

---

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

---

Shanshan Han<sup>1</sup> Baturalp Buyukates<sup>2</sup> Zijian Hu<sup>3</sup> Han Jin<sup>2</sup>  
Weizhao Jin<sup>2</sup> Lichao Sun<sup>4</sup> Xiaoyang Wang<sup>5</sup> Chulin Xie<sup>5</sup> Kai Zhang<sup>4</sup>  
Qifan Zhang<sup>1</sup> Yuhui Zhang<sup>6</sup> Chaoyang He<sup>3</sup> Salman Avestimehr<sup>2,3</sup>  
<sup>1</sup> UCI <sup>2</sup> USC <sup>3</sup> FedML <sup>4</sup> Lehigh University <sup>5</sup> UIUC <sup>6</sup> Zhejiang University  
*shanshan.han@uci.edu buyukate@usc.edu zjh@fedml.ai*  
*hanjin@usc.edu weizhaoj@usc.edu lis221@lehigh.edu xw28@illinois.edu*  
*chulinx2@illinois.edu kaz321@lehigh.edu qifan.zhang@uci.edu*  
*zhangyuhui42@zju.edu.cn ch@fedml.ai avestime@usc.edu*

## Abstract

This paper introduces FedMLSecurity, a benchmark that simulates adversarial attacks and corresponding defense mechanisms in Federated Learning (FL). As an integral module of the open-sourced library FedML [22] that facilitates FL algorithm development and performance comparison, FedMLSecurity enhances the security assessment capacity of FedML. FedMLSecurity comprises two principal components: FedMLAttacker, which simulates attacks injected into FL training, and FedMLDefender, which emulates defensive strategies designed to mitigate the impacts of the attacks. FedMLSecurity is open-sourced<sup>1</sup> and is customizable to a wide range of machine learning models (*e.g.*, Logistic Regression, ResNet [23], GAN [19], etc.) and federated optimizers (*e.g.*, FedAVG [32], FedOPT [37], FedNOVA [46], etc.). Experimental evaluations in this paper also demonstrate the ease of application of FedMLSecurity to Large Language Models (LLMs), further reinforcing its versatility and practical utility in various scenarios.

## 1 Introduction

Federated Learning (FL) facilitates training across distributed data and empowers individual clients to utilize their local data to collaboratively train a machine learning model. Instead of sending their local data to a centralized server, in FL, clients train models on their local data and share the local models with the FL server, which then aggregates the local models into a global model. Upon aggregation, the global model is redistributed to the clients, enabling the clients to further fine-tune their local models. This iterative process continues until the model converges to an optimal solution.

FL maintains the privacy and security of client data by allowing clients to train locally without spreading their data to other parties. As a result of its privacy-preserving nature, FL has attracted considerable attention across various domains and has been utilized in numerous areas such as next-word prediction [21, 9, 36], hotword detection [27], financial risk assessment [8], and cancer risk prediction [11], demonstrating its wide-ranging versatility and potential for further expansion.

Even though FL does not require sharing client data with other parties, the decentralized and collaborative nature of FL may still expose systems to potential privacy and security risks. Adversarial clients may submit randomly generated models under the guise of contributing to training and pretend

---

<sup>1</sup><https://github.com/FedML-AI/FedML/tree/master/python/fedml/core/security>

they have been training or tamper with local models to disrupt the convergence of the global model. Adversarial clients may also attempt to manipulate the global model to misclassify a specific set of samples, which may degrade the quality of the global model.

**Related works.** In recent years, various benchmarks have emerged for FL, such as TensorFlow Federated [1], PySyft [53], FATE [29], Flower [3], FedScale [26], NVIDIA FLARE [39], OpenFL [38], Fed-BioMed [41], IBM Federated Learning [30], FederatedScope [49], and FLUTE [13]. However, none of them emphasizes potential adversarial attacks and possible defensive strategies in FL.

**Benchmark for attacks and defenses in FL.** This paper introduces FedMLSecurity, which is a robust security module that has been integrated into FedML [22], an open-source research library designed to foster the development of FL algorithms and to enable unbiased comparisons of their performance. FedMLSecurity comprises two primary components: FedMLAttacker and FedMLDefender. FedMLAttacker simulates prevalent attacks in FL to aid in understanding and preparing for potential security breaches. FedMLDefender is equipped with various defense mechanisms to counteract the threats simulated by FedMLAttacker, effectively mitigating these risks and enhancing the overall security of FL systems. Our contributions include the following:

- We introduce the first benchmark to enable evaluations of attacks and defenses in FL. Attacks in FedMLSecurity include Byzantine attacks of random/zero/flipping modes [10, 14], label flipping backdoor attack [43], deep leakage gradient [52], and model replacement backdoor attack [2]. The defense mechanisms in FedMLSecurity include norm clipping [42], Robust Learning rate [33], Krum [6], SLSGD [48], geometric median [10], weak DP [42], CClip [25], coordinate-wise median [51], RFA [34], Foolsgold [17], CRFL [47], and coordinate-wise trimmed mean [51].
- We design FedMLSecurity to be easy to be customized to a wide range of models and FL optimizers, including LLMs.
- We provide APIs in FedMLSecurity to enable users to integrate user-defined attacks and defenses.
- Through extensive experiments, we evaluate various attacks and defenses in FedMLSecurity. We further experimentally evaluate the benchmark on LLMs.

The *key takeaways* of our evaluations are as follows: 1) When an attack may happen, users can first evaluate the impact of this attack and then decide whether to involve a defensive strategy, as some attacks can have a limited impact on the quality of the global model. In such cases, introducing a defense, which impacts the aggregation result in an effort to mitigate the attack, may degrade the performance of the model even more than the potentially weak attack. 2) When implementing defenses in LLMs, one should consider the resource constraints of the machines and avoid using defenses that require memorizing a lot of information (such as models) from previous training rounds.

## 2 Preliminaries

This section presents adversarial models in FedMLSecurity and the interactions between FedMLSecurity and FedML.

### 2.1 Adversarial Model

Potential adversaries in FL can be classified into two categories: active and passive adversaries.

*Active Adversaries.* Active adversaries intentionally manipulate training data or trained models to achieve malicious goals. This could involve altering models to prevent global model convergence (*e.g.*, Byzantine attacks [10, 14]), or subtly misclassifying a specific set of samples to minimally impact the overall performance of the global model (*e.g.*, backdoor attacks [2, 44]). Active adversaries can take various forms, including: 1) malicious clients who manipulate their local models [2, 10, 14] or submit contrived models without actual training [45]; 2) a global “sybil” [43, 17] that has full access to the FL system and possesses complete knowledge of the entire system, including local and global models for each training round and clients’ local datasets. This “sybil” may also modify data within the FL system, such as clients’ local datasets and their submitted local models; and 3) external

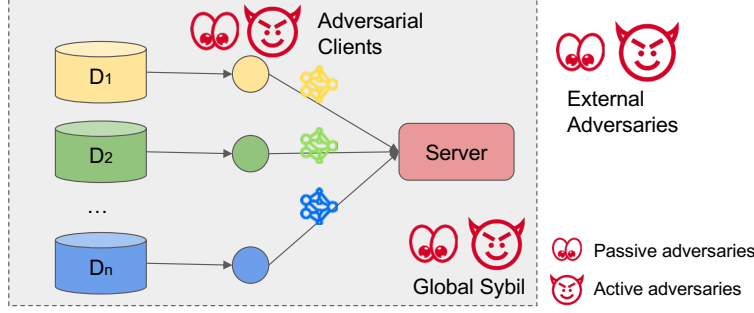


Figure 1: Active adversaries and passive adversaries in FL systems.

adversaries capable of monitoring the communication channel between 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 may still pose a threat to data privacy by potentially deducing sensitive information (such as original training data) from revealed models (or gradients) [52]. Examples of passive adversaries include: 1) an adversarial FL server attempting to infer local training data using submitted local models; 2) adversarial FL clients trying to deduce other clients’ training data using the global model provided by the server; and 3) external adversaries, such as hackers, who can access the communication channel to acquire local and global models transferred between clients and the FL server.

The adversarial model of FL systems is summarized in Figure 1.

## 2.2 Interaction with FedML

FedMLSecurity serves as an external component that injects attacks and defense mechanisms at different stages of FL training without altering the existing training processes in FedML. We adopt the singleton design pattern [18] to maintain the black box nature of FedMLSecurity. This pattern ensures the existence of only one instance of a class while providing global access to that instance. FedMLSecurity creates a single instance of both FedMLAttacker and FedMLDefender, which are accessible by other FedML objects. The singleton instance maintains a unified shared state, offering a single point of access to specific resources for FedMLAttacker and FedMLDefender.

*Injection of attacks.* Existing literature categorizes attacks in FL into various types, such as byzantine attacks [10, 14], backdoor attacks [2, 44], model poisoning attacks [14, 40, 4], data reconstructing attacks [52], data poisoning attacks [43], free-rider attacks [45], inference attacks [31], and so on. Without loss of generality, FedMLAttacker classifies attacks in FL into three classes based on the targets of the attacks as well as the stages of FL training at which the attacks are executed, including data poisoning attacks, model poisoning attacks, and data reconstruction attacks.

- **Data poisoning attacks:** Such attacks modify clients’ local datasets thus injected at clients [43, 12].
- **Model poisoning attacks:** Such attacks temper with local models submitted by clients. To account for all active adversaries (as discussed in §2.1) that may conduct such attacks, FedMLAttacker injects such attacks before the aggregation of local models in each FL training round at the server, so that FedMLAttacker can get access to all client models submitted in this training round.
- **Data reconstruction attacks:** Such attacks use the local models (or model updates) to infer information about the training data. In an FL system, the FL server may be interested in clients’ data and uses the global models and the submitted local models to reconstruct data; also, a client may be interested in other clients’ data when receiving a global model from the FL server and tries to reconstruct sensitive information using the global model and its local model. To cover these scenarios, FedMLAttacker injects such attacks at the FL server, where it has access to all local models and the global model of each iteration.

```

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

```

Figure 2: Byzantine attack configuration [10, 14].

```

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

```

Figure 3: Krum defense configuration [6].

*Injection of defenses.* FedMLDefender incorporates defense mechanisms to mitigate (or eliminate) the impacts of the injected attacks. Recognizing that the defenses either address issues related to tampered local models<sup>2</sup> or prevent adversaries from deducing information from the local/global models shared between clients and the FL server, to get access to all local models and global models in each FL training round, FedMLDefender deploys defenses at the FL server, and injects functions at different stages of the FL aggregation, including:

- Before-aggregation functions that modify local models submitted by clients.
- On-aggregation functions that modify the FL aggregation function to mitigate the impacts of local models submitted by adversarial clients.
- After-aggregation functions that modify the aggregated global model (*e.g.*, by adding noise or clipping) to protect the real global model or improve the quality of the global model.

**Configuration.** To activate FedMLAttacker and FedMLDefender, users can set two parameters, “enable\_attack” and “enable\_defense” to be true in the .yaml configuration file, and add additional parameters for the injected attack and defense, if necessary. We give sample configurations for byzantine attack [10, 14] and Krum defense [6] in Figures 2 and 3, respectively.

**Extensibility.** FedMLSecurity is easily extensible to various machine learning models, including Logistic Regression, LeNet, ResNet, CNN, RNN, GAN, and so on. It can also be extended to different federated optimizers, including FedAVG, FedOPT, FedPROX, Split NN, FedGKT, FedGAN, FedNAS, FedNOVA, FedSEG, Vertical FL, and so on.

### 3 Implementation of Attacks in FedMLAttacker

FedMLAttacker injects model poisoning attacks, data poisoning attacks, and data reconstruction attacks at different stages of FL training and provides APIs for different types of attacks.

#### 3.1 Model Poisoning Attacks

Model poisoning attacks are designed to alter the local models submitted by clients. In FedMLAttacker, such attacks are injected before FL aggregation in each iteration, effecting modifying each local model directly. Model poisoning attacks implemented in FedMLAttacker include Byzantine attacks [10, 14] and Model Replacement Backdoor attack [2]. As an example, Byzantine attacks disrupt the training process by fabricating counterfeit local models, thus obstructing the convergence of the global model. FedMLAttacker implements Byzantine attacks by selecting clients to poison in each FL iteration and modifying their local models with various modes. FedMLAttacker has three modes of Byzantine attacks, as follows:

- Zero mode: This mode poisons the client models by setting each parameter of the models to zero.
- Random mode: This mode manipulates client models by attributing a random value to each parameter.
- Flipping mode: This mode updates the global model in the opposite direction by formulating a poisoned local model, denoted as  $\mathbf{w}'_\ell$ , based on the global model  $\mathbf{w}_g$  and the real local model  $\mathbf{w}_\ell$ :  $\mathbf{w}'_\ell = \mathbf{w}_g + (\mathbf{w}_g - \mathbf{w}_\ell)$ .

<sup>2</sup>Note that poisoning local datasets also results in tampered local models.

**APIs for Model Poisoning Attacks.** FedMLAttacker has two APIs for model poisoning attacks.

- `attack_model(local_models, auxiliary_info)`: This function takes the local models submitted by clients in the current FL iteration and modifies the local models. The input `local_models` is a list of tuples containing the number of data samples and the submitted local model for each client. The input `auxiliary_info` may include any information used in the defense, typically the global model from the last FL iteration.
- `is_model_poisoning_attack()`: This function checks whether the attack component is activated and whether the attack modifies client models.

### 3.2 Data poisoning attacks.

Data poisoning attacks modify local datasets of some clients to induce the clients to train their local models using poisoned data to achieve some malicious goals, such as degrading the performance of the global model or inducing the global model to misclassify specific samples. As an example, in label flipping attack [43], 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  $c_t$ , the local dataset of each poisoned client is modified such that all samples with class  $c_s$  are now associated with an incorrect label  $c_t$ .

**APIs for Data Poisoning Attacks.** FedMLAttacker has two APIs for data poisoning attacks.

- `poison_data(dataset)`: This function takes a local dataset and mislabels a set of chosen samples based on the clients’ (or attackers’) requirements, which are included in configuration. Normally, clients would change labels of a specific subset of samples to some other labels in the same dataset, or label a set of samples to new classes that do not exist in the dataset.
- `is_data_poisoning_attack()`: This function examines whether FedMLAttacker is enabled and whether the attack requires poisoning the datasets.

### 3.3 Data reconstruction attacks.

Unlike the other two types of attacks that require an active adversary to intentionally manipulate data or models to achieve malicious goals, data reconstruction attacks are performed by a passive adversary that is attempting to infer sensitive information without actively interfering with the FL training or the local data. In the context of FL, we assume that there is no leakage during the local training process, as clients train models using their local data at their fully trusted local machines. As a result, data reconstruction attacks take the trained models (either the global model or the local models) and aim to revert the training data. An example data reconstruction attack is the Deep Leakage from Gradients (DLG) attack [52] that infers private local training data from the publicly shared gradients in distributed training. In the context of FL, a passive adversary can use the global model from the previous FL training round and the newly obtained model to compute a “gradient update” between models of different FL training iterations to deduce the training data. The adversary initializes dummy input data and labels using a normal distribution  $\mathcal{N}(0, 1)$  and then calculates dummy gradients with the shared gradients while updating input data and labels to match the model. Upon iterating the optimization, the attacker tries to revert the original training data.

**APIs for Data Reconstruction Attacks.** We have two APIs for data reconstruction attacks.

- `reconstruct_data(model, auxiliary_info)`: This function takes a client model or a global model to reconstruct the training data. It also takes some extra information (`auxiliary_info`) to help infer.
- `is_data_reconstruction_attack()`: This function examines whether the attack component is enabled and whether the attack requires to reconstruct training data using the trained models.

### 3.4 Integrate a New Attack

To customize a new attack, users should first determine the type of the attack, *i.e.*, model poisoning, data poisoning, or data reconstruction. Next, users can create a new class for the attack in

fedml/core/security/attack and implement the corresponding functions using the APIs, *e.g.*, `attack_model(*)`, `poison_data(*)`, and `reconstruct_data(*)`, to inject attacks at the appropriate stages of FL training. Finally, users should add the attack name to the corresponding enabler functions, *i.e.*, `is_model_poisoning_attack()`, `is_data_poisoning_attack()`, and `is_data_reconstruction_attack()`, within the `FedMLAttacker` class to ensure that the injected attacks are activated at the proper stages of FL training.

## 4 Implementation of Defenses in FedMLDefender

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

### 4.1 Before-aggregation Defenses

Before-aggregation functions operate on local models of each FL training iteration to mitigate (or eliminate) impacts of potential attacks. We use Krum [6] and Foolsgold [17] as examples.

**Krum.** Krum [6] tolerate  $f$  Byzantine clients among  $n$  clients by retaining only one local model that is the least likely to be poisoned, *i.e.*, the local model that is the most likely to be honest is chosen as the global model. Before aggregating local models at the FL server, Krum computes a score for each client, and selects the local model with the minimum score as the global model of the current FL iteration, which means the selected local model is “closest” to its  $n - f - 2$  neighbor models. To achieve this, Krum computes the squared Euclidean distance between each pair of client models  $\mathbf{w}_i$  and  $\mathbf{w}_j$  as  $\|\mathbf{w}_i - \mathbf{w}_j\|^2$ , and selects the closest  $n - f - 2$  local models to each client model  $\mathbf{w}_i$ , denoted as  $\mathcal{N}(\mathbf{w}_i)$ . The Krum score of client  $i$  is then computed as  $s(\mathbf{w}_i) = \sum_{\mathbf{w}_j^{(i)} \in \mathcal{N}(\mathbf{w}_i)} \|\mathbf{w}_i - \mathbf{w}_j^{(i)}\|^2$ .

An advanced version of Krum is  $m$ -Krum [6] which selects  $m$  client models with the smallest Krum scores for aggregation instead of only one. This approach requires the  $m$  client models to be benign, which leads to  $n - m > 2f + 2$ . Compared with Krum,  $m$ -Krum allows more clients to participate in FL training, which improves the quality of the global model. However, proper  $m$  selection requires the knowledge of the (maximum possible) number of Byzantine clients, as Byzantine local models should not be included in the aggregation. This can be a limitation, as the number of Byzantine clients may not always be available in real systems.

**Foolsgold.** Foolsgold [17] uses a score to evaluate inter-client contribution similarity for each client model and re-weights client models to reduce the impact of potentially adversarial client models while increasing the influence of those that are more likely to be benign. To represent each client model, Foolsgold first computes a “feature importance” that is then used to calculate a score for each client model with other models based on cosine similarities. Specifically, Foolsgold applies a logit function, *i.e.*, the inverse sigmoid function, to the scores to achieve higher divergence for values near the tails and to avoid penalizing honest clients with a low, non-zero similarity value. During aggregation, each client model is re-weighted using the score.

**APIs for before-aggregation functions.** We provide two APIs for before-aggregation functions:

- `defend_before_aggregation(local_models, auxiliary_info)`: This function modifies the client models of the current FL iteration. The input `local_models` is a list of tuples that contain the number of samples and the local model submitted by each client in the current FL iteration. The input `auxiliary_info` can be any information that is utilized in the defense functions.
- `is_defense_before_aggregation()`: This function checks whether the defense component is activated and whether the current defense requires injecting functions before aggregating local models at the FL server.

## 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. We take Robust Federated Aggregation (RFA) defense [34] as an example. RFA computes a geometric median of the client models in each FL iteration when aggregating client models, 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. Consider a list of  $n$  client models  $\mathbf{w}_1, \dots, \mathbf{w}_n$  in an FL training round and their respective weights  $\alpha_1, \dots, \alpha_n$ . The geometric median  $\theta$  can be determined by minimizing  $\sum_{i=1}^n \alpha_i \|\theta - \mathbf{w}_i\|$ , where  $\|\cdot\|$  denotes the Euclidean norm between  $\theta$  and a local model  $\mathbf{w}_i$ . In practice, the geometric median is calculated using the Smoothed Weiszfeld Algorithm [34].

**APIs for on-aggregation defenses.** We provide two APIs for on-aggregation defense functions:

- `defend_on_aggregation(local_models, auxiliary_info)`: This function takes the local models of the current training round for aggregation. The input `local_models` is a list of tuples that contain the number of samples and the local model submitted by each client in the current FL iteration. The input `auxiliary_info` can include any information required by the defense functions.
- `is_defense_on_aggregation()`: This function checks if the defense component is enabled and whether the current defense requires the injection of functions during aggregation.

## 4.3 After-aggregation Defense

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 [47] 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.

**APIs for After-Aggregation Defenses.** We provide two APIs to support after-aggregation defenses:

- `defend_after_aggregation(global_model)`: This function directly modifies the global model after aggregation using methods such as clipping or adding noise.
- `is_defense_after_aggregation()`: This function checks if the defense component is activated and whether the current defense requires injecting functions after aggregation.

## 4.4 Integrate 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 at `fedml/core/security/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 at `fedml/core/security/fedml_defender.py` to activate the injected function at the different stages of FL. As an example, we integrate a new defense that detects outlier local models to FedMLDefender in [7].

## 5 Experiments

This section offers a comprehensive evaluation of how FedMLSecurity facilitates the benchmarking of various attack and defense strategies in FL. Our initial set of experiments focused on the effectiveness of some representative attacks, such as different modes of Byzantine attacks [50] and label-flipping attacks [43]. Then, we turned our attention towards representative defense mechanisms that have gained significant attention in the literature, such as  $m$ -Krum [6], Foolsgold [17], and

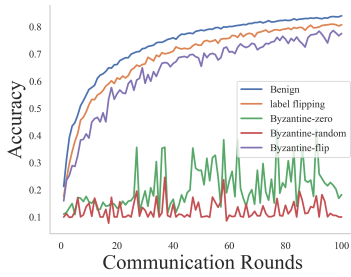


Figure 4: Attack comparison.

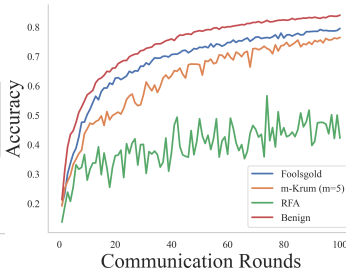


Figure 5: Defense comparison.

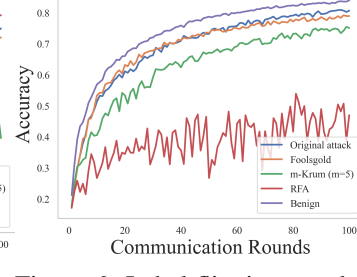
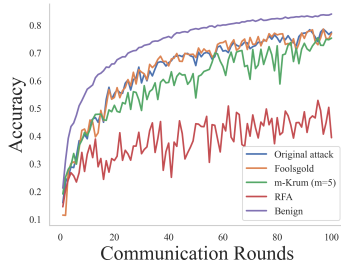
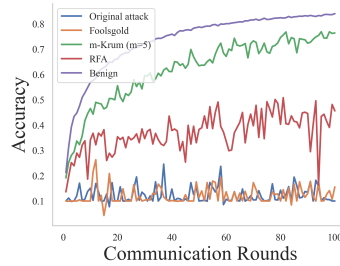


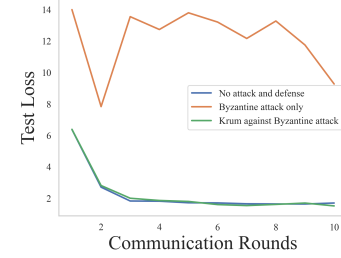
Figure 6: Label flipping attacks with defenses.



(a) Flipping mode.



(b) Random mode.



(c) FedLLM training with Krum and Byzantine attack.

Figure 7: Effectiveness of various defenses under Byzantine attacks.

RFA [34]. We evaluated the impact of injecting the defenses by analyzing whether the defense mechanisms, when activated, compromise the quality of the model by negatively influencing its accuracy, even in the absence of attacks. Next, we evaluated the performance of the defense mechanisms by employing these defenses in the context of activated attacks. Finally, we demonstrate the practical application of FedMLSecurity for LLMs by activating the Byzantine attack and the Krum defense mechanism during FL training.

**Experimental setting.** We utilized the FedAVG optimizer in our experiments. For evaluations on FL, we employed the non-i.i.d. CIFAR10 dataset and the ResNet20 model. Our hardware setup for these evaluations comprised 8 Nvidia Quadro RTX 5000 GPUs. For evaluations on LLMs, we utilized FedLLM<sup>3</sup> that trains LLM in a federated setting. We employed a non-i.i.d. biomedical research dataset, PubMedQA [24], which contains 212,269 questions for question answering, and the Pythia-1b model [5], which has 1B parameters. Our evaluations were facilitated by 8 Tesla V100-SXM2-32GB GPUs.

## 5.1 Evaluations on FL

We use 10 clients for FL training and set the percentage of malicious clients to 10% when injecting attacks. The selected attacks for evaluation are the label flipping attack and Byzantine attacks of random mode and flipping mode. 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. Regarding defenses, we utilize three mechanisms:  $m$ -Krum, Foolsgold, and RFA. For  $m$ -Krum, the parameter  $m$  is set to 5, which means 5 out of 10 submitted local models participate in aggregation in each FL training round. The results are evaluated with the accuracy of the FL model.

**Exp1: Attack Comparisons.** This experiment measures the effectiveness of various attacks in terms of test accuracy. A no-attack scenario is included as a baseline for comparison. The results in Figure 4 indicate that the Byzantine attacks (in both the random and zero modes) significantly impact

<sup>3</sup><https://blog.fedml.ai/releasing-fedllm-build-your-own-large-language-models-on-proprietary-data-using-the-fedml-platform/>



accuracy, whereas the label flipping attack and Byzantine flipping mode attack have less impact on accuracy.

**Exp2: Defense Comparisons.** In this experiment, we deactivated attacks to examine whether the defense mechanisms cause a decrease in accuracy when all clients are benign. A no-defense situation is also included as a baseline. As depicted in Figure 5, all defense strategies slightly reduce the accuracy of results when all clients are benign, which may be attributed to the potential exclusion of some benign local models, alteration of the aggregation function, or reweighting of local models. Specifically, the RFA defense mechanism significantly affects accuracy as it computes a geometric median of the local models instead of using the original FedAVG optimizer.

**Exp3: Evaluations of defense mechanisms against activated attacks.** This experiment evaluates the effect of defense mechanisms in the context of ongoing attacks. We include two baseline scenarios: 1) an “original attack” scenario with an activated attack but without any defense, and 2) a “benign” scenario with no activated attack or defense. Results for the label flipping and Byzantine attacks are in Figure 6, Figure 7a, and Figure 7b, respectively. The results indicate that the defenses may contribute to minor improvements in accuracy for low-impact attacks, *e.g.*, Foolsgold in Figure 6 and Figure 7a. In certain cases, it is also noteworthy that the defensive mechanisms may inadvertently compromise accuracy, such as the case with RFA in Figure 6 and Figure 7a. For high-impact attacks, such as the Byzantine attack of Random and Zero modes, the Krum defense mechanism exhibits resilience, effectively neutralizing the negative impact of the attacks, as shown in Figure 7b.

## 5.2 Evaluations on LLMs

We employed 7 clients for FL training, and 1 out of 7 clients is malicious in each round of FL training. We apply LLMs to our benchmark to show the scalability of FedMLSecurity.

**Exp4: Evaluations of Krum defense against the Byzantine attack on LLMs.** In this experiment, we utilize the  $m$ -Krum defense to counter an injected random-mode Byzantine attack. 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 based on the test loss during FL training, as shown in Figure 7c. These results reveal that as the number of communication rounds increases, the loss decreases. In addition, we observe that the Byzantine attack significantly increases the test loss during training. Nevertheless, the  $m$ -Krum defense effectively mitigates the adversarial effect, bringing it closer to the level in the attack-free experiment. This showcases the effectiveness of the  $m$ -Krum defense.

## 6 Conclusion

This paper presents FedMLSecurity, an integrated module within FedML [22] designed to simulate potential adversarial attacks and corresponding defense strategies in FL. FedMLSecurity contains two principal components: FedMLAttacker, which simulates various attacks injected during the FL training process, and FedMLDefender, which facilitates defense strategies to mitigate the impacts of these attacks. While FedMLSecurity offers a robust foundation, we recognize its potential for further enhancement. Our plan for improvement includes the following aspects: 1) conducting more comprehensive experiments on LLMs to provide a more comprehensive understanding of vulnerabilities in LLMs; and 2) designing and implementing advanced defense mechanisms against potential adversaries in asynchronous FL scenarios. FedMLSecurity 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. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.
  - [3] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchu Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.
  - [4] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.
  - [5] 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. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*, 2023.
  - [6] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30, 2017.
  - [7] Baturalp Buyukates, Chaoyang He, Shanshan Han, Zhiyong Fang, Yupeng Zhang, Jieyi Long, Ali Farahanchi, and Salman Avestimehr. Proof-of-contribution-based design for collaborative machine learning on blockchain. *arXiv preprint arXiv:2302.14031*, 2023.
  - [8] David Byrd and Antigoni Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–9, 2020.
  - [9] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*, 2019.
  - [10] Y. Chen, L. Su, and J. Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *ACM on Measurement and Analysis of Computing Systems*, 1(2):1–25, Dec 2017.
  - [11] Alexander Chowdhury, Hasan Kassem, Nicolas Padoy, Renato Umeton, and Alexandros Karargyris. 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*, pages 3–24. Springer, 2022.
  - [12] Trung Dang, Om Thakkar, Swaroop Ramaswamy, Rajiv Mathews, Peter Chin, and Françoise Beaufays. Revealing and protecting labels in distributed training. *Advances in Neural Information Processing Systems*, 34:1727–1738, 2021.
  - [13] Dimitrios Dimitriadis, Mirian Hipolito Garcia, Daniel Madrigal Diaz, Andre Manoel, and Robert Sim. Flute: A scalable, extensible framework for high-performance federated learning simulations. *arXiv preprint arXiv:2203.13789*, 2022.
  - [14] M. Fang, X. Cao, J. Jia, and N. Gong. Local model poisoning attacks to Byzantine-robust federated learning. In *USENIX Security*, Aug 2020.

- [15] Shuhao Fu, Chulin Xie, Bo Li, and Qifeng Chen. Attack-resistant federated learning with residual-based reweighting. *arXiv preprint arXiv:1912.11464*, 2019.
- [16] C. Fung, C. J. M. Yoon, and I. Beschastnikh. Mitigating sybils in federated learning poisoning. Aug 2018. Available on arXiv:1808.04866.
- [17] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *RAID*, pages 301–316, 2020.
- [18] Erich Gamma, Richard Helm, Ralph Johnson, Ralph E Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.
- [19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.
- [20] Rachid Guerraoui, Sébastien Rouault, et al. The hidden vulnerability of distributed learning in byzantium. In *International Conference on Machine Learning*, pages 3521–3530. PMLR, 2018.
- [21] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.
- [22] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. FedML: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.
- [23] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.
- [24] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 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)*, pages 2567–2577, 2019.
- [25] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. Byzantine-robust learning on heterogeneous datasets via bucketing. *arXiv preprint arXiv:2006.09365*, 2020.
- [26] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. FedScale: Benchmarking model and system performance of federated learning at scale. In *International Conference on Machine Learning*, pages 11814–11827. PMLR, 2022.
- [27] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345, 2019.
- [28] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.
- [29] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. Fate: An industrial grade platform for collaborative learning with data protection. *The Journal of Machine Learning Research*, 22(1):10320–10325, 2021.
- [30] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, et al. IBM Federated Learning: An Enterprise Framework White Paper v0.1. *arXiv preprint arXiv:2007.10987*, 2020.

- [31] Xinjian Luo, Yuncheng Wu, Xiaokui Xiao, and Beng Chin Ooi. Feature inference attack on model predictions in vertical federated learning. In *IEEE International Conference on Data Engineering (ICDE)*, pages 181–192. IEEE, 2021.
- [32] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2016.
- [33] Mustafa Safa Ozdayi, Murat Kantarcioglu, and Yulia R Gel. Defending against backdoors in federated learning with robust learning rate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9268–9276, 2021.
- [34] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 70:1142–1154, 2022.
- [35] F. Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, May 1994.
- [36] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.
- [37] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.
- [38] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, et al. Openfl: An open-source framework for federated learning. *arXiv preprint arXiv:2105.06413*, 2021.
- [39] Holger R Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, et al. NVIDIA FLARE: Federated learning from simulation to real-world. *arXiv preprint arXiv:2210.13291*, 2022.
- [40] Virat Shejwalkar and Amir Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*, 2021.
- [41] Santiago Silva, Andre Altmann, Boris Gutman, and Marco Lorenzi. Fed-BioMed: 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*, pages 201–210. Springer, 2020.
- [42] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.
- [43] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*, pages 480–501. Springer, 2020.
- [44] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J. Sohn, K. Lee, and D. Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*, Dec 2020.
- [45] Jianhua Wang. Pass: Parameters audit-based secure and fair federated learning scheme against free rider. *arXiv preprint arXiv:2207.07292*, 2022.
- [46] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *ArXiv*, abs/2007.07481, 2020.
- [47] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. CRFL: Certifiably robust federated learning against backdoor attacks. In *International Conference on Machine Learning*, pages 11372–11382. PMLR, 2021.

- [48] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. SLSGD: Secure and Efficient Distributed On-device Machine Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 213–228. Springer, 2020.
- [49] Yuexiang Xie, Zhen Wang, Daoyuan Chen, Dawei Gao, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. *arXiv preprint arXiv:2204.05011*, 2022.
- [50] H. Yang, X. Zhang, M. Fang, and J. Liu. Byzantine-resilient stochastic gradient descent for distributed learning: A Lipschitz-inspired coordinate-wise median approach. In *IEEE CDC*, Dec 2019.
- [51] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR, 2018.
- [52] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in Neural Information Processing Systems*, 32, 2019.
- [53] Alexander Ziller, Andrew Trask, Antonio Lopardo, Benjamin Szymkow, Bobby Wagner, Emma Bluemke, Jean-Mickael Nounahon, Jonathan Passerat-Palmbach, Kritika Prakash, Nick Rose, et al. PySyft: A library for easy federated learning. *Federated Learning Systems: Towards Next-Generation AI*, pages 111–139, 2021.