mathbb{R}^d : \|\mathbf{v}\| = 1\}$, $\nabla f(\boldsymbol{\theta}, X) = \frac{1}{|X|}\sum_{x \in X} \nabla f(\boldsymbol{\theta}, x)$, and $q(\boldsymbol{\theta}, X) \stackrel{\text{def}}{=} \nabla f(\boldsymbol{\theta}, X) - \nabla f(\boldsymbol{\theta}^*, X)$. We use $X_s$ to denote the trusted dataset at the server. Next, we first state the assumptions in our theoretical analysis and then describe our theoretical results.

Assumption 1. *The expected loss $F(\boldsymbol{\theta})$ has $L$-Lipschitz continuous gradients and is $\mu$-strongly convex, i.e., $\forall \boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$, the following inequalities hold:*

$$\left\|\nabla F(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta}')\right\| \leq L\left\|\boldsymbol{\theta} - \boldsymbol{\theta}'\right\|,$$
$$F(\boldsymbol{\theta}) + \left\langle \nabla F(\boldsymbol{\theta}), \boldsymbol{\theta}' - \boldsymbol{\theta}\right\rangle + \frac{\mu}{2}\left\|\boldsymbol{\theta}' - \boldsymbol{\theta}\right\|^2 \leq F(\boldsymbol{\theta}').$$

Assumption 2. *There exist constants $\alpha_1 > 0$ and $\rho_1 > 0$ such that for any $\mathbf{v} \in \boldsymbol{V}$, $\langle \nabla f(\boldsymbol{\theta}^*, X), \mathbf{v}\rangle$ is sub-exponential. That is, $\forall |\varphi| \leq 1/\rho_1$, we have:*

$$\sup_{\mathbf{v} \in \boldsymbol{V}} \mathbb{E}\left[\exp\left(\varphi \left\langle \nabla f(\boldsymbol{\theta}^*, X), \mathbf{v}\right\rangle\right)\right] \leq e^{\alpha_1^2 \varphi^2/2}.$$

Assumption 3. *There exist constants $\alpha_2 > 0$, $\rho_2 > 0$ such that for any $\mathbf{v} \in \boldsymbol{V}$, $\boldsymbol{\theta} \in \Theta$ and $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$. $\langle q(\boldsymbol{\theta}, X) - \mathbb{E}\left[q(\boldsymbol{\theta}, X)\right], \mathbf{v}\rangle / \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|$ is sub-exponential. That is, $\forall |\varphi| \leq 1/\rho_2$, we have:*

$$\sup_{\boldsymbol{\theta} \in \Theta, \mathbf{v} \in \boldsymbol{V}} \mathbb{E}\left[\exp\left(\varphi \left\langle q(\boldsymbol{\theta}, X) - \mathbb{E}\left[q(\boldsymbol{\theta}, X)\right], \mathbf{v}\right\rangle / \left\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\right\|\right)\right] \leq e^{\alpha_2^2 \varphi^2/2}.$$

Assumption 4. *For any $\beta \in (0, 1)$, there exists a constant $H > 0$ such that the following inequality holds:*

$$\mathbb{P}\left\{\sup_{\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta : \boldsymbol{\theta} \neq \boldsymbol{\theta}'} \left\|\frac{1}{|X_s|}\sum_{x \in X_s}\left(\nabla f(\boldsymbol{\theta}, x) - \nabla f(\boldsymbol{\theta}', x)\right)\right\| \leq H\left\|\boldsymbol{\theta} - \boldsymbol{\theta}'\right\|\right\} \geq 1 - \beta/3.$$

Assumption 5. *Clients' local training data are independent and identically distributed (i.i.d.). The trusted dataset held by the server and the overall training data are drawn from the same distribution, and the server delay $\tau_s = 0$.*

Remark. Assumption 1 is satisfied in many learning models (e.g., linear regression and quadratically regularized models). Note that we only assume that the expected loss is strongly-convex, while the empirical loss could still be non-convex. Assumptions 2-3 characterize sub-exponential properties on the gradient vectors. Assumption 2 is a standard assumption in the literature on convergence analysis, while Assumptions 2 and 3 are also widely used in Byzantine-robust FL community (see, e.g., [16, 42, 56]). Assumption 4 is satisfied if the model/loss function is Lipschitz-smooth (e.g., regressions, neural networks). Assumption 5 is a sufficient condition only needed in our theoretical analysis, which characterizes the statistical relations between the server's trusted dataset and the overall training data. Note that we only need these assumptions to provide theoretical analysis of our proposed AFLGuard, and these assumptions are commonly used in the machine learning and security communities in order to establish the convergence of the FL methods [9, 16, 56]. In practice, some of these assumptions may not hold, e.g., clients' local training data could be non-i.i.d., trusted data held by the server and the overall training data may come from different distributions. In Section 6, we will first use a synthetic dataset that satisfies all assumptions to evaluate the performance of our AFLGuard. Then, we will show that AFLGuard can still effectively defend against poisoning attacks in real-world datasets and complex models when some assumptions are violated. As a concrete example, the following lemma shows that linear regression

models satisfy Assumptions 1-4 with appropriate parameters, and the proof is shown in Appendix A.1.

Lemma 5.1. *Let $x_i = (\boldsymbol{u}_i, y_i)$ be the input data and define the loss function as $f(\boldsymbol{\theta}, x_i) = \frac{(\langle \boldsymbol{u}_i, \boldsymbol{\theta} \rangle - y_i)^2}{2}$. Suppose that $y_i$ is generated by a linear regression model $y_i = \langle \boldsymbol{u}_i, \boldsymbol{\theta}^* \rangle + e_i$, where $\boldsymbol{\theta}^*$ is the unknown true model, $\boldsymbol{u}_i \sim N(0, \boldsymbol{I})$, $e_i \sim N(0,1)$ and $e_i$ is independent of $\boldsymbol{u}_i$. The linear regression model satisfies Assumptions 1-4 with the following parameters: i) Assumption 1 is satisfied with $L = 1, \mu = 1$; ii) Assumption 2 is satisfied with $\alpha_1 = \sqrt{2}, \rho_1 = \sqrt{2}$; iii) Assumption 3 is satisfied with $\alpha_2 = \sqrt{8}, \rho_2 = 8$; and iv) Assumption 4 is satisfied with $H = 2log(4/\beta) + 2\sqrt{dlog(4/\beta)} + d$.*

The following theorem shows the security of AFLGuard:

Theorem 1. *Suppose Assumptions 1-5 are satisfied. If the global learning rate in Algorithm 2 satisfies $\eta \leq \frac{2}{\mu+L}$ and each client uses one mini-batch to calculate the model update, then for any number of malicious clients, with probability at least $1-\beta$, we have:*

$$\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\| \leq (1-q)^t \left\|\boldsymbol{\theta}^0 - \boldsymbol{\theta}^*\right\| + 4\eta\Gamma(\lambda+1)/q, \tag{4}$$

*where $q = 1-(\sqrt{1-\frac{2\eta\mu L}{\mu+L}} + \eta L\lambda + 8\eta\Lambda(\lambda+1))$, $\Gamma = \alpha_1\sqrt{2K_1/|X_s|}$, $\Lambda = \alpha_2\sqrt{2(K_2+K_3)/|X_s|}$, $K_1 = d\log 6 + \log(3/\beta)$, $K_2 = d\log(18R/\alpha_2)$, $K_3 = \frac{1}{2}d\log(|X_s|/d) + \log\left(\frac{6\alpha_2^2\epsilon\sqrt{|X_s|}}{\rho_2\alpha_1\beta}\right)$, $R = \max\{L, H\}$, $\epsilon > 0$ is a constant, $d$ is the dimension of $\boldsymbol{\theta}$, and $|X_s|$ is the trusted dataset size.*

Proof. Please see Appendix A.2. □

Remark. Our Theorem 1 shows that the convergence of AFLGuard does not require the trusted dataset size to depend on the number of model parameters, the client dataset sizes, and the number of malicious clients. The trusted dataset size affects the convergence neighborhood size (the second term on the right-hand-side of Eq. (4)). The larger the trusted dataset size, the smaller the convergence neighborhood.

# 6 EMPIRICAL EVALUATION

## 6.1 Experimental Setup

*6.1.1 Compared Methods.* We compare our AFLGuard with the following asynchronous methods:

**1) AsyncSGD [58]:** In AsyncSGD, the server updates the global model according to Algorithm 1 upon receiving a model update from any client.

**2) Kardam [17]:** In Kardam, the server keeps an empirical Lipschitz coefficient for each client, and filters out potentially malicious model updates based on the Lipschitz filter.

**3) BASGD [55]:** In BASGD, the server holds several buffers. Upon receiving a model update from any client, the server stores it into one of these buffers according to a mapping table. When all buffers are non-empty, the server computes the average of model updates in each buffer, takes the median of all buffers, and uses it to update the global model.

**4) Zeno++ [53]:** In Zeno++, the server has a trusted dataset. Upon receiving a client's model update, the server computes a server model update based on the trusted dataset. If the cosine similarity between the server model update and the client's model update is positive, then the server normalizes the client's model update to have the same magnitude as the server model update and uses the normalized model update to update the global model.

*6.1.2 Datasets.* We evaluate AFLGuard and the compared methods using one synthetic dataset and five real-world datasets (MNIST, Fashion-MNIST, Human Activity Recognition (HAR), Colorectal Histology MNIST, CIFAR-10). The synthetic dataset is for linear regression, which satisfies the Assumptions 1-4 in Section 5 and is used to validate our theoretical results. Other datasets are used to train complex models, which aim to show the effectiveness of AFLGuard even if the Assumptions 1-4 are not satisfied. The details of these datasets are shown in Appendix A.3 due to limited space.

*6.1.3 Poisoning Attacks.* We use the following poisoning attacks in our experiments.

**1) Label flipping (LF) attack [56]:** In the LF attack, the label $y$ of each training example in the malicious clients is replaced by $C-1-y$, where $C$ is the total number of classes. For instance, for the MNIST dataset, digit "1" is replaced by digit "8".

**2) Gaussian (Gauss) attack [7]:** In the Gauss attack, each model update from malicious clients is drawn from a zero-mean Gaussian distribution (we set the standard deviation to 200).

**3) Gradient derivation (GD) attack [18]:** In the GD attack adapted from [18], a malicious client computes a model update based on its local training data and then scales it by a negative constant (−10 in our experiments) before sending it to the server.

**4) Backdoor (BD) attack [5, 9, 23]:** BD attack is a targeted poisoning attack. We use the same strategy in [23] to embed the trigger in MNIST, Fashion-MNIST and Colorectal Histology MNIST datasets. Following [9], the target label is set to "WALKING UPSTAIRS" and the trigger is generated by setting every 20th feature to 0 for the HAR dataset. For the CIFAR-10 dataset, the target label is set to "bird" and we use the same pattern trigger as suggested in [5].

**5) Adaptive (Adapt) attack:** In [18], a general adaptive attack framework is proposed to attack FL with any aggregation rule. We apply this attack framework to construct an adaptive attack to our AFLGuard. In particular, the attack framework is designed for synchronized FL, in which the server aggregates model updates from multiple clients to update the global model. The key idea is to craft model updates at the malicious clients such that the aggregated model update deviates substantially from the before-attack one. To apply this general attack framework to AFLGuard, we assume that the server accepts or rejects a client's model update based on AFLGuard and computes the average of the accepted model updates. Then, we craft the model updates on the malicious clients based on the attack framework.

*6.1.4 Evaluation Metrics.* For the synthetic dataset, we use the following two evaluation metrics since it is a regression problem: i) *Mean Squared Error (MSE)*: MSE is computed as $\text{MSE} = \frac{1}{N_t}\sum_{i=1}^{N_t}(\hat{y}_i - y_i)^2$, where $\hat{y}_i$ is the predicted value, $y_i$ is the true value, and $N_t$ is the number of testing examples; ii) *Model Estimation Error (MEE)*: MEE is computed as $\text{MEE} = \|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*\|_2$, where $\hat{\boldsymbol{\theta}}$ is the learnt model and $\boldsymbol{\theta}^*$ is the true model. MEE is commonly used in measuring the performance of linear regression [24, 45]. The five real-world datasets represent classification tasks, and we consider

**Table 1: MSE and MEE of different defenses under different attacks on synthetic dataset. The results are in the form of "MSE / MEE". "$> 1000$" means the value is larger than 1000.**

| | AsyncSGD | Kardam | BASGD | Zeno++ | AFLGuard |
|---|---|---|---|---|---|
| No attack | 0.03 / 0.18 | 0.03 / 0.18 | 0.09 / 1.43 | 0.03 / 0.40 | 0.03 / 0.18 |
| LF attack | 21.05 / 25.75 | 0.04 / 0.60 | 16.71 / 22.70 | 0.03 / 0.40 | 0.03 / 0.18 |
| Gauss attack | 0.78 / 4.82 | 0.03 / 0.36 | 0.85 / 5.32 | 0.03 / 0.40 | 0.03 / 0.18 |
| GD attack | "$> 1000$" / "$> 1000$" | 30.14 / 30.65 | "$> 1000$" / "$> 1000$" | 0.03 / 0.40 | 0.03 / 0.18 |
| Adapt attack | "$> 1000$" / "$> 1000$" | "$> 1000$" / "$> 1000$" | "$> 1000$" / "$> 1000$" | 0.03 / 0.42 | 0.03 / 0.18 |

**Table 2: Test error rates and attack success rates of different defenses under different attacks on real-world datasets. The results of BD attack are in the form of "test error rate / attack success rate".**

**(a) MNIST**

| | AsyncSGD | Kardam | BASGD | Zeno++ | AFLGuard |
|---|---|---|---|---|---|
| No attack | 0.05 | 0.12 | 0.19 | 0.08 | 0.06 |
| LF attack | 0.09 | 0.15 | 0.26 | 0.09 | 0.07 |
| Gauss attack | 0.91 | 0.39 | 0.27 | 0.09 | 0.07 |
| GD attack | 0.90 | 0.90 | 0.89 | 0.09 | 0.07 |
| BD attack | 0.90 / 1.00 | 0.91 / 1.00 | 0.91 / 1.00 | 0.09 / 0.01 | 0.07 / 0.01 |
| Adapt attack | 0.91 | 0.91 | 0.90 | 0.10 | 0.07 |

**(b) Fashion-MNIST**

| | AsyncSGD | Kardam | BASGD | Zeno++ | AFLGuard |
|---|---|---|---|---|---|
| No attack | 0.15 | 0.29 | 0.24 | 0.26 | 0.17 |
| LF attack | 0.19 | 0.29 | 0.24 | 0.29 | 0.21 |
| Gauss attack | 0.90 | 0.29 | 0.35 | 0.28 | 0.19 |
| GD attack | 0.90 | 0.90 | 0.90 | 0.29 | 0.21 |
| BD attack | 0.90 / 1.00 | 0.90 / 1.00 | 0.90 / 1.00 | 0.29 / 0.05 | 0.20 / 0.04 |
| Adapt attack | 0.90 | 0.90 | 0.90 | 0.29 | 0.21 |

**(c) HAR**

| | AsyncSGD | Kardam | BASGD | Zeno++ | AFLGuard |
|---|---|---|---|---|---|
| No attack | 0.05 | 0.06 | 0.07 | 0.06 | 0.05 |
| LF attack | 0.19 | 0.22 | 0.08 | 0.08 | 0.05 |
| Gauss attack | 0.30 | 0.23 | 0.24 | 0.07 | 0.05 |
| GD attack | 0.83 | 0.48 | 0.67 | 0.08 | 0.05 |
| BD attack | 0.18 / 0.47 | 0.17 / 0.02 | 0.41 / 0.28 | 0.07 / 0.01 | 0.05 / 0.01 |
| Adapt attack | 0.93 | 0.52 | 0.90 | 0.08 | 0.05 |

**(d) Colorectal Histology MNIST**

| | AsyncSGD | Kardam | BASGD | Zeno++ | AFLGuard |
|---|---|---|---|---|---|
| No attack | 0.21 | 0.28 | 0.29 | 0.31 | 0.22 |
| LF attack | 0.29 | 0.37 | 0.40 | 0.39 | 0.23 |
| Gauss attack | 0.65 | 0.44 | 0.61 | 0.43 | 0.22 |
| GD attack | 0.87 | 0.68 | 0.87 | 0.39 | 0.32 |
| BD attack | 0.75 / 0.84 | 0.67 / 0.02 | 0.85 / 0.84 | 0.44 / 0.02 | 0.27 / 0.02 |
| Adapt attack | 0.88 | 0.88 | 0.88 | 0.64 | 0.33 |

**(e) CIFAR-10**

| | AsyncSGD | Kardam | BASGD | Zeno++ | AFLGuard |
|---|---|---|---|---|---|
| No attack | 0.26 | 0.29 | 0.47 | 0.41 | 0.26 |
| LF attack | 0.40 | 0.52 | 0.54 | 0.53 | 0.34 |
| Gauss attack | 0.88 | 0.63 | 0.81 | 0.52 | 0.33 |
| GD attack | 0.90 | 0.90 | 0.90 | 0.60 | 0.30 |
| BD attack | 0.76 / 0.99 | 0.82 / 1.00 | 0.74 / 0.98 | 0.49 / 0.06 | 0.29 / 0.01 |
| Adapt attack | 0.90 | 0.90 | 0.90 | 0.82 | 0.36 |

the following two evaluation metrics: 1) *test error rate*, which is the fraction of clean testing examples that are misclassified; and 2) *attack success rate*, which is the fraction of trigger-embedded testing inputs that are predicted as the attacker-chosen target label. Note that attack success rate is only applicable for targeted poisoning attack (i.e., BD attack in our experiments). The smaller the error (MSE, MEE, or test error rate) and attack success rate, the better the defense. Note that we do not consider targeted poisoning attacks on synthetic dataset since there are no such attacks designed for linear regression.

*6.1.5 Parameter Setting.* We assume 100 clients ($n = 100$) for synthetic, MNIST, and Fashion-MNIST datasets, and 40 clients ($n = 40$) for Colorectal Histology MNIST and CIFAR-10 datasets. The HAR dataset is collected from smartphones of 30 real-world users, and each user is considered as a client. Thus, there are 30 clients ($n = 30$) in total for HAR. By default, we assume 20% of the clients are malicious. We train a convolutional neural network (CNN) on MNIST and Fashion-MNIST datasets, and its architecture is shown in Table 4 in Appendix. We train a logistic regression classifier on HAR dataset. We train a ResNet-20 [25] model for Colorectal Histology MNIST and CIFAR-10 datasets. We set 2,000, 2,000, 6,000, 1,000, 20,000 and 20,000 iterations for synthetic, MNIST, Fashion-MNIST, HAR, Colorectal Histology MNIST and CIFAR-10 datasets, respectively. The batch sizes for the six datasets are 16, 32, 64, 32, 32 and 64, respectively. The learning rates are set to 1/1600, 1/320 for synthetic and HAR datasets, respectively; and are set to 1/3200 for the other four datasets. We use different parameters for different datasets because they have different data statistics. In the synthetic dataset, we assume the clients' local training data are i.i.d. However, the local training data non-i.i.d. across clients in the five real-world datasets. In particular, we use the approach in [18] to simulate the non-i.i.d. setting. The non-i.i.d. degree is set to 0.5 for MNIST, Fashion-MNIST, Colorectal Histology MNIST, and CIFAR-10 datasets. Note that each user is a client in HAR dataset, and thus the clients' local training data are already heterogeneous for HAR.

In AFLGuard, the trusted dataset size is set to 100 for all six datasets. By default, for the synthetic data, we assume that the trusted dataset held by the server and the overall training data are generated from the same distribution. For the real-world datasets, we do not make this assumption. We will empirically show that our method works well even if the distribution of trusted data deviates from that of the overall training data, i.e., there exists a *distribution shift (DS)* between these two datasets. The larger the DS, the larger the deviation between the trusted and overall training datasets. In our experiments, we simulate DS in the following way: a fraction of samples in the trusted dataset are drawn from one particular

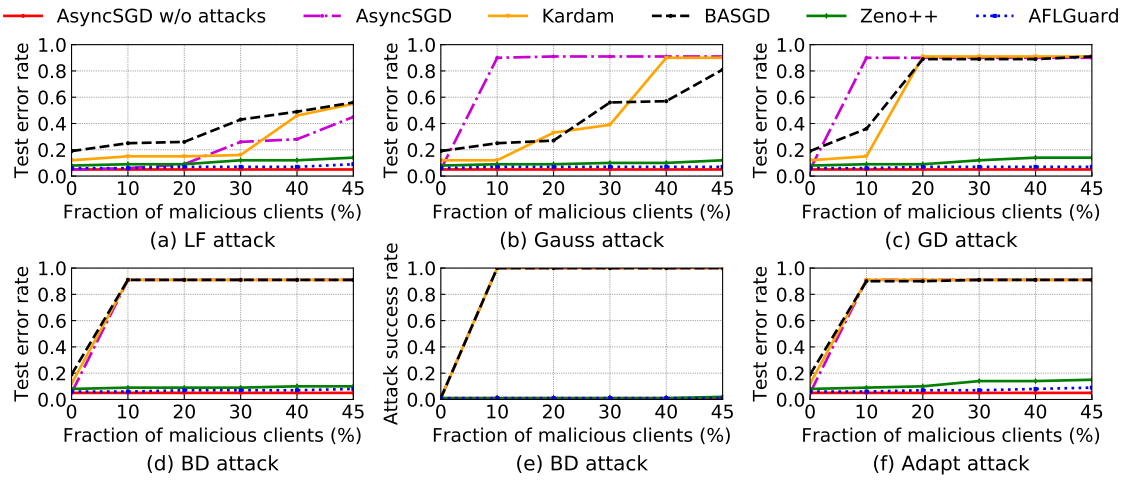



**Figure 3: Test error rates and attack success rates of different defenses under different attacks with different fraction of malicious clients on MNIST dataset.**

class (the first class in our experiments) of the overall training data, and the remaining samples in the trusted dataset are drawn from the remaining classes of the overall training data uniformly at random. We use this fraction value as a proxy for DS. Note that when the trusted and overall training datasets are drawn from the same distribution, DS is equal to $1/C$, where $C$ is the total number of classes. By default, we set DS to 0.5 for the five real-world datasets.

In our experiments, we use a separate validation dataset to tune the parameter $\lambda$ in AFLGuard. Note that this validation dataset is different from the trusted dataset held by the server. We use the validation dataset to tune the hyperparameter of AFLGuard, while the server in AFLGuard uses the trusted dataset to filter out potential malicious information. The size of the validation dataset is 200. The validation data and the overall training data (the union of the local training data of all clients) are from the same distribution. For example, there are 10 classes in MNIST dataset. We sample 20 training examples from each class of the overall training data uniformly at random. After fine tuning the parameter, we set $\lambda = 1.5$ for synthetic, MNIST, and HAR datasets, and $\lambda = 1.8$ for the other three datasets. The server updates $\boldsymbol{g}_s^{t-\tau_s}$ every 10 ($\tau_s = 10$) iterations by default. We use the approach in [53] to simulate asynchrony. We sample client delay from the interval $[0, \tau_{\max}]$ uniformly at random, where $\tau_{\max}$ is the maximum client delay. We set $\tau_{\max} = 10$ by default.

## 6.2 Experimental Results

**AFLGuard is Effective:** We first show results on the linear regression model for synthetic dataset, which satisfies Assumptions 1-4 in Section 5 to support our theoretical results. The MSE and MEE of different methods under different attacks on synthetic dataset are shown in Table 1. We observe that AFLGuard is robust in both non-adversarial and adversarial settings. In particular, the MSE and MEE of AFLGuard under various attacks are the same as those of AsyncSGD without attacks. Moreover, we also observe that AFLGuard outperforms the compared methods. For instance, the MSE and MEE of BASGD are both larger than 1,000 under the GD and Adapt attacks.

Next, we show results on the five real-world datasets. The test error rates and attack success rates of different methods under different attacks are shown in Table 2. "No attack" in Table 2 represents the test error rate without any attacks. For the untargeted poisoning attacks (LF attack, Gauss attack, GD attack, and Adapt attack), the results are the test error rates; and for the targeted poisoning attacks (BD attack), the results are in the form of "test error rate / attack success rate". We note that only using the trusted data held by the server to update the global model can not achieve satisfactory accuracy. For instance, the test error rate is 0.21 when we only use the trusted data of the server to update the global model on the MNIST dataset. We also remark that our asynchronous AFLGuard algorithm achieves a performance similar to its synchronous counterpart. For instance, on MNIST, the test error rates of synchronous AFLGuard under LF, Gauss, and GD attacks are all 0.05.

First, we observe that AFLGuard is effective in non-adversarial settings. When there are no malicious clients, AFLGuard has similar test error rate as AsyncSGD. For instance, on MNIST, the test error rates without attacks are respectively 0.05 and 0.06 for AsyncSGD and AFLGuard, while the test error rates are respectively 0.12 and 0.19 for Kardam and BASGD. Second, AFLGuard is robust against various poisoning attacks and outperforms all baselines. For instance, the test error rate of Kardam increases to 0.90 under the GD attack on the MNIST and Fashion-MNIST datasets, while the test error rates are 0.07 and 0.21 for AFLGuard under the same setting. Likewise, the attack success rates of AFLGuard are at most 0.04 for all real-world datasets, while the attack success rates of AsyncSGD, Kardam, and BASGD are high. Note that in Table 1, we use synthetic data that satisfies Lemma 5.1 to evaluate the performance of our AFLGuard. Since the variance of the synthetic data is small, Zeno++ and AFLGuard have similar MSE and MEE. However, Table 2 shows that, for real-world datasets, AFLGuard significantly outperforms Zeno++.

**Impact of the Fraction of Malicious Clients:** Fig. 3 illustrates the test error rates and attack success rates of different methods

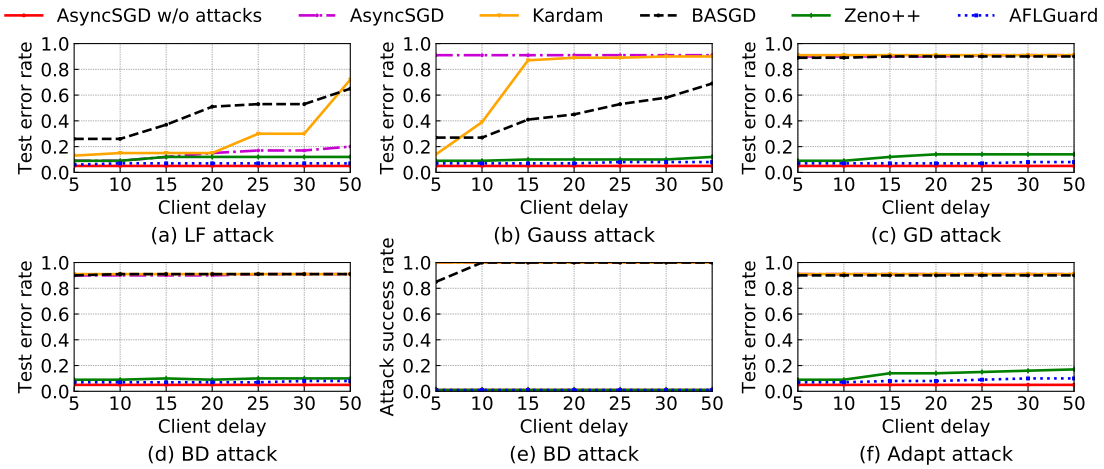



**Figure 4: Test error rates and attack success rates of different defenses under different attacks with different client delays on MNIST dataset.**

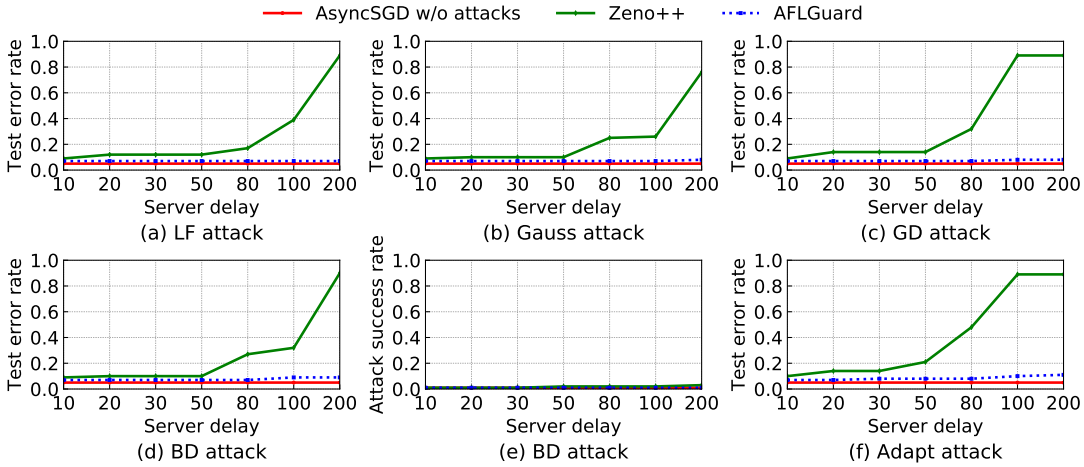



**Figure 5: Test error rates and attack success rates of Zeno++ and AFLGuard under different attacks with different server delays on MNIST dataset.**

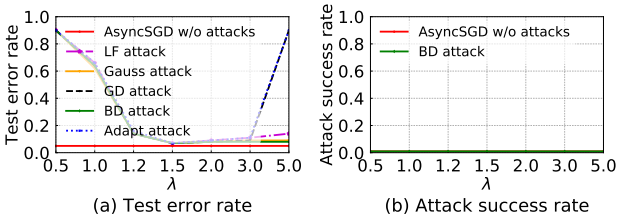



**Figure 6: Test error rates and attack success rates of AFLGuard under different attacks with different $\lambda$ on MNIST dataset.**

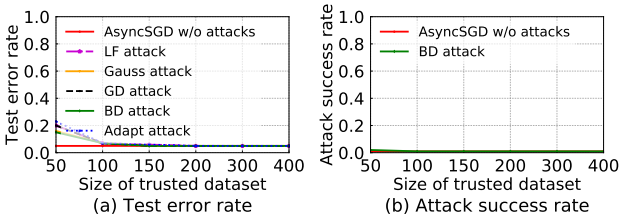



**Figure 7: Test error rates and attack success rates of AFLGuard under different attacks with different size of trusted dataset on MNIST dataset.**

under different attacks on the MNIST dataset, when the fraction of malicious clients increases from 0 to 45%. Note that Fig. 3(e) shows the attack success rates of different defenses under BD attack, while other figures are the test error rates of different defenses under untargeted and targeted poisoning attacks. We observe that AFLGuard achieves a test error rate similar to that of AsyncSGD without attacks, when 45% of clients are malicious. This shows that AFLGuard is robust against a large fraction of malicious clients.

**Impact of the Number of Clients:** Fig. 8 in Appendix shows the results of different defenses under different attacks, when the total

**Table 3: Test error rates and attack success rates of Zeno++ and AFLGuard under different attacks with different distribution shifts (DSs) between the trusted data and overall training data on MNIST dataset. The results of BD attack are in the form of "test error rate / attack success rate".**

| DS | 0.1 | | 0.5 | | 0.6 | | 0.8 | | 1.0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Attack | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard |
| No attack | 0.05 | 0.05 | 0.08 | 0.06 | 0.10 | 0.06 | 0.55 | 0.06 | 0.88 | 0.11 |
| LF attack | 0.07 | 0.05 | 0.09 | 0.07 | 0.12 | 0.07 | 0.86 | 0.07 | 0.89 | 0.11 |
| Gauss attack | 0.07 | 0.05 | 0.09 | 0.07 | 0.12 | 0.07 | 0.59 | 0.07 | 0.89 | 0.12 |
| GD attack | 0.07 | 0.06 | 0.09 | 0.07 | 0.12 | 0.07 | 0.78 | 0.08 | 0.89 | 0.12 |
| BD attack | 0.06 / 0.01 | 0.05 / 0.01 | 0.09 / 0.01 | 0.07 / 0.01 | 0.11 / 0.01 | 0.07 / 0.01 | 0.55 / 0.03 | 0.08 / 0.01 | 0.90 / 0.01 | 0.11 / 0.01 |
| Adapt attack | 0.07 | 0.06 | 0.10 | 0.07 | 0.12 | 0.08 | 0.88 | 0.10 | 0.90 | 0.12 |

number of clients $n$ varies from 50 to 500. The fraction of malicious clients is set to 20%. We observe that our AFLGuard can effectively defend against various poisoning attacks for different number of clients. In particular, AFLGuard under attacks achieves test error rates similar to AsyncSGD without attack.

**Impact of the Client Delay:** A challenge and key feature in asynchronous FL is the delayed client model updates. In this experiment, we investigate the impact of the maximum client delays $\tau_{\max}$ on the test error rate of different defenses under different attacks on the MNIST dataset, where the server delay is set to 10. The results are shown in Fig. 4. We observe that AFLGuard is insensitive to the delays on the client side, and the test error rates remain almost unchanged when the client delay varies from 5 to 50. However, Kardam and BASGD are highly sensitive to client delays. For example, under the Gauss attack, the test error rate of Kardam increases from 0.14 to 0.39 when the client delay increases from 5 to 10. Moreover, under the GD and Adapt attacks, the test error rates of Kardam and BASGD are both 0.90 when the client delay is only 5.

**Impact of the Server Delay:** In both Zeno++ and our AFLGuard, the server uses server model update. In this experiment, we investigate the impact of server delays on the performance of Zeno++ and AFLGuard under different attacks, where the maximum client delay is set to 10. The results are shown in Fig. 5. We observe that AFLGuard can effectively defend against various poisoning attacks with different server delays. AFLGuard under attacks has test error rates similar to those of AsyncSGD under no attacks when the server delay ranges from 10 to 200. However, Zeno++ is highly sensitive to server delays. For instance, Zeno++ can only resist the Adapt attack up to 80 server delays.

**Impact of $\lambda$:** Fig. 6 shows the test error rates and attack success rates of AFLGuard under different attacks with different $\lambda$ values on the MNIST dataset. We observe that if $\lambda$ is too small (e.g., $\lambda = 0.5$), the test error rate of AFLGuard is large since the server rejects many benign model updates. When $\lambda$ is large (e.g., $\lambda = 5.0$), the test error rates of AFLGuard under the GD and Adapt attacks are large. This is because the server falsely accepts some model updates from malicious clients.

**Impact of the Trusted Dataset:** The trusted dataset can be characterized by its size and distribution. Therefore, we explore the impact of both its size and distribution. Fig. 7 shows the results of AFLGuard under different attacks on the MNIST dataset, when the size of the trusted dataset increases from 50 to 400 (other parameters are set to their default values). We find that AFLGuard only requires a small trusted dataset (e.g., 100 examples) to defend against different attacks.

Table 3 shows the results of Zeno++ and AFLGuard under different attacks when the DS value between the trusted data and overall training data varies on the MNIST dataset. The results on the other four real-world datasets are shown in Table 5 in Appendix. Note that, for the synthetic data, we assume that the trusted data and the overall training data are generated from the same distribution. Thus, there is no need to study the impact of DS on synthetic data. First, we observe that AFLGuard outperforms Zeno++ across different DS values in most cases, especially when DS is large. This is because Zeno++ classifies a client's model update as benign if it is not negatively correlated with the (delayed) server model update. However, when the trusted dataset deviates substantially from the overall training dataset, it is very likely that the server model update and the model updates from benign clients are not positively correlated. Second, AFLGuard outperforms Zeno++ even if the trusted data has the same distribution as that of overall training data (corresponding to DS being 0.1 for MNIST, Fashion-MNIST and CIFAR-10 datasets, 0.167 for HAR dataset, and 0.125 for Colorectal Histology MNIST dataset). Third, AFLGuard can tolerate a large DS value, which means that AFLGuard does not require the trusted dataset to have similar distribution with the overall training data.

## 7 CONCLUSION AND FUTURE WORK

In this paper, we propose a Byzantine-robust asynchronous FL framework called AFLGuard to defend against poisoning attacks in asynchronous FL. In AFLGuard, the server holds a small and clean trusted dataset to assist the filtering of model updates from malicious clients. We theoretically analyze the security guarantees of AFLGuard. We extensively evaluate AFLGuard against state-of-the-art and adaptive poisoning attacks on one synthetic and five real-world datasets. Our results show that AFLGuard effectively mitigates poisoning attacks and outperforms existing Byzantine-robust asynchronous FL methods. One interesting future work is to investigate the cases where the server has no knowledge of the training data domain.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers and shepherd Briland Hitaj for their constructive comments. This work was supported by NSF grant CAREER CNS-2110259, CNS-2112471, CNS-2102233, CNS-2131859, CNS-2112562, CNS-2125977 and CCF-2110252, as well as ARO grant No. W911NF2110182.

# REFERENCES


[1] [n.d.]. *Federated Learning: Collaborative Machine Learning without Centralized Training Data.* https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[2] [n.d.]. *Making federated learning faster and more scalable: A new asynchronous method.* https://ai.facebook.com/blog/asynchronous-federated-learning/

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[4] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra Perez, and Jorge Luis Reyes Ortiz. 2013. A public domain dataset for human activity recognition using smartphones. In *ESANN*.

[5] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *AISTATS*.

[6] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In *ICML*.

[7] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NeurIPS*.

[8] Sébastien Bubeck. 2014. Convex optimization: Algorithms and complexity. *arXiv preprint arXiv:1405.4980* (2014).

[9] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2021. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. In *NDSS*.

[10] Xiaoyu Cao and Neil Zhenqiang Gong. 2022. MPAF: Model Poisoning Attacks to Federated Learning based on Fake Clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3396–3404.

[11] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. Provably Secure Federated Learning against Malicious Clients. In *AAAI*.

[12] Xinyang Cao and Lifeng Lai. 2019. Distributed gradient descent algorithm robust to an arbitrary number of byzantine attackers. In *IEEE Transactions on Signal Processing*.

[13] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. 2020. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081* (2020).

[14] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[15] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous online federated learning for edge devices with non-iid data. In *Big Data*.

[16] Yudong Chen, Lili Su, and Jiaming Xu. 2017. Distributed Statistical Machine Learning in Adversarial Settings: Byzantine Gradient Descent. In *POMACS*.

[17] Georgios Damaskinos, Rachid Guerraoui, Rhicheek Patra, Mahsa Taziki, et al. 2018. Asynchronous Byzantine machine learning (the case of SGD). In *ICML*.

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

[19] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. 2020. Influence function based data poisoning attacks to top-n recommender systems. In *Proceedings of The Web Conference*.

[20] Minghong Fang, Minghao Sun, Qi Li, Neil Zhenqiang Gong, Jin Tian, and Jia Liu. 2021. Data poisoning attacks and defenses to crowdsourcing systems. In *Proceedings of The Web Conference*.

[21] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning attacks to graph-based recommender systems. In *ACSAC*.

[22] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2020. The limitations of federated learning in sybil settings. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. 301–316.

[23] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[24] Nirupam Gupta and Nitin H Vaidya. 2019. Byzantine fault tolerant distributed linear regression. *arXiv preprint arXiv:1903.08752* (2019).

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

[26] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. 2022. Papaya: Practical, private, and scalable federated learning. *Proceedings of Machine Learning and Systems* 4 (2022), 814–832.

[27] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[28] Jakob Nikolas Kather, Cleo-Aron Weis, Francesco Bianconi, Susanne M Melchers, Lothar R Schad, Timo Gaiser, Alexander Marx, and Frank Gerrit Zöllner. 2016. Multi-class texture analysis in colorectal cancer histology. *Scientific reports* 6 (2016), 27988.

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

[30] Yann LeCun, Corinna Cortes, and CJ Burges. 1998. MNIST handwritten digit database. *Available: http://yann. lecun. com/exdb/mnist* (1998).

[31] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *NeurIPS*.

[32] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*.

[33] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. 2018. The hidden vulnerability of distributed learning in byzantium. In *ICML*.

[34] Chenglin Miao, Qi Li, Lu Su, Mengdi Huai, Wenjun Jiang, and Jing Gao. 2018. Attack under disguise: An intelligent data poisoning attack mechanism in crowdsourcing. In *Proceedings of The Web Conference*.

[35] Luis Muñoz-González, Kenneth T Co, and Emil C Lupu. 2019. Byzantine-robust federated machine learning through adaptive model averaging. *arXiv preprint arXiv:1909.05125* (2019).

[36] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated learning with buffered asynchronous aggregation. In *AISTATS*.

[37] Thien Duc Nguyen, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Shaza Zeitouni, et al. 2022. FLAME: Taming Backdoors in Federated Learning. In *USENIX Security Symposium*.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*.

[39] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J Doug Tygar. 2009. Antidote: understanding and defending against poisoning of anomaly detectors. In *IMC*.

[40] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NeurIPS*.

[41] Virat Shejwalkar and Amir Houmansadr. 2021. Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning. In *NDSS*.

[42] Lili Su and Jiaming Xu. 2019. Securing distributed gradient descent in high dimensional statistical learning. In *POMACS*.

[43] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. 2019. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963* (2019).

[44] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. 2017. Gradient coding: Avoiding stragglers in distributed learning. In *ICML*.

[45] Berkay Turan, Cesar A Uribe, Hoi-To Wai, and Mahnoosh Alizadeh. 2021. Robust Distributed Optimization With Randomly Corrupted Gradients. *arXiv preprint arXiv:2106.14956* (2021).

[46] Marten van Dijk, Nhuong V Nguyen, Toan N Nguyen, Lam M Nguyen, Quoc Tran-Dinh, and Phuong Ha Nguyen. 2020. Asynchronous Federated Learning with Reduced Number of Rounds and with Differential Privacy from Less Aggregated Gaussian Noise. *arXiv preprint arXiv:2007.09208* (2020).

[47] Roman Vershynin. 2010. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027* (2010).

[48] Martin J Wainwright. 2019. High-dimensional statistics: A non-asymptotic viewpoint, Vol. 48. Cambridge University Press.

[49] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. 2020. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*.

[50] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:cs.LG/cs.LG/1708.07747

[51] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. 2019. Dba: Distributed backdoor attacks against federated learning. In *ICLR*.

[52] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).

[53] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2020. Zeno++: Robust fully asynchronous SGD. In *ICML*.

[54] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. 2021. Asynchronous federated learning on heterogeneous devices: A survey. *arXiv preprint arXiv:2109.04269* (2021).

[55] Yi-Rui Yang and Wu-Jun Li. 2021. BASGD: Buffered Asynchronous SGD for Byzantine Learning. In *ICML*.

[56] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *ICML*.

[57] Zaixi Zhang, Xiaoyu Cao, Jinayuan Jia, and Neil Zhenqiang Gong. 2022. FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients. In *KDD*.

[58] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. 2017. Asynchronous stochastic gradient descent with delay compensation. In *ICML*.

# A APPENDIX

## A.1 Proof of Lemma 5.1

The proof of Lemma 5.1 is mainly from [16, 42]. We first check Assumption 1. Consider the linear regression model defined in Lemma 5.1, that is $y_i = \langle \boldsymbol{u}_i, \boldsymbol{\theta}^* \rangle + e_i$, where $\boldsymbol{\theta}^*$ is the unknown true model parameter, $\boldsymbol{u}_i \sim N(0, \boldsymbol{I})$, $e_i \sim N(0,1)$, $e_i$ is independent of $\boldsymbol{u}_i$. The population risk of (1) is given by $\min_{\boldsymbol{\theta}} \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2 + \frac{1}{2}$. $F(\boldsymbol{\theta}) \triangleq \mathbb{E}[f(\boldsymbol{\theta}, X)] = \mathbb{E}\left[\frac{1}{2}(\langle \boldsymbol{u}, \boldsymbol{\theta}\rangle - y)^2\right] = \mathbb{E}\left[\frac{1}{2}(\langle \boldsymbol{u}, \boldsymbol{\theta}\rangle - \langle \boldsymbol{u}, \boldsymbol{\theta}^*\rangle - e)^2\right] = \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2 + \frac{1}{2}$. Then the gradient of population risk is $\nabla F(\boldsymbol{\theta}) = \boldsymbol{\theta} - \boldsymbol{\theta}^*$. We can see that the population risk $F(\cdot)$ is $L$-Lipschitz continuous with $L = 1$, and $\mu$-strongly convex with $\mu = 1$.

We then check Assumption 2. Let $\boldsymbol{u} \sim N(0, \boldsymbol{I})$, $e \sim N(0,1)$ and $e$ is independent of $\boldsymbol{u}$, then one has $\nabla f(\boldsymbol{\theta}, X) = \boldsymbol{u}\langle \boldsymbol{u}, \boldsymbol{\theta} - \boldsymbol{\theta}^*\rangle - \boldsymbol{u}e$. Let $\mathbf{v} \in \boldsymbol{V}$ be the unit vector, we further have that:

$$\left\langle \nabla f(\boldsymbol{\theta}^*, X), \mathbf{v} \right\rangle = -e\langle \boldsymbol{u}, \mathbf{v}\rangle. \tag{5}$$

Since $\boldsymbol{u} \sim N(0, \boldsymbol{I})$, $\mathbf{v}$ is the unit vector and $\boldsymbol{u}$ is independent of $e$, so we have $\langle \boldsymbol{u}, \mathbf{v}\rangle \sim N(0,1)$ and $\langle \boldsymbol{u}, \mathbf{v}\rangle$ is independent of $e$. According to the standard conditioning argument, for $\varphi^2 \le 1$, one has:

$$\begin{aligned}
&\mathbb{E}\left[\exp\left(\varphi\left\langle \nabla f(\boldsymbol{\theta}^*, X), \mathbf{v}\right\rangle\right)\right] \overset{(a)}{=} \mathbb{E}\left[\exp\left(-\varphi e\langle \boldsymbol{u}, \mathbf{v}\rangle\right)\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\exp\left(-\varphi y\langle \boldsymbol{u}, \mathbf{v}\rangle\right)|\varphi = y\right]\right] \\
&\overset{(b)}{=} \mathbb{E}\left[\exp\left(\varphi^2 e^2/2\right)\right] \overset{(c)}{=} \left(1-\varphi^2\right)^{-1/2} \overset{(d)}{\le} e^{\varphi^2},
\end{aligned} \tag{6}$$

where $(a)$ is because of Eq. (5); $(b)$ is obtained by applying the moment generating function of Gaussian distribution; $(c)$ is because for the moment generating function of $\chi^2$ distribution, we have $\mathbb{E}\left[\exp\left(t\varphi^2\right)\right] = (1-2t)^{-1/2}$ for $t < 1/2$; $(d)$ is due to the fact that $1 - \varphi^2 \ge e^{-2\varphi^2}$ for $|\varphi| \le 1/\sqrt{2}$. Therefore, Assumption 2 holds when $\alpha_1 = \sqrt{2}$ and $\rho_1 = \sqrt{2}$.

Next, we check Assumption 3. As $\nabla f(\boldsymbol{\theta}, X) = \boldsymbol{u}\langle \boldsymbol{u}, \boldsymbol{\theta} - \boldsymbol{\theta}^*\rangle - \boldsymbol{u}e$, $\nabla f(\boldsymbol{\theta}^*, X) = -\boldsymbol{u}e$, so $q(\boldsymbol{\theta}, X) = \nabla f(\boldsymbol{\theta}, X) - \nabla f(\boldsymbol{\theta}^*, X) = \boldsymbol{u}\langle \boldsymbol{u}, \boldsymbol{\theta} - \boldsymbol{\theta}^*\rangle$. As $\mathbb{E}[q(\boldsymbol{\theta}, X)] = \boldsymbol{\theta} - \boldsymbol{\theta}^*$, so $\langle q(\boldsymbol{\theta}, X) - \mathbb{E}[q(\boldsymbol{\theta}, X)], \mathbf{v}\rangle = \langle \boldsymbol{u}, \boldsymbol{\theta} - \boldsymbol{\theta}^*\rangle\langle \boldsymbol{u}, \mathbf{v}\rangle - \langle \boldsymbol{\theta} - \boldsymbol{\theta}^*, \mathbf{v}\rangle$.

For a fixed $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, $\boldsymbol{\theta} \ne \boldsymbol{\theta}^*$, we let $\eth = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| > 0$. We further decompose $\boldsymbol{\theta} - \boldsymbol{\theta}^*$ as $\boldsymbol{\theta} - \boldsymbol{\theta}^* = \sqrt{c_1}\mathbf{v} + \sqrt{c_2}\hat{\mathbf{v}}$, where $\hat{\mathbf{v}}$ is an vector perpendicular to $\mathbf{v}$, $c_1 + c_2 = \eth^2$. We further have that $\langle \boldsymbol{u}, \hat{\mathbf{v}}\rangle \sim N(0,1)$ and:

$$\begin{aligned}
&\left\langle \boldsymbol{u}, \boldsymbol{\theta} - \boldsymbol{\theta}^*\right\rangle\langle \boldsymbol{u}, \mathbf{v}\rangle - \left\langle \boldsymbol{\theta} - \boldsymbol{\theta}^*, \mathbf{v}\right\rangle \\
&= \sqrt{c_1}\left(\langle \boldsymbol{u}, \mathbf{v}\rangle^2 - 1\right) + \sqrt{c_2}\langle \boldsymbol{u}, \hat{\mathbf{v}}\rangle\langle \boldsymbol{u}, \mathbf{v}\rangle.
\end{aligned} \tag{7}$$

One also has $\mathbb{E}[\langle \boldsymbol{u}, \hat{\mathbf{v}}\rangle\langle \boldsymbol{u}, \mathbf{v}\rangle] = \mathbb{E}\left[\hat{\mathbf{v}}^\top \mathbf{u}\mathbf{u}^\top \mathbf{v}\right] = \hat{\mathbf{v}}^\top \mathbb{E}\left[\mathbf{u}\mathbf{u}^\top\right]\mathbf{v} = 0$, where $\mathbf{u}^\top$ is the transpose of $\mathbf{u}$. Hence, $\langle \boldsymbol{u}, \hat{\mathbf{v}}\rangle$ and $\langle \boldsymbol{u}, \mathbf{v}\rangle$ are mutually independent. For any $\varphi$ satisfies $\varphi\sqrt{c_1} < 1/4$ and $\varphi^2 c_2 < 1/4$, we have:

$$\begin{aligned}
&\mathbb{E}\left[\exp\left(\varphi\langle q(\boldsymbol{\theta}, X) - \mathbb{E}[q(\boldsymbol{\theta}, X)], \mathbf{v}\rangle\right)\right] \\
&\overset{(a)}{=} \mathbb{E}\left[\exp\left(\varphi\sqrt{c_1}\left(\langle \boldsymbol{u}, \mathbf{v}\rangle^2 - 1\right) + \varphi\sqrt{c_2}\langle \boldsymbol{u}, \hat{\mathbf{v}}\rangle\langle \boldsymbol{u}, \mathbf{v}\rangle\right)\right] \\
&\overset{(b)}{\le} \sqrt{\mathbb{E}\left[e^{2\varphi\sqrt{c_1}\left(\langle \boldsymbol{u}, \mathbf{v}\rangle^2 - 1\right)}\right]\mathbb{E}\left[e^{2\varphi\sqrt{c_2}\langle \boldsymbol{u}, \hat{\mathbf{v}}\rangle\langle \boldsymbol{u}, \mathbf{v}\rangle}\right]} \\
&= e^{-\varphi\sqrt{c_1}}\sqrt{\mathbb{E}\left[e^{2\varphi\sqrt{c_1}\left(\langle \boldsymbol{u}, \mathbf{v}\rangle^2\right)}\right]\mathbb{E}\left[e^{2\varphi\sqrt{c_2}\langle \boldsymbol{u}, \hat{\mathbf{v}}\rangle\langle \boldsymbol{u}, \mathbf{v}\rangle}\right]} \\
&\overset{(c)}{=} e^{-\varphi\sqrt{c_1}}\left(1 - 4\varphi\sqrt{c_1}\right)^{-1/4}\left(1 - 4\varphi^2 c_2\right)^{-1/4},
\end{aligned} \tag{8}$$

where $(a)$ holds by plugging in Eq. (7); $(b)$ is true by applying the Cauchy-Schwartz's inequality; $(c)$ is true by applying the moment generating function of $\chi^2$ distribution. Since $1 - t \ge e^{-4t}$ for $0 \le t \le 1/2$, and $e^{-t}/\sqrt{1-2t} \le e^{2t^2}$ for $|t| \le 1/4$. Thus, for $\varphi^2 \le 1/(64\eth^2)$, one has $\mathbb{E}\left[\exp\left(\varphi\langle q(\boldsymbol{\theta}, X) - \mathbb{E}[q(\boldsymbol{\theta}, X)], \mathbf{v}\rangle\right)\right] \le \exp\left(4\varphi^2(c_1 + c_2)\right) \le \exp\left(4\varphi^2\eth^2\right)$. Therefore, Assumption 3 holds with $\alpha_2 = \sqrt{8}$ and $\rho_2 = 8$.

Last, we check Assumption 4. As $\nabla f(\boldsymbol{\theta}, X) = \boldsymbol{u}\langle \boldsymbol{u}, \boldsymbol{\theta} - \boldsymbol{\theta}^*\rangle - \boldsymbol{u}e$, then $\nabla^2 f(\boldsymbol{\theta}, X) = \boldsymbol{u}\boldsymbol{u}^\top$, thus it suffices to show the following $\mathbb{P}\left\{\left\|\frac{1}{|X_s|}\sum_{x \in X_s}\nabla^2 f(\boldsymbol{\theta}, x)\right\| \le H\right\} = \mathbb{P}\left\{\left\|\frac{1}{|X_s|}\sum_{j=1}^{|X_s|}\boldsymbol{u}_j\boldsymbol{u}_j^\top\right\| \le H\right\} \ge 1 - \beta/3$. Let $\boldsymbol{U} = \left[\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_{|X_s|}\right] \subset \mathbb{R}^{d \times |X_s|}$, one has $\sum_{j=1}^{|X_s|}\boldsymbol{u}_j\boldsymbol{u}_j^\top = \boldsymbol{U}\boldsymbol{U}^\top$ and $\mathbb{P}\left\{\left\|\frac{1}{|X_s|}\sum_{j=1}^{|X_s|}\boldsymbol{u}_j\boldsymbol{u}_j^\top\right\| \le H\right\} = \mathbb{P}\left\{\|\boldsymbol{U}\| \le \sqrt{|X_s|\,H}\right\}$. Since $\boldsymbol{U}$ is an i.i.d. standard Gaussian matrix, then according to [47], for $t \ge 0$, we have that: $\mathbb{P}\left\{\|\boldsymbol{U}\| \le \sqrt{|X_s|} + \sqrt{d} + t\right\} \ge 1 - \exp\left(-t^2/2\right)$. Setting $H = \left(\sqrt{|X_s|} + \sqrt{d} + \sqrt{2\log(4/\beta)}\right)^2/|X_s|$ and $t = \sqrt{2\log(4/\beta)}$ to complete the proof.

## A.2 Proof of Theorem 1

Since we assume that the server delay $\tau_s$ is zero, i.e., $\tau_s = 0$. Then in the following, we use $\boldsymbol{g}_s^t$ to denote the server model update.

If server updates the global model following the AFLGuard algorithm, i.e., Algorithm 2, then for any $t > 0, \tau_i > 0$, we have:

$$\begin{aligned}
\left\|\boldsymbol{\theta}^{t+1} - \boldsymbol{\theta}^*\right\| &= \left\|\boldsymbol{\theta}^t - \eta\boldsymbol{g}_i^{t-\tau_i} - \boldsymbol{\theta}^*\right\| \\
&\le \left\|\boldsymbol{\theta}^t - \eta\nabla F(\boldsymbol{\theta}^t) - \boldsymbol{\theta}^*\right\| + \eta\left\|\boldsymbol{g}_i^{t-\tau_i} - \nabla F(\boldsymbol{\theta}^t)\right\| \\
&\overset{(a)}{\le} \underbrace{\left\|\boldsymbol{\theta}^t - \eta\nabla F(\boldsymbol{\theta}^t) - \boldsymbol{\theta}^*\right\|}_{\clubsuit} + \eta\lambda\underbrace{\left\|\nabla F(\boldsymbol{\theta}^t) - \nabla F(\boldsymbol{\theta}^*)\right\|}_{\bigstar} \\
&\quad + \eta(\lambda+1)\underbrace{\left\|\boldsymbol{g}_s^t - \nabla F(\boldsymbol{\theta}^t)\right\|}_{\blacklozenge} \\
&\overset{(b)}{\le} \left(\sqrt{1 - \frac{2\eta\mu L}{\mu + L}} + \eta L\lambda + 8\eta\Lambda(\lambda+1)\right)\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\| \\
&\quad + 4\eta\Gamma(\lambda+1),
\end{aligned} \tag{9}$$

where $(a)$ uses $\nabla F(\boldsymbol{\theta}^*) = 0$ and Lemma 1, $(b)$ is true by plugging in Lemma 2, Assumption 1 and Lemma 3 into $\clubsuit$, $\bigstar$, and $\blacklozenge$, respectively. Telescoping, one has $\left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\| \le (1-q)^t\left\|\boldsymbol{\theta}^0 - \boldsymbol{\theta}^*\right\| + 4\eta\Gamma(\lambda+1)/q$, where $q = 1 - \left(\sqrt{1 - 2\eta\mu L/(\mu+L)} + \eta L\lambda + 8\eta\Lambda(\lambda+1)\right)$.

Next, we proof Lemma 1, Lemma 2 and Lemma 3 one by one.

Lemma 1. *If the server uses the AFLGuard algorithm to update the global model, then for arbitrary number of malicious clients, one has:*

$$\left\|\boldsymbol{g}_i^{t-\tau_i} - \nabla F(\boldsymbol{\theta}^t)\right\| \le (\lambda+1)\left\|\boldsymbol{g}_s^t - \nabla F(\boldsymbol{\theta}^t)\right\| + \lambda\left\|\nabla F(\boldsymbol{\theta}^t)\right\|.$$

Proof.

$$\begin{aligned}
&\left\|\boldsymbol{g}_i^{t-\tau_i} - \nabla F(\boldsymbol{\theta}^t)\right\| \\
&\le \left\|\boldsymbol{g}_i^{t-\tau_i} - \boldsymbol{g}_s^t\right\| + \left\|\boldsymbol{g}_s^t - \nabla F(\boldsymbol{\theta}^t)\right\|
\end{aligned}$$

$$\overset{(a)}{\leq} \lambda \left\|\boldsymbol{g}_s^t\right\| + \left\|\boldsymbol{g}_s^t - \nabla F(\boldsymbol{\theta}^t)\right\|$$
$$\leq \lambda \left\|\boldsymbol{g}_s^t - \nabla F(\boldsymbol{\theta}^t)\right\| + \lambda \left\|\nabla F(\boldsymbol{\theta}^t)\right\| + \left\|\boldsymbol{g}_s^t - \nabla F(\boldsymbol{\theta}^t)\right\|$$
$$= (\lambda+1)\left\|\boldsymbol{g}_s^t - \nabla F(\boldsymbol{\theta}^t)\right\| + \lambda \left\|\nabla F(\boldsymbol{\theta}^t)\right\|, \tag{10}$$

where $(a)$ is because for the AFLGuard algorithm, we have that $\left\|\boldsymbol{g}_i^{t-\tau_i} - \boldsymbol{g}_s^t\right\| \leq \lambda \left\|\boldsymbol{g}_s^t\right\|$. □

Lemma 2. *Suppose Assumption 1 holds, if the global learning rate satisfies $\eta \leq \frac{2}{\mu+L}$, then we have the following:*

$$\left\|\boldsymbol{\theta}^t - \eta \nabla F(\boldsymbol{\theta}^t) - \boldsymbol{\theta}^*\right\| \leq \sqrt{1 - 2\eta\mu L/(\mu+L)} \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|.$$

Proof. $\left\|\boldsymbol{\theta}^t - \eta \nabla F(\boldsymbol{\theta}^t) - \boldsymbol{\theta}^*\right\|^2 = \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|^2 + \eta^2 \left\|\nabla F(\boldsymbol{\theta}^t)\right\|^2 - 2\eta \left\langle \boldsymbol{\theta}^t - \boldsymbol{\theta}^*, F(\boldsymbol{\theta}^t)\right\rangle$. According to [8], if $F(\boldsymbol{\theta})$ is $L$-smooth and $\mu$-strongly convex, for any $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$, one has $\frac{\mu L}{\mu+L} \left\|\boldsymbol{\theta} - \boldsymbol{\theta}'\right\|^2 + \frac{1}{\mu+L} \left\|\nabla F(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta}')\right\|^2 \leq \left\langle \nabla F(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta}'), \boldsymbol{\theta} - \boldsymbol{\theta}'\right\rangle$. Setting $\boldsymbol{\theta} = \boldsymbol{\theta}^t, \boldsymbol{\theta}' = \boldsymbol{\theta}^*$, since $\nabla F(\boldsymbol{\theta}^*) = 0$, we have that $\frac{\mu L}{\mu+L} \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|^2 + \frac{1}{\mu+L} \left\|\nabla F(\boldsymbol{\theta}^t)\right\|^2 \leq \left\langle \nabla F(\boldsymbol{\theta}^t), \boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\rangle$. Thus one has:

$$\left\|\boldsymbol{\theta}^t - \eta \nabla F(\boldsymbol{\theta}^t) - \boldsymbol{\theta}^*\right\|^2 \leq \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|^2 + \eta^2 \left\|\nabla F(\boldsymbol{\theta}^t)\right\|^2$$
$$- 2\eta \left( \frac{\mu L}{\mu+L} \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|^2 + \frac{1}{\mu+L} \left\|\nabla F(\boldsymbol{\theta}^t)\right\|^2 \right)$$
$$= (1 - 2\eta\mu L/(\mu+L)) \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|^2 + \eta\left(\eta - 2/(\mu+L)\right) \left\|\nabla F(\boldsymbol{\theta}^t)\right\|^2$$
$$\overset{(a)}{\leq} (1 - 2\eta\mu L/(\mu+L)) \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|^2, \tag{11}$$

where $(a)$ is because $0 < \eta \leq 2/(\mu+L)$. $\left\|\boldsymbol{\theta}^t - \eta \nabla F(\boldsymbol{\theta}^t) - \boldsymbol{\theta}^*\right\| \leq \sqrt{1 - 2\eta\mu L/(\mu+L)} \left\|\boldsymbol{\theta}^t - \boldsymbol{\theta}^*\right\|$. □

The proof of Lemma 3 is mainly motivated from [9, 16]. To simplify the notation, we will ignore the superscript $t$ in $\boldsymbol{g}_s^t$. Define $\nabla \bar{f}_s(\boldsymbol{\theta}) = \frac{1}{|X_s|} \sum_{x \in X_s} \nabla f(\boldsymbol{\theta}, x)$.

Lemma 3. *If Assumptions 2-4 hold and $\Theta \subset \{\boldsymbol{\theta} : \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \leq \epsilon\sqrt{d}\}$ holds for some parameter $\epsilon > 0$. For any $\beta \in (0,1)$, if $\Gamma \leq \alpha_1^2/\rho_1$ and $\Lambda \leq \alpha_2^2/\rho_2$, we have that:*

$$\mathbb{P}\left\{\|\boldsymbol{g}_s - \nabla F(\boldsymbol{\theta})\| \leq 8\Lambda \left\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\right\| + 4\Gamma\right\} \geq 1 - \beta,$$

*where $\Gamma, \Lambda$ are defined in Theorem 1.*

Proof. We define $\xi = \frac{\rho_2 \alpha_1}{2\alpha_2^2} \sqrt{\frac{d}{|X_s|}}$ and let $\psi = \left\lceil \epsilon\sqrt{d}/\xi \right\rceil$. Then for any integer $1 \leq l \leq \psi$, we define $\Theta_l = \{\boldsymbol{\theta} : \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \leq \xi l\}$. For a given integer $l$, we let $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_{\hat{\varsigma}}$ be an $\omega$-cover of $\Theta_l$, where $\omega = \frac{\alpha_2 \xi l}{R} \sqrt{d/|X_s|}$, where $R = \max\{L, H\}$. From [47], we know that $\log \hat{\varsigma} \leq d \log(3\xi l/\omega)$. For any $\boldsymbol{\theta} \in \Theta_l$, there exists a $1 \leq c \leq \omega$ such that $\|\boldsymbol{\theta} - \boldsymbol{\theta}_c\| \leq \omega$ holds. Then, based on the triangle inequality, one has $\left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta})\right\| \leq \|\nabla F(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta}_c)\| + \left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla \bar{f}_s(\boldsymbol{\theta}_c)\right\| + \left\|\nabla \bar{f}_s(\boldsymbol{\theta}_c) - \nabla F(\boldsymbol{\theta}_c)\right\|$. By Assumption 1, one has $\|\nabla F(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta}_c)\| \leq L \|\boldsymbol{\theta} - \boldsymbol{\theta}_c\| \leq L\omega$. We define event $E_1$ as:

$$E_1 = \left\{\sup_{\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta : \boldsymbol{\theta} \neq \boldsymbol{\theta}'} \left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla \bar{f}_s(\boldsymbol{\theta}')\right\| \leq H \left\|\boldsymbol{\theta} - \boldsymbol{\theta}'\right\|\right\}. \tag{12}$$

According to Assumption 4, we have $\mathbb{P}\{E_1\} \geq 1 - \beta/3$. One also has $\sup_{\boldsymbol{\theta} \in \Theta} \left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla \bar{f}_s(\boldsymbol{\theta}_c)\right\| \leq H\omega$. By the triangle inequality again, and because $\mathbb{E}[q(\boldsymbol{\theta}, X)] = \nabla F(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta}^*)$, we have:

$$\left\|\nabla \bar{f}_s(\boldsymbol{\theta}_c) - \nabla F(\boldsymbol{\theta}_c)\right\| \leq \left\|\nabla \bar{f}_s(\boldsymbol{\theta}^*) - \nabla F(\boldsymbol{\theta}^*)\right\|$$
$$+ \left\|\nabla \bar{f}_s(\boldsymbol{\theta}_c) - \nabla \bar{f}_s(\boldsymbol{\theta}^*) - \left(\nabla F(\boldsymbol{\theta}_c) - \nabla F(\boldsymbol{\theta}^*)\right)\right\|$$
$$\leq \left\|\nabla \bar{f}_s(\boldsymbol{\theta}^*) - \nabla F(\boldsymbol{\theta}^*)\right\| + \left\| \frac{1}{|X_s|} \sum\nolimits_{x \in X_s} q(\boldsymbol{\theta}_c, x) - \mathbb{E}[q(\boldsymbol{\theta}_c, X)] \right\|.$$

Define events $E_2 = \left\{\left\|\nabla \bar{f}_s(\boldsymbol{\theta}^*) - \nabla F(\boldsymbol{\theta}^*)\right\| \leq 2\Gamma\right\}$ and $E_l$ as:

$$E_l = \left\{ \sup_{1 \leq k \leq N} \left\| \frac{1}{|X_s|} \sum\nolimits_{x \in X_s} q(\boldsymbol{\theta}_k, x) - \mathbb{E}[q(\boldsymbol{\theta}_k, X)] \right\| \leq 2\Lambda\xi l \right\}.$$

By Proposition 1, Proposition 2, since $\Gamma \leq \alpha_1^2/\rho_1$, $\Lambda \leq \alpha_2^2/\rho_2$, we have $\mathbb{P}\{E_2\} \geq 1 - \beta/3$, $\mathbb{P}\{E_l\} \geq 1 - \beta/(3\psi)$. Thus, on event $E_1 \cap E_2 \cap E_l$, one has $\sup_{\boldsymbol{\theta} \in \Theta_l} \left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta})\right\| \leq L\omega + H\omega + 2\Gamma + 2\Lambda\xi l \leq 4\Lambda\xi l + 2\Gamma$. Thus, we have at least $1 - \beta$ that event $E = E_1 \cap E_2 \cap (\cap_{l=1}^{\psi} E_l)$. Also, on event $E$, for any $\boldsymbol{\theta} \in \Theta_\psi$, there exists an $1 \leq l \leq \psi$ such that $(l-1)\xi < \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| \leq \xi l$ holds. If $l = 1$, we have $\left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta})\right\| \leq 4\Lambda\xi + 2\Gamma \leq 4\Gamma$; and $2(l-1) \geq l$ if $l \geq 2$. Thus, one has $\left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta})\right\| \leq 8\Lambda \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| + 2\Gamma$. On $E$, one has $\sup_{\boldsymbol{\theta} \in \Theta_\psi} \left\|\nabla \bar{f}_s(\boldsymbol{\theta}) - \nabla F(\boldsymbol{\theta})\right\| \leq 8\Lambda \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| + 4\Gamma$. □

The following proof of Proposition 1 is mainly motivated from [9, 16].

Proposition 1. *Suppose Assumption 2 holds. For any $\beta \in (0,1)$, $\boldsymbol{\theta} \in \Theta$, let $\Gamma = \sqrt{2}\alpha_1 \sqrt{(d \log 6 + \log(3/\beta))/|X_s|}$. If $\Gamma \leq \alpha_1^2/\rho_1$, then we have:*

$$\mathbb{P}\left\{ \left\| \frac{1}{|X_s|} \sum\nolimits_{x \in X} \nabla f(\boldsymbol{\theta}^*, x) - \nabla F(\boldsymbol{\theta}^*) \right\| \geq 2\Gamma \right\} \leq \beta/3.$$

Proof. We let $\mathbf{B} = \{\mathbf{v}_1, \cdots, \mathbf{v}_\varsigma\}$ be one $\frac{1}{2}$-cover of the unit sphere $\mathbf{V}$. By [47], one has $\log \varsigma \leq d \log 6$ and $\left\|\nabla \bar{f}_s(\boldsymbol{\theta}^*) - \nabla F(\boldsymbol{\theta}^*)\right\| \leq 2 \sup_{\mathbf{v} \in \mathbf{B}} \left\{\left\langle \nabla \bar{f}_s(\boldsymbol{\theta}^*) - \nabla F(\boldsymbol{\theta}^*), \mathbf{v}\right\rangle\right\}$. If Assumption 2 and the condition $\Gamma \leq \alpha_1^2/\rho_1$ satisfy, and according to the concentration inequalities for sub-exponential random variables [48], we have that $\mathbb{P}\left\{\left\langle \nabla \bar{f}_s(\boldsymbol{\theta}^*) - \nabla F(\boldsymbol{\theta}^*), \mathbf{v}\right\rangle \geq \Gamma\right\} \leq \exp\left(-|X_s|\Gamma^2/(2\alpha_1^2)\right)$. One has $\mathbb{P}\left\{\left\|\nabla \bar{f}_s(\boldsymbol{\theta}^*) - \nabla F(\boldsymbol{\theta}^*)\right\| \geq 2\Gamma\right\} \leq \exp\left(-|X_s|\Gamma^2/(2\alpha_1^2) + d \log 6\right)$ by the union bound. Put in $\Gamma$ finishes the proof. □

Proposition 2. *Suppose Assumption 3 holds. For any $\beta \in (0,1)$, $\boldsymbol{\theta} \in \Theta$, let $\Delta = \sqrt{2}\alpha_2 \sqrt{(d \log 6 + \log(3/\beta))/|X_s|}$. If $\Delta \leq \alpha_2^2/\rho_2$, then we have:*

$$\mathbb{P}\left\{ \left\| \frac{1}{|X_s|} \sum_{x \in X_s} \nabla q(\boldsymbol{\theta}, x) - \mathbb{E}[q(\boldsymbol{\theta}, X)] \right\| \geq 2\Delta \left\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\right\| \right\} \leq \beta/3.$$

Proof. The proof of Proposition 2 is similar to that of Proposition 1, and is omitted here for brevity. □

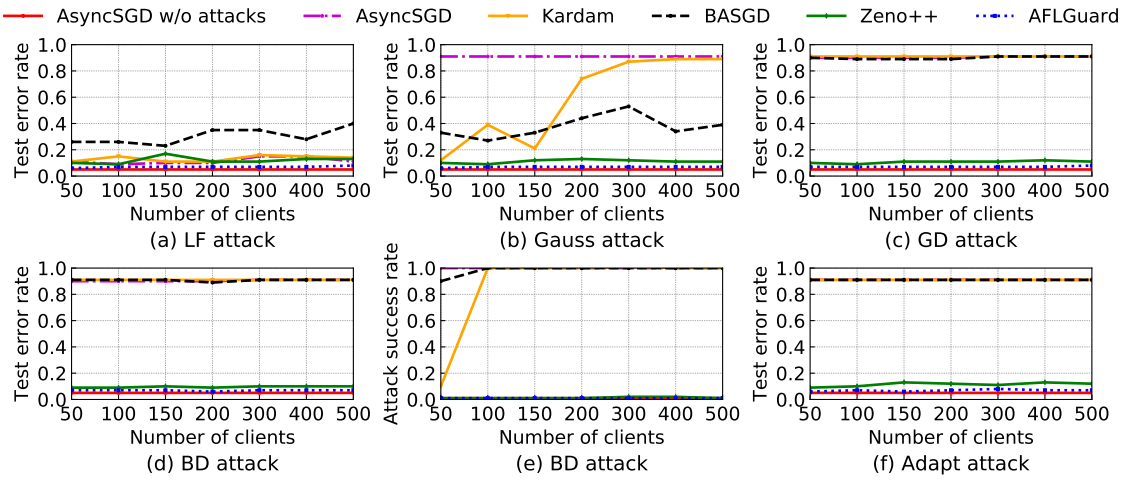



**Figure 8: Test error rates and attack success rates of different defenses under different attacks with different number of clients on MNIST dataset.**

**Table 4: The CNN architecture.**

| Layer | Size |
|---|---|
| Input | $28 \times 28 \times 1$ |
| Convolution + ReLU | $3 \times 3 \times 30$ |
| Max Pooling | $2 \times 2$ |
| Convolution + ReLU | $3 \times 3 \times 50$ |
| Max Pooling | $2 \times 2$ |
| Fully Connected + ReLU | 100 |
| Softmax | 10 |

## A.3 Datasets

**1) Synthetic Dataset:** We randomly generate 10,000 data samples of dimensions $d = 100$. Each dimension follows the Gaussian distribution $N(0, 1)$ and noise $e_i$ is sampled from $N(0, 1)$. We use $N(0, 25)$ to generate each entry of $\boldsymbol{\theta}^*$. We generate $y_i$ according to the linear regression model in Lemma 5.1. We randomly draw 8,000 samples for training and use the remaining 2,000 samples for testing.

**2) MNIST [30]:** MNIST is a 10-class handwritten digits image classification dataset, which contains 60,000 samples for training and 10,000 examples for testing.

**3) Fashion-MNIST [50]:** Fashion-MNIST is a dataset containing images of 70,000 fashion products from 10 classes. The training set has 60,000 images and the testing set has 10,000 images.

**4) Human Activity Recognition (HAR) [4]:** The HAR dataset aims to recognize 6 types of human activities and the dataset is collected from smartphones of 30 real-world users. There are 10,299 examples in total and each example includes 561 features. We randomly sample 75% of each client's examples as training data and use the rest as test data.

**5) Colorectal Histology MNIST [28]:** Colorectal Histology MNIST is an 8-class dataset for classification of textures in human colorectal cancer histology. This dataset contains 5,000 images and each image has 64×64 grayscale pixels. We randomly select 4,000 images for training and use the remaining 1,000 images for testing.

**6) CIFAR-10 [29]:** CIFAR-10 consists of 60,000 color images. This dataset has 10 classes, and there are 6,000 images of each class. The training set has 50,000 images and the testing set has 10,000 images.

**Table 5: Test error rates and attack success rates of Zeno++ and AFLGuard under different attacks with different distribution shifts (DSs) on Fashion-MNIST, HAR, Colorectal Histology MNIST and CIFAR-10 datasets. The results of BD attack are in the form of "test error rate / attack success rate".**

**(a) Fashion-MNIST**

| DS | 0.1 | | 0.5 | | 0.6 | | 0.8 | | 1.0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Attack | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard |
| No attack | 0.25 | 0.16 | 0.26 | 0.17 | 0.31 | 0.18 | 0.58 | 0.21 | 0.90 | 0.22 |
| LF attack | 0.25 | 0.18 | 0.29 | 0.21 | 0.32 | 0.21 | 0.72 | 0.22 | 0.90 | 0.22 |
| Gauss attack | 0.26 | 0.18 | 0.28 | 0.19 | 0.34 | 0.19 | 0.58 | 0.22 | 0.90 | 0.23 |
| GD attack | 0.26 | 0.18 | 0.29 | 0.21 | 0.35 | 0.21 | 0.58 | 0.21 | 0.90 | 0.23 |
| BD attack | 0.26 / 0.04 | 0.17 / 0.04 | 0.29 / 0.05 | 0.20 / 0.04 | 0.33 / 0.04 | 0.20 / 0.04 | 0.58 / 0.03 | 0.20 / 0.03 | 0.90 / 0.01 | 0.22 / 0.02 |
| Adapt attack | 0.26 | 0.19 | 0.29 | 0.21 | 0.36 | 0.21 | 0.72 | 0.22 | 0.90 | 0.25 |

**(b) HAR**

| DS | 0.167 | | 0.5 | | 0.6 | | 0.8 | | 1.0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Attack | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard |
| No attack | 0.06 | 0.05 | 0.06 | 0.05 | 0.08 | 0.05 | 0.10 | 0.07 | 0.43 | 0.12 |
| LF attack | 0.07 | 0.05 | 0.08 | 0.05 | 0.09 | 0.05 | 0.10 | 0.09 | 0.43 | 0.36 |
| Gauss attack | 0.07 | 0.05 | 0.07 | 0.05 | 0.09 | 0.05 | 0.10 | 0.07 | 0.43 | 0.36 |
| GD attack | 0.07 | 0.05 | 0.08 | 0.05 | 0.09 | 0.06 | 0.12 | 0.09 | 0.43 | 0.42 |
| BD attack | 0.06 / 0.07 | 0.05 / 0.01 | 0.07 / 0.01 | 0.05 / 0.01 | 0.09 / 0.02 | 0.05 / 0.01 | 0.10 / 0.01 | 0.07 / 0.01 | 0.43 / 0.01 | 0.36 / 0.01 |
| Adapt attack | 0.07 | 0.05 | 0.08 | 0.05 | 0.09 | 0.06 | 0.14 | 0.09 | 0.55 | 0.54 |

**(c) Colorectal Histology MNIST**

| DS | 0.125 | | 0.5 | | 0.6 | | 0.8 | | 1.0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Attack | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard |
| No attack | 0.25 | 0.18 | 0.31 | 0.22 | 0.49 | 0.29 | 0.62 | 0.32 | 0.71 | 0.34 |
| LF attack | 0.31 | 0.21 | 0.39 | 0.23 | 0.53 | 0.37 | 0.63 | 0.41 | 0.88 | 0.41 |
| Gauss attack | 0.35 | 0.21 | 0.43 | 0.22 | 0.59 | 0.32 | 0.70 | 0.35 | 0.86 | 0.35 |
| GD attack | 0.29 | 0.21 | 0.39 | 0.32 | 0.66 | 0.36 | 0.74 | 0.49 | 0.88 | 0.58 |
| BD attack | 0.42 / 0.02 | 0.22 / 0.02 | 0.44 / 0.02 | 0.27 / 0.02 | 0.57 / 0.01 | 0.31 / 0.02 | 0.62 / 0.01 | 0.32 / 0.01 | 0.82 / 0.24 | 0.51 / 0.03 |
| Adapt attack | 0.44 | 0.29 | 0.64 | 0.33 | 0.72 | 0.43 | 0.77 | 0.51 | 0.88 | 0.62 |

**(d) CIFAR-10**

| DS | 0.1 | | 0.5 | | 0.6 | | 0.8 | | 1.0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Attack | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard | Zeno++ | AFLGuard |
| No attack | 0.32 | 0.24 | 0.41 | 0.26 | 0.54 | 0.29 | 0.68 | 0.31 | 0.90 | 0.31 |
| LF attack | 0.33 | 0.34 | 0.53 | 0.34 | 0.64 | 0.35 | 0.71 | 0.35 | 0.90 | 0.38 |
| Gauss attack | 0.33 | 0.32 | 0.52 | 0.33 | 0.72 | 0.33 | 0.76 | 0.35 | 0.90 | 0.35 |
| GD attack | 0.32 | 0.27 | 0.60 | 0.30 | 0.83 | 0.31 | 0.85 | 0.33 | 0.90 | 0.32 |
| BD attack | 0.32 / 0.95 | 0.28 / 0.01 | 0.49 / 0.06 | 0.29 / 0.01 | 0.62 / 0.00 | 0.32 / 0.02 | 0.80 / 0.01 | 0.34 / 0.01 | 0.90 / 0.00 | 0.36 / 0.04 |
| Adapt attack | 0.77 | 0.32 | 0.82 | 0.36 | 0.90 | 0.36 | 0.90 | 0.36 | 0.90 | 0.39 |