# FedSecurity: A Benchmark for Attacks and Defenses in Federated Learning and Federated LLMs

Shanshan Han
University of California, Irvine
Irvine, USA
shanshan.han@uci.edu

Baturalp Buyukates
University of Southern California
Los Angeles, USA
buyukate@usc.edu

Zijian Hu
TensorOpera Inc.
Palo Alto, USA
zjh@tensoropera.com

Han Jin
University of Southern California
Los Angeles, USA
hanjin@usc.edu

Weizhao Jin
University of Southern California
Los Angeles, USA
weizhaoj@usc.edu

Lichao Sun
Lehigh University
Bethlehem, USA
lis221@lehigh.edu

Xiaoyang Wang
UIUC
Urbana, USA
xw28@illinois.edu

Wenxuan Wu
Texas A&M University
College Station, USA
ww6726@tamu.edu

Chulin Xie
UIUC
Urbana, USA
chulinx2@illinois.edu

Yuhang Yao
Carnegie Mellon University
Pittsburgh, USA
yuhangya@andrew.cmu.edu

Kai Zhang
Lehigh University
Bethlehem, USA
kaz321@lehigh.edu

Qifan Zhang
University of California, Irvine
Irvine, USA
qifan.zhang@uci.edu

Yuhui Zhang
Zhejiang University
Hangzhou, China
zhangyuhui42@zju.edu.cn

Carlee Joe-Wong
Carnegie Mellon University
Pittsburgh, USA
cjoewong@andrew.cmu.edu

Salman Avestimehr[*]
USC
Los Angeles, USA
avestime@usc.edu

Chaoyang He
TensorOpera Inc.
Palo Alto, USA
ch@tensoropera.com

## ABSTRACT

This paper introduces FedSecurity, an end-to-end benchmark that serves as a supplementary component of the FedML library for simulating adversarial attacks and corresponding defense mechanisms in Federated Learning (FL). FedSecurity eliminates the need for implementing the fundamental FL procedures, *e.g.*, FL training and data loading, from scratch, thus enables users to focus on developing their own attack and defense strategies. It contains two key components, including FedAttacker that conducts a variety of attacks during FL training, and FedDefender that implements defensive mechanisms to counteract these attacks. FedSecurity has the following features: *i)* It offers extensive customization options to accommodate a broad range of machine learning models (*e.g.*, Logistic Regression, ResNet, and GAN) and FL optimizers (*e.g.*, FedAVG, FedOPT, and FedNOVA); *ii)* it enables exploring the effectiveness of attacks and defenses across different datasets and models; and *iii)* it supports flexible configuration and customization through a configuration file and some APIs. We further demonstrate FedSecurity's utility and adaptability through federated training of Large

Language Models (LLMs) to showcase its potential on a wide range of complex applications.

## 1 INTRODUCTION

Federated Learning (FL) [67] facilitates training across distributed data and enable clients to utilize their local data to train machine learning models collaboratively. Instead of collecting data to a centralized server, FL clients train models on their local data and share the local models with the FL server, where the local models are aggregated into a global model. FL has attracted considerable attention across various domains and has been utilized in numerous areas such as next-word prediction [15, 35, 76], hot-word detection [55], financial risk assessment [11], and cancer risk prediction [17], etc.

Recently, FL applications are expanded with large language models (LLMs), *i.e.*, *federated LLMs* [13]. Currently, there are industry products that utilize FL (or distributed training) to train LLMs, such as Deepspeed ZeRO [74, 91], HuggingFace Accelerate [33], Pytorch Lightning Fabric [2]. FL facilitates LLM training due to the following reasons: *i) Distributed nature of LLM training data:* LLMs are pre-trained using large amounts of data, which often reside in different locations. Collecting such data to a central server is

**Table 1: Types of Defenses and Their Implementations Against Specific Attacks**

| Type of Defenses | Implementations | Attacks Against (Selected) |
|---|---|---|
| Before-aggregation defenses | SLSGD [98] | Data poisoning attacks, *e.g.*, [89], |
| | Residual Reweighting Defense [26] | Backdoor attacks, *e.g.*, [31, 89] |
| | Foolsgold [27] | Backdoor attacks, *e.g.* [89] |
| | Krum and *m*-Krum [9]   Bulyan [32] Norm Clipping [87]   Fl-wbc [85] coordinate-wise median [102]   CClip [46] coordinate-wise trimmed mean [102] anomaly detection [34]   weak DP [87] | Model poisoning attacks, *e.g.*, Byzantine attacks [16, 22, 60] or Backdoor attacks that attack by poisoning model updates [3] |
| On-aggregation defenses | Robust Learning Rate [72] | Backdoor attacks, *e.g.*, [89] |
| | SLSGD [98] | Data poisoning attacks, *e.g.*, [89], |
| | Geometric median [16], RFA [73] | Byzantine attacks [16, 22, 60], data poisoning attacks [89] |
| After-aggregation defenses | CClip [46] | Byzantine attacks or backdoor attacks |
| | CRFL [96] | Backdoor attacks |

**Table 2: Attacks Implemented in FedSecurity**

| Type of Attacks | Implementations |
|---|---|
| Model poisoning | Byzantine attack [16, 22, 60]: zero mode, random mode, and flipping mode Minimizing Distance Backdoor [4] Model Replacement Backdoor [3] Lazy Worker (or Free Rider) [25, 93] |
| Data poisoning | Label Flipping Backdoor attack [89] Edge Case Backdoor Attack [92] |
| Data reconstruction | Deep Leakage Attack [106] Inverting Gradient Attack [29] Revealing Labels Attack [18] |

expensive and may also leak sensitive user information, while a viable way is to train LLMs in a federated manner. *ii) Scalability and efficiency:* LLMs, such as GPT-3 [10], have an extremely large number of parameters. Training LLMs on a single machine is infeasible and inflexible, while FL can be a good choice. *iii) Continuous improvement with user data:* LLMs can be deployed in a federated manner and local instances of the models can be further finetuned based on the local data, which enables the global model to improve over time based on users' data without ever having direct access to the data. This is particularly relevant for privacy-sensitive fields such as healthcare or personal communications.

FL, as well as federated LLMs, aim to maintain privacy and security of client data by allowing clients to train locally without sharing their data to other parties. However, its decentralized and collaborative nature inadvertently introduce privacy and security vulnerabilities. Adversarial clients compromise the integrity of the global model by submitting spurious models to prevent the global model from converging [3, 4, 16, 22, 25, 60, 93] and/or planting backdoors to induce the global model to mis-classify specific samples [3, 4, 89, 92]. Adversaries can also reconstruct training data

from model updates or gradients [18, 29, 42, 43, 106]. Meanwhile, a wide range of defense mechanisms has emerged to mitigate the impact of these attacks [9, 14, 16, 27, 46, 49, 50, 59, 65, 66, 72, 73, 86, 87, 96, 98, 102, 104]. Despite the efforts for addressing the vulnerability of FL systems, there still lacks a comprehensive benchmark for comparing approaches under unified sittings. Moreover, while existing works have explored effectiveness of attacks and defenses on small-scale models, there remains a significant gap in understanding how these mechanisms perform against LLMs. Given that LLMs possess a large number of parameters and are trained on complex datasets obtained from unregulated sources, the effectiveness of attacks and defenses may be diminished when applied to LLMs. These motivate an urgent need for a standardized and comprehensive benchmark to evaluate baseline attack and defense strategies in the context of FL and federated LLMs.

This paper introduces FedSecurity[1], a benchmark that simulates attacks and defenses in FedML [37]. FedSecurity comprises two primary components: FedAttacker and FedDefender. FedAttacker simulates attacks in FL to help understand and prepare for potential security risks, while FedDefender is equipped with the state-of-the-art defense mechanisms to counteract the attacks injected by FedAttacker. We summarize our contributions as follows:

*i) Enabling benchmarking of several different attacks and defenses in FL.* FedSecurity implements attacks and defenses that are widely considered in the literature. We summarize the defenses and the attacks in Table 1 and Table 2, respectively.

*ii) Supporting flexible configuration and customization.* FedSecurity supports configurations using a .yaml file. Sample configurations for attacks and defenses are shown in Figures 1a and Figures 1b, respectively. FedSecurity also provides APIs to enable customizing attacks and defenses.

*iii) Supporting various models and FL optimizers.* FedSecurity can be utilized with a wide range of models, including Logistic Regression, LeNet [54], ResNet [39], CNN [53], RNN [81],

---

[1]Code: https://github.com/FedML-AI/FedML/tree/master/python/fedml/core/security

```
attack_args:
  enable_attack: true
  attack_type: byzantine
  attack_mode: random
  byzantine_client_num: 1
```

```
defense_args:
  enable_defense: true
  defense_type: krum
  krum_param_m: 5
  byzantine_client_num: 1
```

(a) Byzantine attack [16, 22].

(b) *m*-Krum [9].

Figure 1: Examples of attack and defense configurations.

GAN [30], etc. FedSecurity is compatible with various FL optimizers, such as FedAVG [69], FedSGD [83], FedOPT [78], FedPROX [58], FedGKT [36], FedGAN [77], FedNAS [38], FedNOVA [94], etc.

*iv)* **Extensions to federated LLMs and real-world applications.** FedSecurity can simulate attacks and defenses during training of federated LLMs. It can also be integrated with real-world FL applications; see **Exp 7**, where we utilize edge devices from Theta Network [88] as clients instead of simulating on a single machine. *Key takeaways*: *i)* While defense mechanisms can help mitigate attacks, it might also bring a loss of accuracy to the aggregation results. Therefore, when integrating defenses into FL applications, it's crucial to weigh the benefits against potential drawbacks. *ii)* Nearly all existing defense mechanisms are impractical in real-world FL applications, as they compromise accuracy even if no attack happened. However, attacks happen infrequently in practice. A defense strategy that is practical for real-world systems is in need, where the defense should satisfy: 1) it must detect if attacks have happened and only activate the defense mechanism when attacks are detected; and 2) it must identify malicious clients accurately without harming benign local models. *iii)* Based on our findings, we have developed a novel defense strategy; see [34] for details.

## 2 PRELIMINARIES AND OVERVIEW

This section discusses existing FL security frameworks, then introduces adversarial models, and finally overviews FedSecurity.

### 2.1 Existing Benchmark Frameworks

Recent years, researchers have proposed multiple benchmarks for FL [1, 6, 20, 51, 57, 61, 62, 79, 80, 84, 99, 107]. Among these, only Blades [57] and FederatedScope [99] study the implications of adversarial attacks in FL. Blades implements a wide range of attacks, such as [4, 22, 56, 82, 97], as well as corresponding defenses against those attacks, *e.g.*, [9, 47, 100, 100]. While Blades focuses more on model poisoning attacks and data poisoning attacks, it fails to include an important line of work, *i.e.*, data reconstruction attacks. FederatedScope [99] implements data reconstruction attacks that utilize models or gradients to revert sensitive information, including GAN-based leakage attack [41], Passive Property Inference [71], and DLG attack [106]. However, it neglects to address attacks prevalent in the research literature, *e.g.*, Byzantine attacks [101, 102]. It also does not include any defense mechanisms for FL. It is worth noting that, while FederatedScope integrates secret-sharing [5] for enhancing data privacy, it is in the scope of federated analytics [21, 45, 75, 90], instead of FL.

FedSecurity implements attacks that are widely considered in the literature and covers attacks that are injected at different stages of FL, including model poisoning attacks, data poisoning attacks,

and data reconstruction attacks [3, 4, 16, 18, 22, 25, 29, 60, 89, 92, 93, 106]. It also integrates a wide range of defense mechanisms to protect against the attacks [9, 16, 27, 34, 46, 72, 73, 87, 87, 96, 98, 102, 102]. Moreover, FedSecurity offers flexible configurations and APIs, which enables users to customize their attack and defense strategies efficiently.

### 2.2 Adversarial Model

Adversaries in FL have two categories, including active adversaries and passive adversaries, corresponding to security risks and privacy threat in FL, respectively.

**Active Adversaries.** Active adversaries intentionally manipulate training data or trained models to achieve malicious goals. This might involve altering models to prevent global model from converging (*e.g.*, Byzantine attacks [16, 22]), or subtly misclassifying a specific set of samples to minimally impact the overall performance of the global model (*e.g.*, backdoor attacks [3, 92, 105]). Active adversaries can take different forms, including: 1) malicious clients who manipulate their local models [3, 16, 22, 105] or submit contrived models without actual training [93]; 2) a global "sybil" [27, 89] that has full access to the FL system and possesses complete knowledge of the entire system, including local models and global models, as well as clients' local datasets. This "sybil" may also modify clients' local datasets and their local models submitted to the server; and 3) external adversaries or hackers that monitor the communication channel between the clients and the server, thereby intercepting and altering local models during the transfer process.

**Passive Adversaries.** Passive adversaries do not modify data or models, but can still breach data privacy by inferring sensitive information, such as local training data, from model updates or gradients [106]. Examples of passive adversaries include: 1) an adversarial FL server that tries to guess local training data using submitted local model updates and/or global models; 2) adversarial FL clients that attempt to deduce other clients' training data using the global models provided by the server of each FL training round; and 3) external adversaries, *e.g.*, hackers, that access communication channels to obtain local and global models transferred between clients and the FL server.

Adversaries can inject attacks at different stages of FL. Active adversaries can conduct *model poisoning attacks* to manipulate local models or *data poisoning attacks* to tamper with local data, while passive adversaries pose privacy threats by exploiting model updates or gradients, *i.e.*, *data reconstruction attacks*.

### 2.3 Overview of FedSecurity

FedSecurity serves as an external component of FedML [37] and injects attacks and defenses at different stages of FL training, without altering the existing FL procedures. FedSecurity utilizes FedAttacker and FedDefender to initiate two instances to simulate attacks and defenses, respectively. Such two instances are initialized once and are accessed by other objects in the FL system, a design achieved using the singleton design pattern [28].

We summarize the injections of attacks and defenses to the FL framework in FedSecurity in Figure 2. We also provide detailed algorithms for injecting attacks and defenses to different stages of FL training, as shown in Algorithm 1 (for server aggregation) and
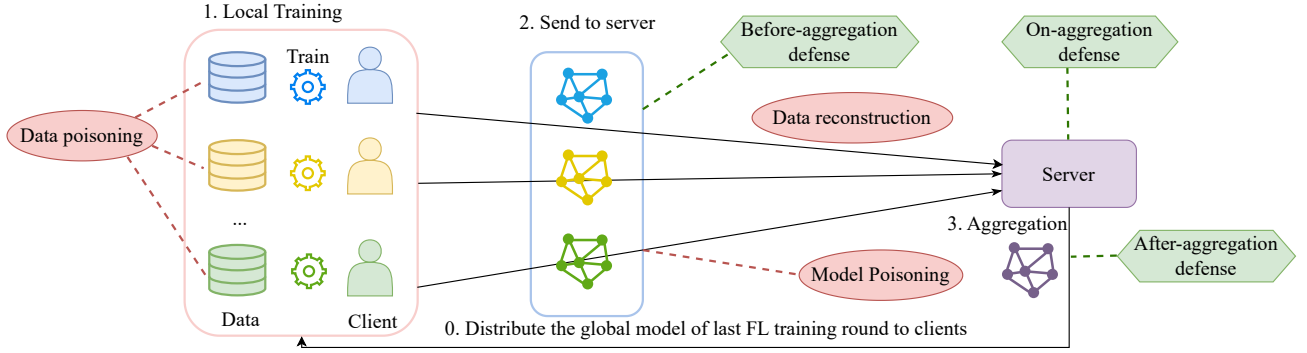
Figure 2: FedSecurity overview. FedSecurity enables injecting attacks/defenses (shown in red/green) at various stages of FL at the clients and at the server.

**Algorithm 1: Server Aggregation**

**Inputs:** $\mathbf{w}'_g$: the global model of last FL training round; $\mathcal{W}_l$: local models of the current FL round.
**Variables:** $\mathcal{A}$: A FedAttacker instance; $\mathcal{D}$: A FedDefender instance.
**Function** *server_aggregation*($\mathcal{W}_l, \mathbf{w}'_g$) **begin**

1    $\mathcal{W}_l \leftarrow$ *before_aggregation_process*($\mathcal{W}_l, \mathbf{w}'_g$)
2    $\mathbf{w}_g \leftarrow$ *on_aggregation_process*($\mathcal{W}_l, \mathbf{w}'_g$)
3    **return** *after_aggregation_process*($\mathbf{w}_g$)

**Function** *before_aggregation_process*($\mathcal{W}_l, \mathbf{w}'_g$) **begin**

4    **if** $\mathcal{A}$.*is_attack_enabled*() **then**
5      **if** $\mathcal{A}$.*is_data_reconstruction_attack*() **then**
       $\mathcal{A}$.*reconstruct_data*($\mathcal{W}_l, \mathbf{w}'_g$)
     **if** $\mathcal{A}$.*is_model_poisoning_attack*() **then**
       $\mathcal{W}_l \leftarrow \mathcal{A}$.*poison_model*($\mathcal{W}_l, \mathbf{w}'_g$)
6    **if** $\mathcal{D}$.*is_defense_enabled*() & $\mathcal{D}$.*is_defense_before_aggregation*() **then**
     $\mathcal{W}_l \leftarrow \mathcal{D}$.*defend_before_aggregation*($\mathcal{W}_l, \mathbf{w}'_g$)
7    **return** $\mathcal{W}_l$

**Function** *on_aggregation_process*($\mathcal{W}_l, \mathbf{w}_g$) **begin**

8    **if** $\mathcal{D}$.*is_defense_enabled*() & $\mathcal{D}$.*is_defense_on_aggregation*() **then**
     **return** $\mathcal{D}$.*defend_on_aggregation*($\mathcal{W}_l, \mathbf{w}_g$)
9    **return** *aggregate*($\mathcal{W}_l$)

**Function** *after_aggregation_process*($\mathbf{w}_g$) **begin**

10   **if** $\mathcal{D}$.*is_defense_enabled*() & $\mathcal{D}$.*is_defense_after_aggregation*() **then**
     **return** $\mathcal{D}$.*defend_after_aggregation*($\mathbf{w}_g$)
11   **return** $\mathbf{w}_g$

**Algorithm 2: Client Training**

**Inputs:** *dataset*: the local dataset of a client.
**Variables:** $\mathcal{A}$: A FedAttacker instance.
1 **Function** *client_training*(*dataset*) **begin**
2    **if** $\mathcal{A}$.*is_attack_enabled*() & $\mathcal{A}$.*is_data_poisoning_attack*() **then**
     *dataset* $\leftarrow \mathcal{A}$.*poison_data*(*dataset*);
3    $\mathbf{w}_l \leftarrow$ *train*(*dataset*)
4    *send_to_server*($\mathbf{w}_l$)

(1) *Data poisoning attacks* conducted by active adversaries to modify clients' local datasets and are injected at clients [18, 89].
(2) *Model poisoning attacks* that are also conducted by active adversaries to temper with local models submitted by clients that participate in the current FL iteration [7, 22, 82].
(3) *Data reconstruction attacks* that are conducted by passive adversaries by exploring local models, model updates, or gradients to infer information about the training data [24, 64, 70, 95, 103].

Without loss of generality, FedAttacker injects data poisoning attacks and model poisoning attacks before the aggregation of local models in each FL training round at the server, such that the FedAttacker instance can get access to all client models submitted in the FL training round. FedAttacker injects data reconstruction attacks at the FL server as well, where the FL server has access to all local models and the global model of each iteration and can perform the attacks with high flexibility.

*2.3.2 Injection of defenses.* FedDefender incorporates defenses to mitigate, if not completely nullify, the impacts of injected attacks. The defenses are designed to address issues related to tampered local models or manipulated local datasets, which lead to compromised model integrity, or information leakage during the exchange of model updates between clients and the FL server. The defenses manipulate local models and the aggregation procedure to counteract the attacks.

To facilitate this, FedDefender deploys defenses at the FL server, and provides flexible APIs that enable obtaining all local models and the global model of each FL round while allowing for a customized

Algorithm 2 (for client training). Below we introduce injections of attacks and defenses, respectively.

*2.3.1 Injection of attacks.* Without loss of generality, we classify the attacks into the following categories based on their targets:

Table 3: APIs for Different Types of Attacks in FedSecurity

| Type of Attacks | APIs | Explanations |
|---|---|---|
| Model Poisoning | $poison\_model(local\_models, auxiliary\_info)$ | Take the local models uploaded by clients in the current FL iteration and modify the local models. |
| | $is\_model\_poisoning\_attack()$ | Examine whether FedAttacker is activated and whether it modifies local models. |
| Data Poisoning | $poison\_data(dataset)$ | Take a local dataset and mislabel a set of chosen samples based on the configuration of FedAttacker. |
| | $is\_data\_poisoning\_attack()$ | Examine whether FedAttacker is enabled and whether the attack requires poisoning datasets. |
| Data Reconstruction | $reconstruct\_data(model, auxiliary\_info)$ | Take a client model or a global model, or a model update to reconstruct the training data. |
| | $is\_data\_reconstruction\_attack()$ | Examine whether FedAttacker is enabled and whether it reconstructs training data. |

The input *local_models* is a list of tuples that contain the number of samples and the local models submitted by clients in each FL iteration.
The input *auxiliary_info* can be any information that is utilized in the attack functions, *e.g.*, the global model in the last FL iteration.

aggregation process. FedDefender utilize three functions at different stages of FL aggregation:

(1) *Before-aggregation functions* that modify local models at the server before aggregating them.
(2) *On-aggregation functions* that modify the FL aggregation function to mitigate the impacts of malicious local models.
(3) *After-aggregation functions* that modify the aggregated global model (*e.g.*, by adding noise or clipping) to protect the real global model or improve its quality.

## 3 IMPLEMENTATION OF ATTACKS

FedAttacker injects model poisoning, data poisoning, and data reconstruction attacks at different stages of FL training and provides APIs for these attacks. We summarize the customization APIs in Table 3 and introduce each type of attacks in details.

### 3.1 Model Poisoning Attacks

Model poisoning attacks modify the local models uploaded by clients in FL iterations. FedAttacker injects such attacks before FL aggregation in each FL iteration and modifies the local models directly. As an example, FedAttacker implements three modes of Byzantine attacks [60, 100–102], as follows.

(1) Zero mode [60] that poisons the client models by setting their weights to zero;
(2) *Random mode* [60] that manipulates client models by attributing random values to model weights; and
(3) *Flipping mode* [100] that updates the global model in the opposite direction by formulating the local model as $\mathbf{w}_g + (\mathbf{w}_g - \mathbf{w}_\ell)$, where $\mathbf{w}_g$ is the global model, and $\mathbf{w}_\ell$ is the real local model.

### 3.2 Data Poisoning Attacks

Data poisoning attacks modify local datasets of one or multiple clients to achieve some malicious goals, *e.g.*, degrading the performance of the global model or inducing the global model to misclassify some samples. As an example, in label flipping attack [89], a global "sybil" controls some clients and modifies their local data

by mislabeling samples of some classes to wrong classes. Given a source class (or label) $c_s$ and a target class $c_t$, all samples with class $c_s$ on the poisoned clients are re-labeled to an incorrect label $c_t$.

While poisoning local data can be performed by either a global "sybil" or malicious clients, to address a more general case, FedAttacker offers APIs to enable control over each local dataset.

### 3.3 Data Reconstruction Attacks

Data reconstruction attacks are performed by passive adversaries that attempts to infer sensitive information without actively interfering with the FL training or the local data. We assume that there is no leakage during the local training process in FL, as clients are assumed to be on their fully trusted local machines, according to the motivation of FL. As a result, data reconstruction attacks take the trained models (either the global model or the local models) or model updates to revert training data. For example, Deep Leakage from Gradients (DLG) attack [106] infers local training data from the publicly shared gradients. A passive adversary can use the global model from the previous FL training round and the newly obtained model to compute a "model update" between models in different FL training rounds to deduce the training data.

### 3.4 Integration of a New Attack

To customize a new attack, users should follow these steps: *i)* determine the type of the attack, *i.e.*, model poisoning, data poisoning, or data reconstruction; *ii)* create a new class for the attack and implement functions using the APIs, *e.g.*, *attack_model*(∗), *poison_data*(∗), and *reconstruct_data*(∗), to inject attacks at the appropriate stages of FL training; and *iii)* add the attack name to the corresponding enabler functions, *e.g.*, *is_model_poisoning_attack*(), within the FedAttacker class to ensure that the injected attacks are activated at the proper stages of FL training.

## 4 IMPLEMENTATION OF DEFENSES

FedDefender injects defense functions at different stages of FL aggregation at the server. Based on the point of injection, FedDefender

**Table 4: APIs for Different Types of Defenses in FedSecurity**

| Type of Defenses | APIs | Explanations |
|---|---|---|
| Before-Aggregation Defenses | *defend_before_aggregation*(*local_models*, *auxiliary_info*) | Modify the client models of the current FL iteration to mitigate (or eliminate) the impact of malicious local models. |
| | *is_defense_before_aggregation*() | Examine whether FedDefender is activated and whether the defense requires injecting functions before aggregating local models at the server. |
| On-Aggregation Defenses | *defend_on_aggregation*(*local_models*, *auxiliary_info*) | Take the local models of the current training round for aggregation and mitigate the impact of malicious clients by modifying aggregation, *e.g.*, altering aggregation functions. |
| | *is_defense_on_aggregation*() | Examine if the defense component is enabled and whether the current defense requires the injection of functions during aggregation. |
| After-Aggregation Defenses | *defend_after_aggregation*(*global_model*) | Directly modify the global model after aggregation using methods such as clipping or adding noise. |
| | *is_defense_after_aggregation*() | Examine if the defense component is activated and whether the current defense requires injecting functions after aggregation. |

The input *local_models* is a list of tuples that contain the number of samples and the local models submitted by clients in each FL iteration.
The input *auxiliary_info* can be any information that is utilized in the defense functions.

provides three types of functions to support defense mechanisms, including 1) before-aggregation, 2) on-aggregation, and 3) after-aggregation. Note that a defense may inject functions at one or multiple stages of FL aggregation. The APIs for defense functions in FedDefender are summarized in Table 4.

## 4.1 Before-Aggregation Defenses

Before-aggregation functions operate on local models at the FL server before aggregating the local models. We use Krum [9] as an example to demonstrate before-aggregation defenses.

**Krum.** Krum [9] tolerates $f$ Byzantine clients among $n$ clients by retaining only one local model that is the most likely to be benign as the global model. That is, Krum selects a single model as the global model in aggregation. A generalization of Krum is $m$-Krum [9] that selects $m$ client models with the $m$ lowest scores for aggregation, instead of choosing only one local model. This approach requires less than $\frac{n-m}{2} - 1$ clients to be malicious.

## 4.2 On-Aggregation Defenses

On-aggregation defense functions modify the aggregation function to a robust version that tolerates or mitigates impacts of the potential adversarial client models. As an example, RFA (Robust Federated Aggregation) [73] computes a geometric median of the client models in each iteration as the aggregated model, instead of simply averaging the client models. RFA defense effectively mitigates the impact of poisoned client models, as the geometric median can represent the central tendency of the client models, and the median point is chosen in a way to minimize the sum of distances between that point and the other client models of the current FL iteration. In practice, the geometric median is calculated using the Smoothed Weiszfeld Algorithm [73].

## 4.3 After-Aggregation Defenses

After-aggregation defense functions modify the aggregation result, *i.e.*, the global model, of each FL iteration to mitigate the effects of poisoned local models or protect the global model from potential adversaries. As an example, CRFL [96] clips the global model to bound the norm of the model each time after aggregation at the FL server. The FL server then adds Gaussian noise to the clipped global model before distributing the global model to the clients for the next FL iteration.

## 4.4 Integration of a New Defense

To implement a self-designed defense mechanism, users should first determine the stages to inject the defense functions (*i.e.*, before/on/after-aggregation), add a class for the new defense and implement the corresponding defense functions using the aforementioned APIs, *i.e.*, *defend_before_aggregation*(∗), *defend_on_aggregation*(∗), and *defend_after_aggregation*(∗), to inject functions at appropriate stages of FL. Note that some defenses involve more than one stage; thus, users need to implement all relevant functions. Users should add the name of the defense to the enabler functions to activate the injected function at the different stages of FL. The approach computes some scores using local models submitted by clients, and uses the scores to identify outlier local models before aggregating the local models. As such process only happens before aggregation, we only need to implement *defend_before_aggregation*(∗) for the defense class, and include the name of the defense in *is_defense_after_aggregation*().

## 5 EVALUATIONS

This section presents a comprehensive evaluation of FedSecurity to benchmark some well-known attacks and defenses in FL.

**Experimental setting.** A summary of datasets and models for evaluations can be found in Table 5. We utilize FedAVG in our experiments. By default, we employ ResNet20 and the non-i.i.d. CIFAR10 dataset (partition parameter $\alpha = 0.5$), as the non-i.i.d. setting to

Table 5: Models and datasets for evaluations.

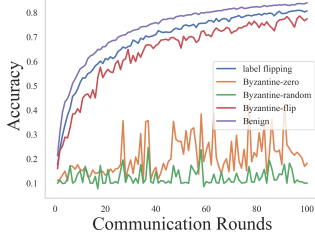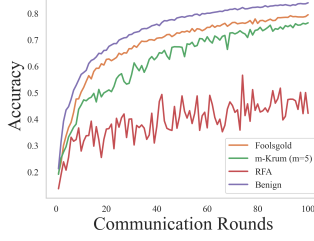| Model | ResNet20 [40] | ResNet56 [40] | CNN [67] | RNN (bi-LSTM) [67] | BERT [19] | Pythia-1B [8] |
|---|---|---|---|---|---|---|
| Dataset | CIFAR10 [48] | CIFAR100 [48] | FEMNIST [12] | Shakespeare [68] | 20News [52] | PubMedQA [63] |



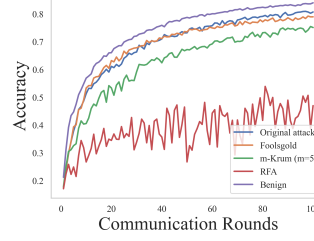Figure 3: Attack comparison.



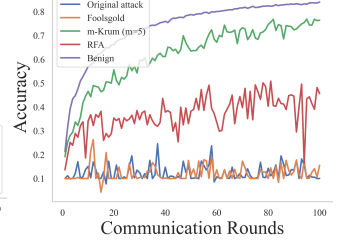Figure 4: Defense comparison.



Figure 5: Label flipping exps.



Figure 6: Random-Byzantine exps.
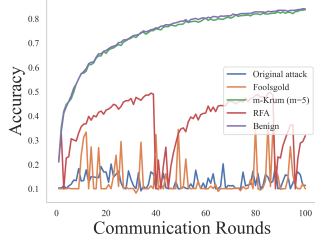


Figure 7: I.I.D. data evaluations.
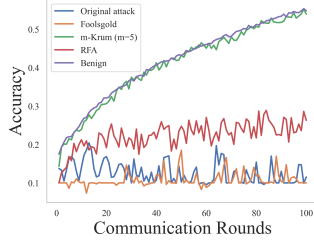


Figure 8: Scale # clients to 100.
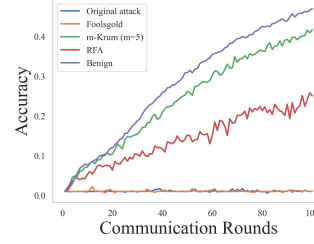


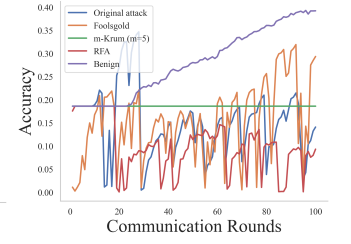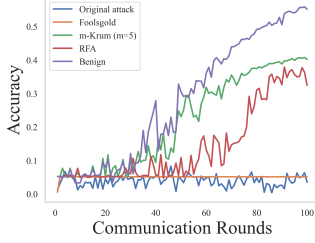Figure 9: ResNet56 (CV).



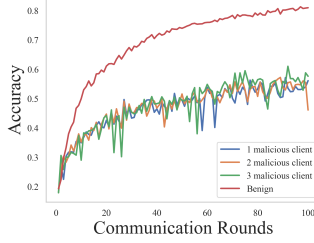Figure 10: RNN (NLP).



Figure 11: CNN (CV).



Figure 12: Varying # adversaries.

capture real-world scenarios closely. We further extend our evaluations to i.i.d. cases and various other models and datasets. For evaluations on LLMs, we utilize FedLLM [23] that trains LLMs in a federated manner. We employ the Pythia-1B model [8] and PubMedQA [44], a non-i.i.d. biomedical research dataset that contains 212,269 questions for question answering. We utilize the "artificial" subset for training and the "labelled" subset for testing. Evaluations are conducted on a server with 8 NVIDIA A100-SXM4-80GB GPUs.

By default, we use 10 clients for FL training, corresponding to real-world FL applications where the number of clients is typically less than 10, especially in business-to-business (B2B) scenarios. We also increase the number of clients to 100 in **Exp 5**, and set the number of clients to 70 in the real-world experiment, where we utilize real edge devices from Theta network [88] to showcase the scalability of FedSecurity (**Exp 10**). Unless otherwise noted, we set the percentage of malicious clients to 10%, and evaluate results

with the accuracy of the global model. We employ three attack mechanisms, including label flipping [89] and Byzantine attacks of random mode and flipping mode [16, 22, 100]. For the label flipping attack, we set the attack to modify the local and test data labels of malicious clients from label 3 to label 9 and label 2 to label 1. We utilize three defense mechanisms: $m$-Krum [9], Foolsgold [27], and RFA [73]. For $m$-Krum, we set $m$ to 5, which means 5 out of 10 local models participate in aggregation in each training round.

## 5.1 Evaluations on FL

**Exp 1: Attack Comparisons.** We evaluate the impact of attacks on test accuracy, using a no-attack scenario as a baseline. As illustrated in Figure 3, Byzantine attacks, specifically in the random and zero modes, substantially degrade accuracy. In contrast, the label flipping attack and the flipping mode of the Byzantine attack show a milder impact on accuracy. This can be attributed to the nature of Byzantine attacks, where Byzantine attackers would prevent the global model from converging, especially for the random mode that generates weights for models arbitrarily, causing the most significant deviation from the benign local model. In subsequent experiments, unless specified otherwise, we employ the Byzantine attack in the random mode as the default attack, as it provides the strongest impact compared with the other three attacks.

**Exp 2: Defense Comparisons.** We investigate potential impact of defense mechanisms on accuracy in the absence of attacks, *i.e.*, whether defense mechanisms inadvertently degrade accuracy when

all clients are benign. We incorporate a scenario without any defense or attack as our baseline. As illustrated in Figure 4, it becomes evident that when all clients are benign, involving defense strategies to FL training might lead to a reduction in accuracy. This decrease might arise from several factors: the exclusion of some benign local models from aggregation, *e.g.*, as in *m*-Krum, adjustments to the aggregation function, *e.g.*, as in RFA, or re-weighting local models, *e.g.*, as in Foolsgold. Specifically, the RFA defense mechanism significantly impacts accuracy as it computes a geometric median of the local models instead of leveraging the original FedAVG optimizer, which introduces a degradation in accuracy.

**Exp 3: Evaluations of defense mechanisms against activated attacks.** This experiment evaluates the effect of defense mechanisms against some attacks. We include two baseline scenarios: 1) an "original attack" scenario with an activated attack without any defense in place, and 2) a "benign" scenario with no activated attack or defense. We select label flipping attack and the random mode of Byzantine attack based on their impacts in **Exp1**, where label flipping has the least impact and the random mode of Byzantine attack exhibits the largest impact, as shown in Figure 3. Results for the label flipping and the random mode of Byzantine attacks are in Figure 5 and Figure 6, respectively. These results indicate that the defenses may contribute to minor improvements in accuracy for low-impact attacks, *e.g.*, Foolsgold in Figure 5. In certain cases, it is noteworthy that the defensive mechanisms may inadvertently compromise accuracy, such as the case with RFA in Figure 5. For high-impact attacks, such as the Byzantine attack of the random mode, Krum exhibits resilience, effectively neutralizing the negative impact of the attacks, as shown in Figure 6.

**Exp 4: Evaluations on i.i.d. data.** We select the random mode of the Byzantine attack, and employ Foolsgold, *m*-Krum (*m* = 5), and RFA to counteract the adverse effects of this attack. As shown in Figure 7, *m*-Krum is the most effective one among all the defense mechanisms, where the test accuracy is close to the case where all the FL clients are honest, *i.e.*, no attack scenario.

**Exp 5: Scaling the number of clients to 100.** We scale the number of clients to 100 and evaluates the defense mechanisms against the random mode of the Byzantine attack. We employ Foolsgold, *m*-Krum (with *m* = 5), and RFA to counteract the adverse effects of this attack. As shown in Figure 8, *m*-Krum is the most effective one among all the defense mechanisms, and the test accuracy is very close to the case where no attack happens. That is because in each FL iteration, *m*-Krum selects 5 local models that are more likely to be benign, which can represent the other local models, thus can achieve comparable accuracy compared with the benign case.

**Exp 6: Evaluations on different models.** We evaluate defense mechanisms against the random mode of the Byzantine attack with different models and datasets, including: *i*) ResNet56 + CIFAR100, *ii*) RNN + Shakespeare, and *iii*) CNN + FEMNIST. The results are shown in Figures 9, 10, and 11, respectively. The results show that while the defense mechanisms can mitigate the impact of attacks in most cases, some attacks may fail some tasks, *e.g.*, *m*-Krum fails RNN in Figure 10, and Foolsgold fails CNN in Figure 11. This is because the two defense mechanisms either select several local models for aggregation in each FL training round, or significantly re-weight the local models, which may eliminate some local models

that are important to the aggregation in the first several FL training iterations, leading to unchanged test accuracy in later FL iterations.

**Exp 7: Varying the number of malicious clients.** This experiment evaluates the impact of varying numbers of malicious clients on test accuracy. We utilize *m*-Krum to protect against 1, 2, and 3 malicious clients out of 10 clients in each FL training round. As shown in Figure 12, the test accuracy remains relatively consistent across different numbers of malicious clients, as in each FL training round, *m*-Krum selects a local model that is the most likely to be benign to represent the other models, effectively minimizing the impact of malicious client models on the aggregation.

## 5.2 Evaluations on Federated LLMs

We employ two LLMs, BERT [19] and Pythia [8], to showcase the scalability of FedSecurity and its applicability to federated LLM scenarios. We notice that some defenses (*e.g.*, Foolsgold [27]) that require memorizing intermediate results, such as models of previous FL training rounds, might encounter limitations when integrated with LLMs due to the significant cache introduced. Considering this, we utilize *m*-Krum for our experiments, as it does not require storing intermediate results and demonstrates consistent performance in most of our previous experiments.

**Exp 8: Evaluations of Krum against model replacement backdoor attack on BERT.** This experiment utilizes BERT [19] and the 20 news dataset [52] for a classification task. We employ 10 clients and set 1 client to be malicious in each FL training round. We set *m* to 5 in *m*-Krum, *i.e.*, 5 out of 10 local models participate in aggregation in each FL training round. Results in Figure 13 show that *m*-Krum effectively mitigates the adversarial effect, bringing the accuracy closer to the level of the attack-free case.

**Exp 9: Evaluations of Krum against the Byzantine attack on Pythia-1B.** We employ 7 clients for FL training, and 1 out of 7 clients is malicious in each round of FL training. We set the *m* parameter in *m*-Krum to 2, signifying that 2 out of 7 submitted local models participate in the aggregation in each FL training round. The performance is evaluated with the test loss. Results in Figure 14 show that Byzantine attack significantly increases the test loss during training. Nevertheless, *m*-Krum effectively mitigates the adversarial effect.

## 5.3 Evaluation in Real-World Applications

To demonstrate the scalability of our benchmark, we include an experiment using real-world devices, instead of simulations.

**Exp10: Evaluations in real-world applications.** We utilize edge devices from the Theta network [88] to validate the scalability of FedSecurity to real-world applications. The FL client package is integrated into Theta's edge nodes, which periodically fetches data from the Theta back-end. Subsequently, the FL training platform capitalizes on these Theta edge nodes and their associated data to train, fine-tune, and deploy machine learning models.

We select *m*-Krum as the defense and the Byzantine attack of random mode as the attack. Considering the challenges posed by real-world environments, such as devices equipped solely with CPUs (lacking GPUs), potential device connectivity issues, network latency, and limited storage on edge devices (for instance, some mobile devices might have less than 500MB of available storage),
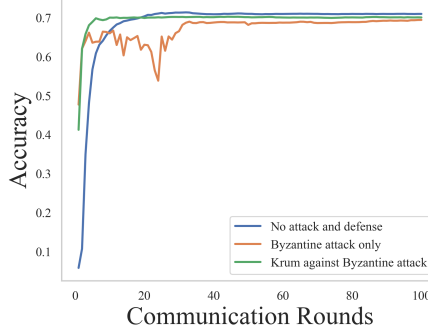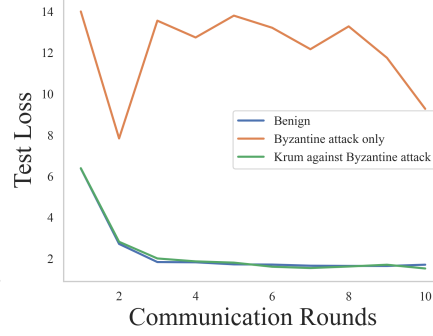
Figure 13: BERT evaluations.
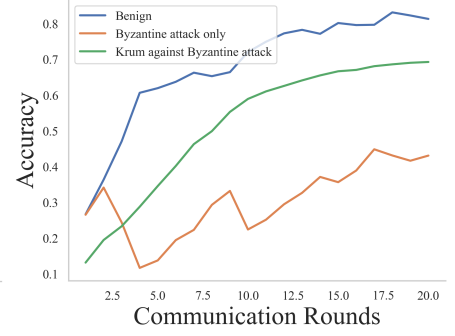


Figure 14: Pythia-1B evaluations.



Figure 15: Real-world application evaluation.



Figure 16: Real-world application. Yellow: aggregation server waiting time; pink: aggregation time; green: client training time; blue: client communication.
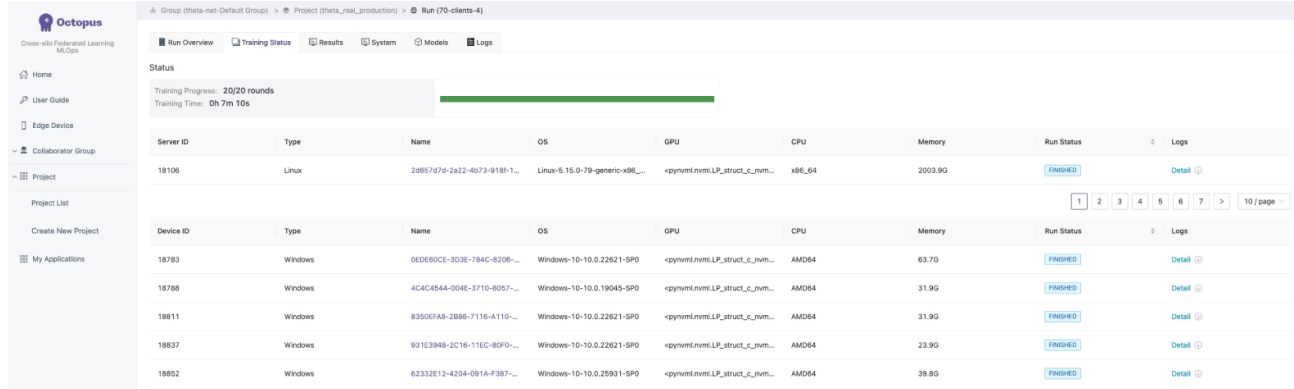


Figure 17: Real-world application: training status of devices.

we choose a simple task by employing the MNIST dataset for a logistic regression task. In our experimental setup, we deploy 70 client edge devices, designating 7 of these as malicious for each FL training round. For $m$-Krum, we set $m$ to 35, meaning that 35 out of the 70 local models are involved in aggregation during each FL training round. As illustrated in Figure 15, $m$-Krum mitigates the adversarial effect of the random-mode Byzantine attack. We also include a screenshot of the platform, as shown in Figure 16 for the

FL training process and Figure 17 for the training status of each real-world device.

## 6 CONCLUSION

This paper presents FedSecurity, a benchmark for adversarial attacks and corresponding defense strategies in FL. FedSecurity contains two components: FedAttacker that simulates various attacks

that can be injected during FL training, and FedDefender that facilitates defense strategies to mitigate the impacts of these attacks. The limitation of FedSecurity is that it does not support asynchronous FL and vertical FL yet. FedSecurity is open-sourced, and we welcome contributions from the research community to enrich the benchmark repository with novel attack and defense strategies to foster a diverse, comprehensive, and robust foundation for ongoing research in FL security.

## REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[2] Luca Antiga. 2023. Introducing PyTorch Lightning 2.0 and Fabric. https://lightning.ai/blog/introducing-lightning-2-0/ (2023).
[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In International Conference on Artificial Intelligence and Statistics. PMLR, 2938–2948.
[4] Gilad Baruch, Moran Baruch, and Yoav Goldberg. 2019. A little is enough: Circumventing defenses for distributed learning. Advances in Neural Information Processing Systems 32 (2019).
[5] Amos Beimel. 2011. Secret-sharing schemes: A survey. In International conference on coding and cryptology. Springer, 11–46.
[6] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. 2020. Flower: A friendly federated learning research framework. arXiv preprint arXiv:2007.14390 (2020).
[7] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In International Conference on Machine Learning. PMLR, 634–643.
[8] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. arXiv preprint arXiv:2304.01373 (2023).
[9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. Advances in neural information processing systems 30 (2017).
[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. Advances in neural information processing systems 33 (2020), 1877–1901.
[11] David Byrd and Antigoni Polychroniadou. 2020. Differentially private secure multi-party computation for federated learning in financial applications. In Proceedings of the First ACM International Conference on AI in Finance. 1–9.
[12] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. arXiv preprint arXiv:1812.01097 (2018).
[13] Chaochao Chen, Xiaohua Feng, Jun Zhou, Jianwei Yin, and Xiaolin Zheng. 2023. Federated Large Language Model : A Position Paper. arXiv preprint arXiv:2307.08925 (2023).
[14] Jiahui Chen, Yi Zhao, Qi Li, Xuewei Feng, and Ke Xu. 2022. FedDef: Defense Against Gradient Leakage in Federated Learning-Based Network Intrusion Detection Systems. IEEE Transactions on Information Forensics and Security 18 (2022), 4561–4576. https://api.semanticscholar.org/CorpusID:253420565
[15] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated learning of out-of-vocabulary words. arXiv preprint arXiv:1903.10635 (2019).
[16] Y. Chen, L. Su, and J. Xu. 2017. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. ACM on Measurement and Analysis of Computing Systems 1, 2 (2017), 1–25.
[17] Alexander Chowdhury, Hasan Kassem, Nicolas Padoy, Renato Umeton, and Alexandros Karargyris. 2022. A review of medical federated learning: Applications in oncology and cancer research. In Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 7th International Workshop, BrainLes 2021, Held in Conjunction with MICCAI 2021, Virtual Event, September 27, 2021, Revised Selected Papers, Part I. Springer, 3–24.
[18] Trung Dang, Om Thakkar, Swaroop Ramaswamy, Rajiv Mathews, Peter Chin, and Françoise Beaufays. 2021. Revealing and protecting labels in distributed

[19] training. Advances in Neural Information Processing Systems 34 (2021), 1727–1738.
[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).
[20] Dimitrios Dimitriadis, Mirian Hipolito Garcia, Daniel Madrigal Diaz, Andre Manoel, and Robert Sim. 2022. Flute: A scalable, extensible framework for high-performance federated learning simulations. arXiv preprint arXiv:2203.13789 (2022).
[21] Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Shanshan Han, Shantanu Sharma, Chaoyang He, Sharad Mehrotra, Salman Avestimehr, et al. 2023. Federated analytics: A survey. APSIPA Transactions on Signal and Information Processing 12, 1 (2023).
[22] 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.
[23] FedML Inc. 2023. Releasing FedLLM: Build Your Own Large Language Models on Proprietary Data using the FedML Platform. https://blog.fedml.ai/releasing-fedllm-build-your-own-large-language-models-on-proprietary-data-using-the-fedml-platform.
[24] Liam Fowl, Jonas Geiping, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. 2021. Robbing the fed: Directly obtaining private data in federated learning with modified models. arXiv preprint arXiv:2110.13057 (2021).
[25] Yann Fraboni, Richard Vidal, and Marco Lorenzi. 2021. Free-rider attacks on model aggregation in federated learning. In International Conference on Artificial Intelligence and Statistics. PMLR, 1846–1854.
[26] Shuhao Fu, Chulin Xie, Bo Li, and Qifeng Chen. 2019. Attack-resistant federated learning with residual-based reweighting. arXiv preprint arXiv:1912.11464 (2019).
[27] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2020. The Limitations of Federated Learning in Sybil Settings.. In RAID. 301–316.
[28] Erich Gamma, Richard Helm, Ralph Johnson, Ralph E Johnson, and John Vlissides. 1995. Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH.
[29] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting gradients-how easy is it to break privacy in federated learning? Advances in Neural Information Processing Systems 33 (2020), 16937–16947.
[30] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In NIPS.
[31] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access 7 (2019), 47230–47244.
[32] Rachid Guerraoui, Sébastien Rouault, et al. 2018. The hidden vulnerability of distributed learning in byzantium. In International Conference on Machine Learning. PMLR, 3521–3530.
[33] Sylvain Gugger. 2021. Introducing Hugging Face Accelerate. https://huggingface.co/blog/accelerate-library.
[34] Shanshan Han, Wenxuan Wu, Baturalp Buyukates, Weizhao Jin, Yuhang Yao, Qifan Zhang, Salman Avestimehr, and Chaoyang He. 2023. Kick Bad Guys Out! Zero-Knowledge-Proof-Based Anomaly Detection in Federated Learning. arXiv preprint arXiv:2310.04055 (2023).
[35] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604 (2018).
[36] Chaoyang He, Murali Annavaram, and Salman Avestimehr. 2020. Group knowledge transfer: Federated learning of large cnns at the edge. Advances in Neural Information Processing Systems 33 (2020), 14068–14080.
[37] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. 2020. FedML: A research library and benchmark for federated machine learning. arXiv preprint arXiv:2007.13518 (2020).
[38] Chaoyang He, Erum Mushtaq, Jie Ding, and Salman Avestimehr. 2021. Fednas: Federated deep learning via neural architecture search. (2021).
[39] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015), 770–778.
[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 770–778.
[41] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. 603–618.
[42] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, and Xuyun Zhang. 2021. Source inference attacks in federated learning. In 2021 IEEE International

*Conference on Data Mining (ICDM)*. IEEE, 1102–1107.

[43] Hongsheng Hu, Xuyun Zhang, Zoran Salcic, Lichao Sun, Kim-Kwang Raymond Choo, and Gillian Dobbie. 2023. Source Inference Attacks: Beyond Membership Inference Attacks in Federated Learning. *IEEE Transactions on Dependable and Secure Computing* (2023).

[44] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 2019. PubMedQA: A Dataset for Biomedical Research Question Answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2567–2577.

[45] Gueyoung Jung, Nathan Gnanasambandam, and Tridib Mukherjee. 2012. Synchronous Parallel Processing of Big-Data Analytics Services to Optimize Performance in Federated Clouds. *2012 IEEE Fifth International Conference on Cloud Computing* (2012), 811–818.

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

[47] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. 2021. Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*. PMLR, 5311–5319.

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

[49] Abhishek Kumar, Vivek Khimani, Dimitris Chatzopoulos, and Pan Hui. 2022. FedClean: A Defense Mechanism against Parameter Poisoning Attacks in Federated Learning. *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2022), 4333–4337. https://api.semanticscholar.org/CorpusID:249437417

[50] Kavita Kumari, Phillip Rieger, Hossein Fereidooni, Murtuza Jadliwala, and Ahmad-Reza Sadeghi. 2023. BayBFed: Bayesian Backdoor Defense for Federated Learning. *arXiv preprint arXiv:2301.09508* (2023).

[51] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. FedScale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*. PMLR, 11814–11827.

[52] Ken Lang. 1995. NewsWeeder: Learning to Filter Netnews. In *Machine Learning Proceedings 1995*, Armand Prieditis and Stuart Russell (Eds.). Morgan Kaufmann, San Francisco (CA), 331–339. https://doi.org/10.1016/B978-1-55860-377-6.50048-7

[53] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.

[54] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[55] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. 2019. Federated learning for keyword spotting. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 6341–6345.

[56] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. 2019. RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 1544–1551.

[57] Shenghui Li, Edith Ngai, Fanghua Ye, Li Ju, Tianru Zhang, and Thiemo Voigt. 2024. Blades: A unified benchmark suite for byzantine attacks and defenses in federated learning. In *2024 IEEE/ACM Ninth International Conference on Internet-of-Things Design and Implementation (IoTDI)*.

[58] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems* 2 (2020), 429–450.

[59] Xingyu Li, Zhe Qu, Shangqing Zhao, Bo Tang, Zhuo Lu, and Yao-Hong Liu. 2022. LoMar: A Local Defense Against Poisoning Attack on Federated Learning. *IEEE Transactions on Dependable and Secure Computing* 20 (2022), 437–450. https://api.semanticscholar.org/CorpusID:245837821

[60] Jierui Lin, Min Du, and Jian Liu. 2019. Free-riders in federated learning: Attacks and defenses. *arXiv preprint arXiv:1911.12560* (2019).

[61] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. 2021. Fate: An industrial grade platform for collaborative learning with data protection. *The Journal of Machine Learning Research* 22, 1 (2021), 10320–10325.

[62] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, et al. 2020. IBM Federated Learning: An Enterprise Framework White Paper v0.1. *arXiv preprint arXiv:2007.10987* (2020).

[63] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. 2022. BioGPT: generative pre-trained transformer for biomedical text generation and mining. *Briefings in Bioinformatics* 23, 6 (09 2022). https://doi.org/10.1093/bib/bbac409 arXiv:https://academic.oup.com/bib/article-pdf/23/6/bbac409/47144271/bbac409.pdf bbac409.

[64] Xinjian Luo, Yuncheng Wu, Xiaokui Xiao, and Beng Chin Ooi. 2021. Feature inference attack on model predictions in vertical federated learning. In *IEEE*

[65] Lingjuan Lyu, Han Yu, Xingjun Ma, Chen Chen, Lichao Sun, Jun Zhao, Qiang Yang, and S Yu Philip. 2022. Privacy and robustness in federated learning: Attacks and defenses. *IEEE transactions on neural networks and learning systems* (2022).

[66] Zhuo Ma, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H. Deng. 2022. ShieldFL: Mitigating Model Poisoning Attacks in Privacy-Preserving Federated Learning. *IEEE Transactions on Information Forensics and Security* 17 (2022), 1639–1654. https://api.semanticscholar.org/CorpusID:248358657

[67] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[68] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[69] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2016. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics*.

[70] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2018. Exploiting Unintended Feature Leakage in Collaborative Learning. *2019 IEEE Symposium on Security and Privacy (SP)* (2018), 691–706. https://api.semanticscholar.org/CorpusID:53099247

[71] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 691–706.

[72] Mustafa Safa Ozdayi, Murat Kantarcioglu, and Yulia R Gel. 2021. Defending against backdoors in federated learning with robust learning rate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 9268–9276.

[73] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. 2022. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing* 70 (2022), 1142–1154.

[74] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[75] Daniel Ramage. 2020. Federated Analytics: Collaborative Data Science Without Data Collection. *Google AI Blog* (May 2020). https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html

[76] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329* (2019).

[77] Mohammad Rasouli, Tao Sun, and Ram Rajagopal. 2020. Fedgan: Federated generative adversarial networks for distributed data. *arXiv preprint arXiv:2006.07228* (2020).

[78] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. 2021. Adaptive Federated Optimization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=LkFG3lB13U5

[79] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, et al. 2021. OpenFL: An open-source framework for Federated Learning. *arXiv preprint arXiv:2105.06413* (2021).

[80] Holger R Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, et al. 2022. NVIDIA FLARE: Federated Learning from Simulation to Real-World. *arXiv preprint arXiv:2210.13291* (2022).

[81] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[82] Virat Shejwalkar and Amir Houmansadr. 2021. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*.

[83] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1310–1321.

[84] Santiago Silva, Andre Altmann, Boris Gutman, and Marco Lorenzi. 2020. FedBioMed: A General Open-Source Frontend Framework for Federated Learning in Healthcare. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning: Second MICCAI Workshop*. Springer, 201–210.

[85] Jingwei Sun, Ang Li, Louis DiValentin, Amin Hassanzadeh, Yiran Chen, and Hai Li. 2021. Fl-wbc: Enhancing robustness against model poisoning attacks in federated learning from a client perspective. *Advances in Neural Information Processing Systems* 34 (2021), 12613–12624.

[86] Lichao Sun, Jianwei Qian, and Xun Chen. 2020. LDP-FL: Practical private aggregation in federated learning with local differential privacy. *arXiv preprint*

arXiv:2007.15789 (2020).

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

[88] Theta Network. 2023. Theta Network Website. https://thetatoken.org/.

[89] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*. Springer, 480–501.

[90] Dan Wang, Siping Shi, Yifei Zhu, and Zhu Han. 2022. Federated Analytics: Opportunities and Challenges. *IEEE Network* 36 (2022), 151–158.

[91] Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. 2023. ZeRO++: Extremely Efficient Collective Communication for Giant Model Training. *arXiv preprint arXiv:2306.10209* (2023).

[92] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J. Sohn, K. Lee, and D. Papailiopoulos. 2020. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*.

[93] Jianhua Wang. 2022. PASS: Parameters Audit-based Secure and Fair Federated Learning Scheme against Free Rider. *arXiv preprint arXiv:2207.07292* (2022).

[94] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. *ArXiv* abs/2007.07481 (2020).

[95] Zhibo Wang, Yuting Huang, Mengkai Song, Libing Wu, Feng Xue, and Kui Ren. 2022. Poisoning-assisted property inference attack against federated learning. *IEEE Transactions on Dependable and Secure Computing* (2022).

[96] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. 2021. CRFL: Certifiably robust federated learning against backdoor attacks. In *International Conference on Machine Learning*. PMLR, 11372–11382.

[97] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2020. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *Uncertainty in Artificial Intelligence*. PMLR, 261–270.

[98] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2020. SLSGD: Secure and Efficient Distributed On-device Machine Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 213–228.

[99] Yuexiang Xie, Zhen Wang, Daoyuan Chen, Dawei Gao, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. 2022. FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. *arXiv preprint arXiv:2204.05011* (2022).

[100] Jian Xu, Shao-Lun Huang, Linqi Song, and Tian Lan. 2022. Byzantine-robust federated learning through collaborative malicious gradient filtering. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1223–1235.

[101] H. Yang, X. Zhang, M. Fang, and J. Liu. Dec 2019. Byzantine-resilient stochastic gradient descent for distributed learning: A Lipschitz-inspired coordinate-wise median approach. In *IEEE CDC*.

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

[103] Jingwen Zhang, Jiale Zhang, Junjun Chen, and Shui Yu. 2020. Gan enhanced membership inference: A passive local attack in federated learning. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[104] Kai Zhang, Yu Wang, Hongyi Wang, Lifu Huang, Carl Yang, Xun Chen, and Lichao Sun. 2022. Efficient federated learning on knowledge graphs via privacy-preserving relation embedding aggregation. *arXiv preprint arXiv:2203.09553* (2022).

[105] Zhengming Zhang, Ashwinee Panda, Linyue Song, Yaoqing Yang, Michael Mahoney, Prateek Mittal, Ramchandran Kannan, and Joseph Gonzalez. 2022. Neurotoxin: Durable backdoors in federated learning. In *International Conference on Machine Learning*. PMLR, 26429–26446.

[106] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. *Advances in Neural Information Processing Systems* 32 (2019).

[107] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, et al. 2021. PySyft: A library for easy federated learning. *Federated Learning Systems: Towards Next-Generation AI* (2021), 111–139.