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

Shanshan Han[1] Baturalp Buyukates[2] Zijian Hu[3] Han Jin[2] Weizhao Jin[2]
Lichao Sun[4] Xiaoyang Wang[5] Wenxuan Wu[6] Chulin Xie[5] Yuhang Yao[7]
Kai Zhang[4] Qifan Zhang[1] Yuhui Zhang[8] Salman Avestimehr[2,3] Chaoyang He[3]
[1] UCI [2] USC [3] FedML [4] Lehigh University
[5] UIUC [6] Texas A&M University [7] CMU [8] Zhejiang University
shanshan.han@uci.edu buyukate@usc.edu zjh@fedml.ai hanjin@usc.edu
weizhaoj@usc.edu lis221@lehigh.edu xw28@illinois.edu ww6726@tamu.edu
chulinx2@illinois.edu yuhangya@andrew.cmu.edu kaz321@lehigh.edu
qifan.zhang@uci.edu zhangyuhui42@zju.edu.cn avestime@usc.edu ch@fedml.ai

Abstract

This paper introduces FedMLSecurity, a benchmark designed to simulate adversarial attacks and corresponding defense mechanisms in Federated Learning (FL). As an integral module of the open-sourced library FedML that facilitates FL algorithm development and performance comparison, FedMLSecurity enhances FedML's capabilities to evaluate security issues and potential remedies in FL. FedMLSecurity comprises two major components: FedMLAttacker that simulates attacks injected during FL training, and FedMLDefender that simulates defensive mechanisms to mitigate the impacts of the attacks. FedMLSecurity is open-sourced [1] and can be customized to a wide range of machine learning models (*e.g.*, Logistic Regression, ResNet, GAN, etc.) and federated optimizers (*e.g.*, FedAVG, FedOPT, FedNOVA, etc.). FedMLSecurity can also be applied to Large Language Models (LLMs) easily, demonstrating its adaptability and applicability 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, and the server aggregates the local models into a global model. Then, the global model is redistributed to the clients, enabling the clients to further fine-tune the model using their local data. 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 [31, 14, 76], hotword detection [58], financial risk assessment [11], and cancer risk prediction [16], demonstrating its wide-ranging versatility and potential for further expansion.

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

Submitted to the 37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks. Do not distribute.

Currently there are industry products that utilize FL (or distributed training) to train large language models (LLMs), including Deepspeed ZeRO [74, 90], HuggingFace Accelerate [30], Pytorch Lightning Fabric [2], and FedLLM [42]. FL can facilitate LLM training due to the following reasons:

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

Figure 1: Byzantine attack [15, 21] configuration.

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

Figure 2: m-Krum [8] configuration.

i) *Distributed nature of LLM training data:* LLMs are pretrained using large amount of data, which often reside in different locations. Collecting such data to a central server is 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 [9], have an extremely large number of parameters. Training LLMs on a single machine is infeasible and inflexible, while federated learning (or training in a distributed manner) can be a good choice. *iii*) *Continuous improvement with user data:* LLMs can be deployed in a federated manner, while local instances of the models can be further finetuned based on the local data. Instead of transferring local data, only the model updates are transferred back to the central server, enabling the global model to improve over time based on users' data without ever having direct access to that data. This is particularly relevant for LLM applications in sensitive fields such as healthcare or personal communications, where data privacy is a major concern.

While FL can facilitate machine learning over distributed data without sharing client data with others, its decentralized and collaborative nature might inadvertently introduce privacy and security vulnerabilities. In recent years, a burgeoning body of research has spotlighted various attack mechanisms in FL [6, 95, 54, 46, 88, 15, 21, 87, 101, 3, 49, 100], where adversarial clients might submit spurious models to disrupt the global model from converging, or sabotage the global model to misidentify particular data samples by planting backdoors. Meanwhile, a wide range of defense mechanisms has emerged to mitigate the impact of these attacks [60, 52, 86, 71, 8, 96, 15, 86, 48, 99, 72, 25, 94, 99, 66, 51, 13]. Despite the expansive landscape of attacks and defenses in FL, there is a notable absence of benchmarks for evaluating the baseline strategies. Moreover, no research to date has explored the connection between FL and LLMs. These motivate a need for a standardized and comprehensive benchmark to assess baseline attack and defense mechanisms in the context of FL and LLMs.

This paper introduces FedMLSecurity, an FL security module of FedML [33]. FedMLSecurity comprises two primary components: FedMLAttacker and FedMLDefender. FedMLAttacker simulates attacks in FL to help understand and prepare for potential security risks, while FedMLDefender is equipped with various defense mechanisms to counteract the threats injected by FedMLAttacker. We also apply FedMLSecurity to FedLLM [42], an open-sourced library that supports training LLMs using geographically distributed GPUs[2]. Our contributions are summarized as follows:

***i*) Enabling benchmarking of various attacks and defenses in FL**. The attacks supported include: Byzantine attacks of random/zero/flipping modes [15, 21], label flipping backdoor attack [87], deep leakage gradient [101], and model replacement backdoor attack [3]. The defense mechanisms supported include: Norm Clipping [86], Robust Learning Rate [71], Krum (and m-Krum) [8], SLSGD [96], geometric median [15], weak DP [86], CClip [48], coordinate-wise median [99], RFA [72], Foolsgold [25], CRFL [94], and coordinate-wise trimmed mean [99]. The example configurations for attacks and defenses are in [43] and [44], respectively.

***ii*) Flexible configuration.** FedMLSecurity supports configurations using .yaml file. Users can utilize two parameters, "enable_attack" and "enable_defense", to activate FedMLAttacker and FedMLDefender. Sample configurations are shown in Figures 1 and Figures 2 , respectively.

[2]The focus of this benchmark is security instead of privacy, thus we do not include DP-related baselines. However, we would like to point out that FedML has a separate DP library, FedMLDP, to address privacy-related concerns in FL; see [40].

***iii*) Supporting customization of attack and defense mechanisms.** We provide APIs in FedMLSecurity to enable users to integrate user-defined attacks and defenses.

***iv*) Supporting various models and FL optimizers.** FedMLSecurity can be utilized with a wide range of models, including Logistic Regression, LeNet [57], ResNet [35], CNN [56], RNN [81], GAN [28], etc. It is compatible with various FL optimizers, such as FedAVG [69], FedOPT [78], FedPROX [59], FedGKT [32], FedGAN [77], FedNAS [34], FedNOVA [93], etc.

***v*) Extensions to LLMs and real-world applications.** Since FedML is a mature industry product that serves customers from different fields, FedMLSecurity can be integrated to real-world applications. We experimentally show the adaptability of FedMLSecurity to LLMs and real-world applications.

2 Preliminaries and Overview

In this section, we first present related works of FedMLSecurity, then introduce adversarial models in FedMLSecurity, and finally overview FedMLSecurity.

2.1 Related Works

Recent years, various benchmarks have been introduced for FL, such as TensorFlow Federated [1], PySyft [103], FATE [62], Flower [5], FedScale [53], NVIDIA FLARE [80], OpenFL [79], FedBioMed [85], IBM Federated Learning [63], FederatedScope [97], and FLUTE [19]. However, only FederatedScope delves into the implications of adversarial attacks in FL, with a focus on data reconstruction attacks that utilize models or gradients to revert sensitive information, including GAN-based leakage attack [37], Passive Property Inference [70], and DLG attack [101]. However, it neglects to address attacks prevalent in the research literature, *i.e.*, Byzantine attacks [99, 98]. It also does not include any defense mechanisms for FL. It is worth noting that, while FederatedScope integrates secret-sharing [4], it does not implement this within FL, placing it more within the scope of federated analytics [20, 75, 89, 47] instead of FL.

FedMLSecurity implements attacks that are widely considered in the literature [99, 87, 101]; it also integrates a wide range of defense mechanisms [86, 71, 8, 96, 15, 86, 48, 99, 72, 25, 94, 99]. Designed with flexibility in mind, FedMLSecurity offers configurable settings and APIs, enabling users to customize their attack and defense mechanisms.

2.2 Adversarial Model

Real-world adversaries in FL systems fall 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 [15, 21]), or subtly misclassifying a specific set of samples to minimally impact the overall performance of the global model (*e.g.*, backdoor attacks [3, 91]). Active adversaries can take various forms, including: 1) malicious clients who manipulate their local models [3, 15, 21] or submit contrived models without actual training [92]; 2) a global "sybil" [87, 25] 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 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 local training data) from revealed models (gradients, or model updates) [101]. 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

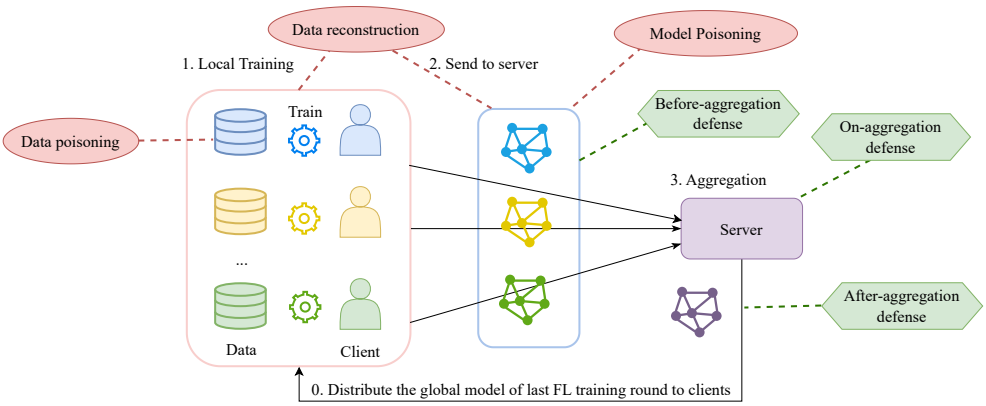

Figure 3: FedMLSecurity overview. FedMLSecurity enables injecting attacks (shown in red) and defenses (shown in green) at various stages of FL training at the clients and at the server.

the server; and 3) external adversaries, *e.g.*, hackers, that access communication channels to acquire local and global models transferred between clients and the FL server.

The adversaries can inject attacks at different stages of FL training. In summary, active adversaries can conduct attacks that modify local models (*model poisoning attacks*) or poison local datasets (*data poisoning attack*), while passive adversaries can infer sensitive information, such as user data, based on the models or gradients they observe, *i.e.*, data reconstruction attack. In the next subsection, we will illustrate how to inject those attacks at different stages of FL frameworks.

2.3 Overview of FedMLSecurity

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. To inject attacks and defenses, FedMLSecurity utilizes FedMLAttacker and FedMLDefender to initiate two instances to simulate attacks and defenses, respectively. The two instances are initialized once and can be accessible by other FedML objects[3].

Injection of attacks. Existing literature categorizes attacks in FL into various types, such as byzantine attacks [15, 21], backdoor attacks [3, 91], model poisoning attacks [21, 84, 6], data reconstructing attacks [101], data poisoning attacks [87], inference attacks [65], etc. Without loss of generality, we classify those attacks into the following three categories based on the targets of the attacks:

i) *Data poisoning attacks* that are conducted by active adversaries to modify clients' local datasets and are injected at clients [87, 17].

ii) *Model poisoning attacks* that are also conducted by active adversaries to temper with local models submitted by clients. To account for all cases of model poisoning attacks as discussed in §2.2, such attacks are injected 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.

iii) *Data reconstruction attacks* that are conducted by passive adversaries by exploring local models or updates to infer information about the training data. To cover the scenarios of such attacks as discussed in §2.2, FedMLAttacker injects such attacks at the FL server, as the FL server has access to all local models and the global model of each iteration, and can perform the attacks with flexibility.

Injection of defenses. FedMLDefender integrates defenses to mitigate, if not wholly nullify, the impacts of the injected attacks. Recognizing that the defenses either address issues related to tampered local models by posioned datasets[4] or prevent adversaries from deducing information from

[3]Such design is achieved by the singleton design pattern [26].

[4]Note that poisoning local datasets also results in tampered local models.

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 FL aggregation, including:

i) *before-aggregation functions* that modify local models submitted by clients;

ii) *on-aggregation functions* that modify the FL aggregation function to mitigate the impacts of local models submitted by adversarial clients; and

iii) *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.

Figure 3 summarizes the injections of attacks and defenses to the FL framework. We also provide detailed algorithms of injecting attacks and defenses to different stages of FL training, as shown in Algorithm 1 (for server aggregation) and Algorithm 2 (for client training) in the appendix. Below, we explain the implementations of attacks and defenses in detail.

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 modify the local models submitted by clients. FedMLAttacker injects such attacks before FL aggregation in each iteration, effecting modifying each local model directly. Model poisoning attacks implemented in FedMLAttacker include Byzantine attacks [15, 21] of three different modes and Model Replacement Backdoor attack [3]. 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, including:

- *Zero mode* that poisons the client models by setting their weights to zero.
- *Random mode* that manipulates client models by attributing random values to model weights.
- *Flipping mode* that updates the global model in the opposite direction by formulating a poisoned local model based on the global model $\mathbf{w}_g$ and the real local model $\mathbf{w}_\ell$ as $\mathbf{w}_g + (\mathbf{w}_g - \mathbf{w}_\ell)$.

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

- $poison\_model(local\_models, auxiliary\_info)$: 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 client models. The input $auxiliary\_info$ is any information used in the defense, *e.g.*, the global model in the last FL iteration.
- $is\_model\_poisoning\_attack()$: This function checks whether the attack component is activated and whether the attack modifies local models.

3.2 Data poisoning attacks.

Data poisoning attacks modify (or, poison) local datasets of some clients to achieve some malicious goals, such as degrading the performance of the global model or inducing the global model to misclassify some samples. As an example, in label flipping attack [87], 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) to revert training data. For example, Deep Leakage from Gradients (DLG) attack [101] infers local training data from the publicly shared gradients. 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 "model update" between models of different FL training rounds to deduce the training data. The adversary initializes dummy data and labels using a normal distribution $\mathcal{N}(0, 1)$ and calculates dummy gradients with the shared gradients while updating 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 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 in $fedml/core/security/attack$ 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, *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 [8] as examples.

Krum. Krum [8] tolerate f Byzantine clients among n clients by retaining only one local model that is most likely to be benign as the global model. Before aggregation, Krum computes a score for each client model using the sum of squared Euclidean distances between the model and its $n - f - 2$ "nearest" client models, and selects the local model with the minimum score as the global model of the current FL iteration. An optimization of Krum is m-Krum [8] 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. Compared with Krum, m-Krum allows more clients to

participate in FL training, which improves the quality of the global model. However, the selection of m requires the knowledge of the (maximum possible) number of Byzantine clients, which can be a limitation, as the number of Byzantine clients may not always be available in real-world systems.

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 FedMLDefender is activated and whether the defense requires injecting functions before aggregating local models at the 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. As an example, RFA (Robust Federated Aggregation) [72] 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. In practice, the geometric median is calculated using the Smoothed Weiszfeld Algorithm [72].

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 [94] 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 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 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 [10] (See Algorithm 1 in [10] for details).

Dataset	CIFAR10 [50]	CIFAR100 [50]	FEMNIST [12]	Shakespeare [67]	20News [55]	PubMedQA [64]
Model	ResNet20 [36]	ResNet56 [36]	CNN [68]	RNN (bi-LSTM) [68]	BERT [18]	Pythia-1B [7]

Table 1: Models and datasets for evaluations.

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 Experimental Evaluations

This section presents a comprehensive evaluation of how FedMLSecurity facilitates benchmarking attack and defense mechanisms in FL. The runnable codes for the experiments are available at [39].

Experimental setting. A summary of datasets and models for evaluations can be found in Table 1. 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 closely captures real-world scenarios. We further extend our evaluations to i.i.d. cases and various other models and datasets. For evaluations on LLMs, we utilize FedLLM [42] that trains LLMs in a federated manner. We employ the Pythia-1B model [7] and PubMedQA [45], 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. We utilize FedAVG in our experiments. Evaluations are conducted on a server with 8 NVIDIA A100-SXM4-80GB GPUs. We also included an experiment that utilizes real-world edge devices in Theta network [41] to showcase the scalability of FedMLSecurity to real-world applications; see Appendix B.

5.1 Evaluations on FL

By default, we use 10 clients for FL training and set the percentage of malicious clients to 10% when injecting attacks. We employ three attack mechanisms, including 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. We utilize three defense mechanisms: m-Krum [8], Foolsgold [25], and RFA [72]. For m-Krum, we set m to 5, which means 5 out of 10 submitted local models participate in aggregation in each FL training round. By default, the results are evaluated with the accuracy of the global model.

Exp1: Attack Comparisons. This experiment evaluates the impact of various attacks on test accuracy, using a no-attack scenario as our baseline. As illustrated in Figure 4, Byzantine attacks, specifically in the random and zero modes, substantially degrade accuracy. In contrast, the label flipping attack and the flipping mode of 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 strongest impact compared with other three attacks.

Exp2: Defense Comparisons. This experiment investigates the 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 5, 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 degrade in accuracy.

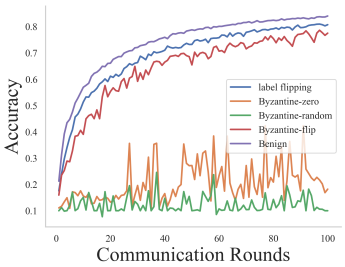

Figure 4: Attack comparison.

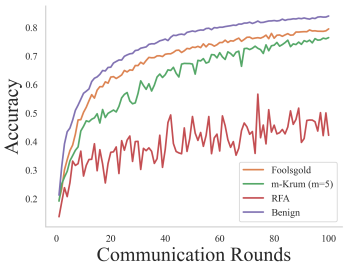

Figure 5: Defense comparison.

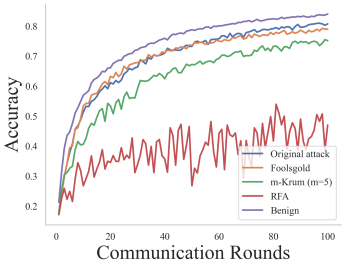

Figure 6: Defense against the label flipping attack.

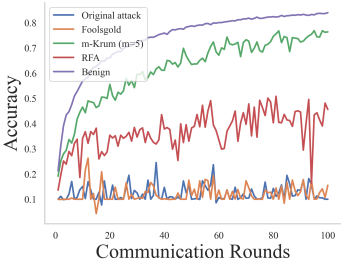

Figure 7: Defense against a random Byzantine attack.

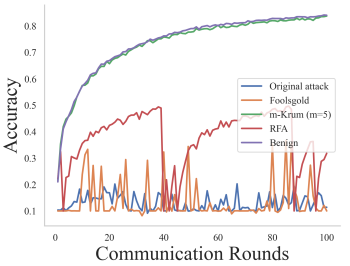

Figure 8: I.I.D. data evaluations.

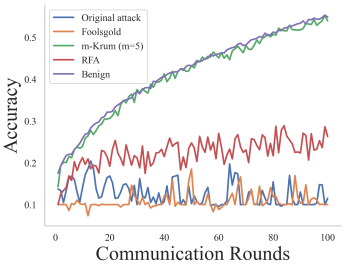

Figure 9: Scale # clients to 100.

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 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 4. Results for the label flipping and the random mode of Byzantine attacks are in Figure 6 and Figure 7, respectively. These results indicate that the defenses may contribute to minor improvements in accuracy for low-impact attacks, *e.g.*, Foolsgold in Figure 6. In certain cases, it is noteworthy that the defensive mechanisms may inadvertently compromise accuracy, such as the case with RFA in Figure 6. 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 7.

Exp 4: Evaluations on i.i.d. data. This experiment evaluates various defense mechanisms against an attack 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 8, 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. This experiment scales 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 9, 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.

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 10, 11, and 12, 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 11, and Foolsgold fails CNN in Figure 12. 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 an unchanged test accuracy in later FL iterations.

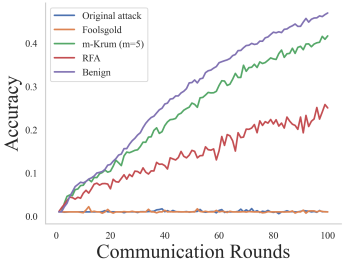

Figure 10: ResNet56 (CV).

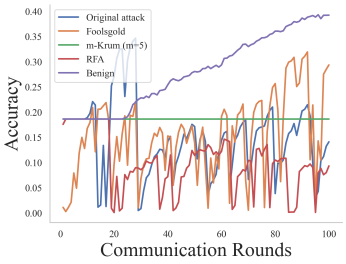

Figure 11: RNN (NLP).

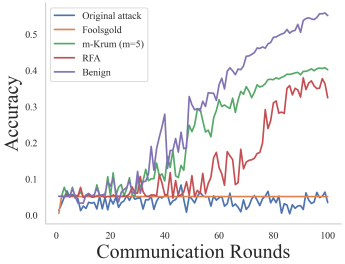

Figure 12: CNN (CV).

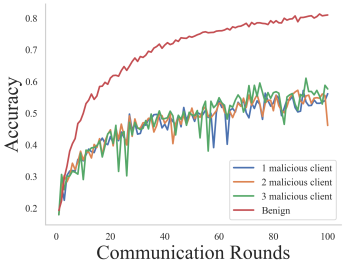

Figure 13: Varying # adversaries.

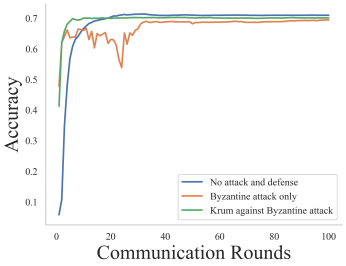

Figure 14: BERT evaluations.

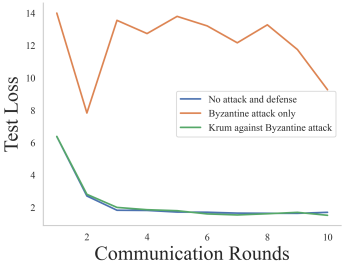

Figure 15: Pythia-1b evaluations.

Exp 7: Varying the number of malicious clients. This experiment evaluates the impact of varying number 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 13, 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 LLMs

We employ two LLMs, BERT [18] and Pythia [7], to showcase the scalability of FedMLSecurity and its applicability to federated LLM scenarios. We notice that some defenses (*e.g.*, Foolsgold [25]) 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.

Exp8: Evaluations of Krum against model replacement backdoor attack on BERT. This experiment utilizes BERT [18] and the 20 news dataset [55] for a classification task. We employ 10 clients and set 1 client to be malicious in each FL training round. We set the 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 14 show that m-Krum effectively mitigates the adversarial effect, bringing it closer to the level in the attack-free case, indicating the effectiveness of the m-Krum defense.

Exp9: 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 based on the test loss. Results in Figure 15 show that Byzantine attack significantly increases the test loss during training. Nevertheless, m-Krum effectively mitigates the adversarial effect.

6 Conclusion

This paper presents FedMLSecurity, an integrated module within FedML [33] designed to simulate potential adversarial attacks and corresponding defense strategies in FL. FedMLSecurity contains two components: FedMLAttacker that simulates various attacks that can be injected during FL training, and FedMLDefender, which facilitates defense strategies to mitigate the impacts of these attacks. 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] Luca Antiga. 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. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[4] Amos Beimel. Secret-sharing schemes: A survey. In *International conference on coding and cryptology*, pages 11–46. Springer, 2011.

[5] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[6] 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.

[7] 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.

[8] 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.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[10] 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. In *IEEE International Conference on Decentralized Applications and Infrastructures (IEEE DAPPS 2023)*, July 2023.

[11] 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.

[12] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[13] Jiahui Chen, Yi Zhao, Qi Li, Xuewei Feng, and Ke Xu. Feddef: Defense against gradient leakage in federated learning-based network intrusion detection systems. *IEEE Transactions on Information Forensics and Security*, 18:4561–4576, 2022.

[14] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*, 2019.

[15] 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.

[16] 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.

[17] 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.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] 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.

[20] Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Shanshan Han, Shantanu Sharma, Chaoyang He, Sharad Mehrotra, Salman Avestimehr, et al. Federated analytics: A survey. *APSIPA Transactions on Signal and Information Processing*, 12(1), 2023.

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

[22] Wikimedia Foundation. Wikimedia downloads.

[23] Shuhao Fu, Chulin Xie, Bo Li, and Qifeng Chen. Attack-resistant federated learning with residual-based reweighting. *arXiv preprint arXiv:1912.11464*, 2019.

[24] C. Fung, C. J. M. Yoon, and I. Beschastnikh. Mitigating sybils in federated learning poisoning. Aug 2018. Available on arXiv:1808.04866.

[25] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *RAID*, pages 301–316, 2020.

[26] Erich Gamma, Richard Helm, Ralph Johnson, Ralph E Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[27] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020.

[28] 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.

[29] 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.

[30] Sylvain Gugger. Introducing hugging face accelerate. https://huggingface.co/blog/accelerate-library, 2021.

[31] 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.

[32] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. *Advances in Neural Information Processing Systems*, 33:14068–14080, 2020.

[33] 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.

[34] Chaoyang He, Erum Mushtaq, Jie Ding, and Salman Avestimehr. Fednas: Federated deep learning via neural architecture search. 2021.

[35] 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.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 603–618, 2017.

[38] FedML Inc. Code of conduct. https://github.com/FedML-AI/FedML/blob/master/CODE_OF_CONDUCT.md, 2022.

[39] FedML Inc. FedMLSecurity experiments. https://github.com/FedML-AI/FedML/tree/master/python/examples/security/fedMLSecurity_experiments, 2022.

[40] FedML Inc. Differential privacy in FedML. https://github.com/FedML-AI/FedML/tree/master/python/fedml/core/dp, 2023.

[41] FedML Inc. FedML & Theta launch a decentralized ai supercluster for generative ai and content recommendation. https://blog.fedml.ai/fedml-theta-launch-a-decentralized-ai-supercluster-for-generative-ai-and-content-recommendation, 2023.

[42] FedML Inc. 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, 2023.

[43] FedML Inc. Sample configurations for attacks. https://github.com/FedML-AI/FedML/tree/master/python/examples/security/mqtt_s3_fedavg_attack_mnist_lr_example, 2023.

[44] FedML Inc. Sample configurations for defenses. https://github.com/FedML-AI/FedML/tree/master/python/examples/security/mqtt_s3_fedavg_defense_mnist_lr_example, 2023.

[45] 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.

[46] Xiao Jin, Pin-Yu Chen, Chia-Yi Hsu, Chia-Mu Yu, and Tianyi Chen. Cafe: Catastrophic data leakage in vertical federated learning. *Advances in Neural Information Processing Systems*, 34:994–1006, 2021.

[47] Gueyoung Jung, Nathan Gnanasambandam, and Tridib Mukherjee. Synchronous parallel processing of big-data analytics services to optimize performance in federated clouds. *2012 IEEE Fifth International Conference on Cloud Computing*, pages 811–818, 2012.

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

[49] Sanjay Kariyappa, Chuan Guo, Kiwan Maeng, Wenjie Xiong, G. Edward Suh, Moinuddin K. Qureshi, and Hsien-Hsin S. Lee. Cocktail party attack: Breaking aggregation-based privacy in federated learning using independent component analysis. In *International Conference on Machine Learning*, 2022.

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

[51] Abhishek Kumar, Vivek Khimani, Dimitris Chatzopoulos, and Pan Hui. Fedclean: A defense mechanism against parameter poisoning attacks in federated learning. *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4333–4337, 2022.

[52] Kavita Kumari, Phillip Rieger, Hossein Fereidooni, Murtuza Jadliwala, and Ahmad-Reza Sadeghi. Baybfed: Bayesian backdoor defense for federated learning. *arXiv preprint arXiv:2301.09508*, 2023.

[53] 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.

[54] Maximilian Lam, Gu-Yeon Wei, David Brooks, Vijay Janapa Reddi, and Michael Mitzenmacher. Gradient disaggregation: Breaking privacy in federated learning by reconstructing the user participant matrix. In *International Conference on Machine Learning*, pages 5959–5968. PMLR, 2021.

[55] Ken Lang. Newsweeder: Learning to filter netnews. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 331–339. Morgan Kaufmann, San Francisco (CA), 1995.

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

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

[58] 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.

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

[60] Xingyu Li, Zhe Qu, Shangqing Zhao, Bo Tang, Zhuo Lu, and Yao-Hong Liu. Lomar: A local defense against poisoning attack on federated learning. *IEEE Transactions on Dependable and Secure Computing*, 20:437–450, 2022.

[61] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[62] 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.

[63] 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.

[64] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. BioGPT: generative pre-trained transformer for biomedical text generation and mining. *Briefings in Bioinformatics*, 23(6), 09 2022. bbac409.

[65] 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.

[66] Zhuo Ma, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H. Deng. Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and Security*, 17:1639–1654, 2022.

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

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

[69] 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.

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

[71] 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.

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

[73] F. Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, May 1994.

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

[75] Daniel Ramage. Federated analytics: Collaborative data science without data collection. *Google AI Blog*, May 2020.

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

[77] Mohammad Rasouli, Tao Sun, and Ram Rajagopal. 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. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[79] 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.

[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. 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. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[82] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[83] Shakespeare. The complete works of william shakespeare by william shakespeare, Jan 1994.

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

[85] 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.

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

[87] 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.

[88] Richard Tomsett, Kevin Chan, and Supriyo Chakraborty. Model poisoning attacks against distributed machine learning systems. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pages 481–489. SPIE, 2019.

[89] Dan Wang, Siping Shi, Yifei Zhu, and Zhu Han. Federated analytics: Opportunities and challenges. *IEEE Network*, 36:151–158, 2022.

[90] Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. Zero++: Extremely efficient collective communication for giant model training. *arXiv preprint arXiv:2306.10209*, 2023.

[91] 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.

[92] Jianhua Wang. Pass: Parameters audit-based secure and fair federated learning scheme against free rider. *arXiv preprint arXiv:2207.07292*, 2022.

[93] 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.

[94] 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.

[95] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International conference on learning representations*, 2019.

[96] 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.

[97] 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.

[98] 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.

[99] 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.

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

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

[102] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[103] 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.

Appendix

A Algorithms for FL Server Aggregation and Client Training

The algorithms for injecting attacks and defenses in FL training are described in Algorithm 1 (for FL server aggregation) and Algorithm 2 (for client training).

Algorithm 1: Server Aggregation

Inputs: $\mathbf{w}'_g$: the global model of last FL training round; $\mathcal{W}_l$: the list of local models submitted by each client in the current FL training round.
Variables: $\mathcal{A}$: A FedMLAttacker instance initialized based on the FL configuration file; $\mathcal{D}$: A FedMLDefender instance that is initialized based on the FL configuration file.

Function $server\_aggregation(\mathcal{W}_l)$ **begin**
  $\mathcal{W}_l \leftarrow before\_aggregation\_process(\mathcal{W}_l, \mathbf{w}'_g)$
  $\mathbf{w}_g \leftarrow before\_aggregation\_process(\mathcal{W}_l, \mathbf{w}'_g)$
  return $after\_aggregation\_process(\mathcal{W}_l, \mathbf{w}_g)$
Function $before\_aggregation\_process(\mathcal{W}_l, \mathbf{w}'_g)$ **begin**
  if $\mathcal{A}.is\_attack\_enabled()$ **then**
    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** $\mathcal{W}_l \leftarrow \mathcal{A}.poison\_model(\mathcal{W}_l, \mathbf{w}'_g)$;
  if $\mathcal{D}.is\_defense\_enabled()$ & $\mathcal{D}.is\_defense\_before\_aggregation()$ **then**
    $\mathcal{W}_l \leftarrow \mathcal{D}.defend\_before\_aggregation(\mathcal{W}_l, \mathbf{w}'_g)$
  return $\mathcal{W}_i$
Function $on\_aggregation\_process(\mathcal{W}_l, \mathbf{w}_g)$ **begin**
  if $\mathcal{D}.is\_defense\_enabled()$ & $\mathcal{D}.is\_defense\_on\_aggregation()$ **then**
    return $\mathcal{D}.defend\_on\_aggregation(\mathcal{W}_l, \mathbf{w}_g)$
  return $aggregate(\mathcal{W}_i)$
Function $after\_aggregation\_process(\mathbf{w}_g)$ **begin**
  if $\mathcal{D}.is\_defense\_enabled()$ & $\mathcal{D}.is\_defense\_after\_aggregation()$ **then**
    return $\mathcal{D}.defend\_after\_aggregation(\mathbf{w}_g)$
  return $\mathbf{w}_g$

Algorithm 2: Client Training

Inputs: $dataset$: the local dataset of a client.
Variables: $\mathcal{A}$: A FedMLAttacker instance initialized based on the FL configuration file;

Function $client\_training(dataset)$ **begin**
  if $\mathcal{A}.is\_attack\_enabled()$ & $\mathcal{A}.is\_data\_poisoning\_attack()$ **then**
    $dataset \leftarrow \mathcal{A}.poison\_data(dataset)$
  $\mathbf{w}_l \leftarrow train(dataset)$
  $send\_to\_server(\mathbf{w}_l)$

B Supplementary Experiment

In this section, 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 [41], a customer of FedML, to validate the scalability of FedMLSecurity to real-world applications. The FedML client package is integrated into Theta's edge nodes, which periodically fetches

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

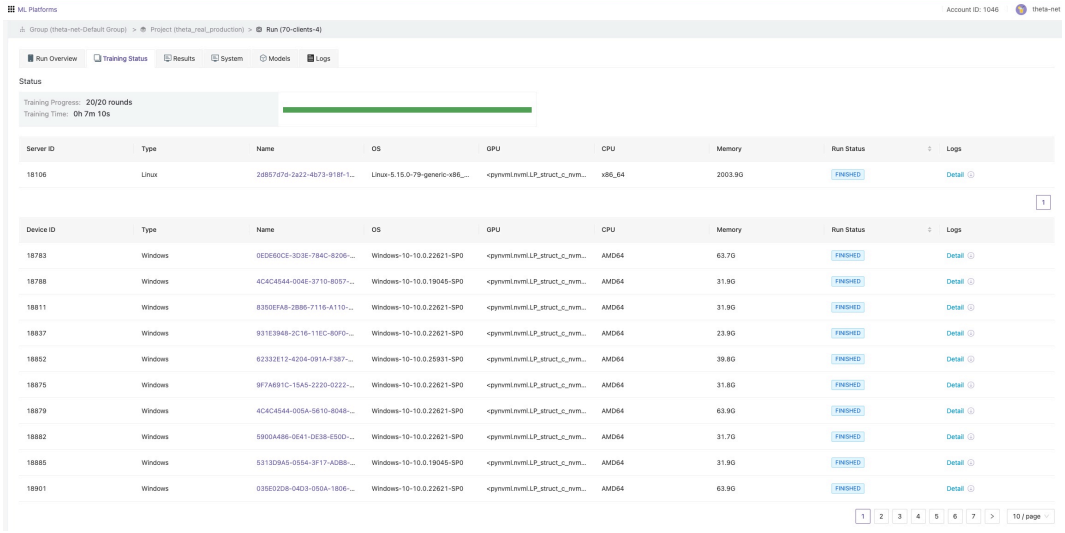

Figure 17: Real-world application: training status of devices.

data from the Theta back-end. Subsequently, the FedML 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), 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 18, 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 device.

C Code of Ethics, Limitations, and Potential Negative Social Impacts

FedMLSecurity, as well as FedML and FedLLM, is under the Apache 2.0 license, ensuring open access and customization. The code of conduct of the library is available at [38]. All datasets used

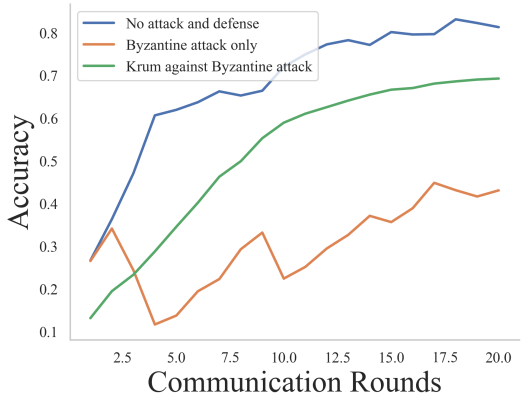

Figure 18: m-Krum against random-mode Byzantine attack in a real-world application.

for evaluations are publicly available, such as CIFAR10 [50], FEMNIST [12], Shakespeare [67], and so on. All models for evaluations are publicly available as well.

C.1 Code of Ethics

Data and Model Ownership. Any data uploaded by the users remains their property. FedML does not claim ownership of any user data. If users employ our FedML platform (https://open.fedml.ai/) for model training, the resulting models and data remain the users' intellectual property.

Data Handling and Protection. We are aware of the risks associated with data processing in FL settings. Users can use the open-sourced libraries (FedML, FedLLM, and FedMLSecurity) to simulate attacks and defenses on any machine without uploading their data and model. If users use the FedML MLOps platform for simulation, only the model weights are uploaded. The uploaded model weights are encrypted (i.e., only users with proper ownership can decrypt them) and can be deleted upon request. That is, any FedML product, including the FedMLSecurity benchmark we propose in this work, has no access to raw user data.

Benchmark Model Documentation and Transparency. We are committed to: i) providing comprehensive documentation on the functionalities of the benchmark; ii) making a detailed datasheet available for the benchmark model, outlining its specifications, capabilities, and intended use cases; and iii) offering transparent and well-documented APIs for users.

C.2 Limitations and Further Improvement

While FedMLSecurity offers a foundation for ML security research, we recognize its limitations and potential for further enhancement. Our plan for improvement includes the following aspects: 1) conducting more experiments on LLMs to provide a comprehensive understanding of vulnerabilities in LLMs within the FL context; and 2) designing and implementing advanced defense mechanisms against potential adversaries in asynchronous FL scenarios.

C.3 Potential Negative Social Impacts

Despite the fact that we put our best efforts in mitigating negative social impacts, the proposed FedMLSecurity benchmark might still be subject to some indistinct negative social impact, including:

- **Potential misuse**: While our module simulates attacks and defenses in FL to help the communities to better understand and compare the attacks in FL, it is not immune to malicious use. The platform could potentially be used to exploit vulnerabilities or develop advanced attack techniques in FL systems.

- **Data security**: FL is susceptible to various threats such as data poisoning. We acknowledge these inherent risks and are actively working on introducing defenses mechanisms to mitigate such attacks.
- **Privacy Concerns**: Although FL aims to train models without sharing raw data, there remains a risk of indirect data leakage, for example, attackers might utilize the models to infer whether specific data points are in the training datasets, where users should be cautious and informed.

C.4 Model Cards

Model Card: ResNet

- **Model Details**.
  - Person or organization developing model: Microsoft Research
  - Model date: 2015.
  - Model version: ResNet-20, ResNet-56
  - Model type: Convolutional neural network
  - Information about training algorithms, parameters, fairness constraints or other applied approaches, and features: the original model is trained on ImageNet [82] with SGD.
  - Paper or other resource for more information: "Deep Residual Learning for Image Recognition" [36].
  - Citation details: He et al. "Deep Residual Learning for Image Recognition", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016 [36].
  - License: our implementation is under Apache 2.0.
- **Intended Use**.
  - Primary intended uses: Image classification, object detection, and other computer vision tasks.
  - Primary intended users: Machine learning and computer vision researchers and engineers.
  - Out-of-scope use cases: Non-computer vision tasks such as audio processing.
- **Factors**.
  - Image resolution, color distribution, and object scales.
- **Metrics**.
  - Model performance measures: Top-1 and Top-5 accuracy.
- **Evaluation Data**.
  - ResNet-20: CIFAR-10 [50] test data split.
  - ResNet-56: CIFAR-100 [50] test data split.
- **Training Data**.
  - ResNet-20: CIFAR-10 [50] training data split.
  - ResNet-56: CIFAR-100 [50] training data split.
- **Ethical Considerations**:
- **Ethical Considerations**: The model is trained on CIFAR-10 and CIFAR-100 which could contain biased information; this might lead to unfair or discriminatory model behavior.
- **Caveats and Recommendations**: This model is trained with low resolution images which is not suited for recognition tasks that requires high resolution input such as medical imaging.

Figure 19: Model card for ResNet.

Model Card: LSTM from FedAVG [69]

- **Model Details**.
  - Person or organization developing model: McMahan et al. from Google introduced this model in [69].
  - Model date: 2016.
  - Model type: Recurrent Neural Network.
  - Information about training algorithms, parameters, fairness constraints or other applied approaches, and features: The original model was trained with FedAvg and local SGD on the Shakespeare dataset. We only used the model architecture in this work.
  - Paper or other resource for more information: "Communication-Efficient Learning of Deep Networks from Decentralized Data" [69].
  - Citation details: McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data" Artificial intelligence and statistics. PMLR, 2017 [69].
  - License: The original implementation was not released. Our implementation is under Apache 2.0.
  - Where to send questions or comments about the model: Open an issue in FedML repo github.com/FedML-AI/FedML/issues.
- **Intended Use**.
  - Primary intended uses: Sequence-to-sequence tasks such as time series prediction, natural language processing, and speech recognition.
  - Primary intended users: Machine learning researchers and engineers.
  - Out-of-scope use cases: Non-sequential data.
- **Factors**.
  - The model is trained on Shakespeare dataset [69] which is constructed from William Shakespeare's plays [83]. Due to its historical context, the text may contain sexist and other form of discriminatory content.
- **Metrics**.
  - Next word prediction accuracy.
- **Evaluation Data**.
  - Shakespeare [69] test data split.
- **Training Data**.
  - Shakespeare [69] training data split.
- **Ethical Considerations**: Due to the historical context of Shakespeare's works, the model output may reflect biases and outdated perspectives.
- **Caveats and Recommendations**: The generated text may look correct but is not actually from Shakespeare's works. The user may need to verify the correctness of the model output.

Figure 20: Model card for LSTM from FedAVG [69].

Model Card: CNN from FedOPT [78]

- **Model Details**.
  - Person or organization developing model: Reddi et al. from Google introduced this model in [78].
  - Model date: 2020.
  - Model type: Convolutional Neural Network.
  - Information about training algorithms, parameters, fairness constraints or other applied approaches, and features: The original model was trained with FedOPT and local SGD on the CIFAR-10 dataset [50]. We only used the model architecture in this work.
  - Paper or other resource for more information: "Adaptive Federated Optimization" [78].
  - Citation details: Reddi et al. "Adaptive Federated Optimization" International Conference on Learning Representations, 2021 [78].
  - License: The original implementation was not released. Our implementation is under Apache 2.0.
  - Where to send questions or comments about the model: Open an issue in FedML repo github.com/FedML-AI/FedML/issues.
- **Intended Use**.
  - Primary intended uses: Image recognition for handwritten characters.
  - Primary intended users: Machine learning researchers and engineers.
  - Out-of-scope use cases: Non-visual data. Non-character recognition tasks.
- **Factors**.
  - Image resolution, color distribution, and object scales
- **Metrics**.
  - Classification accuracy
- **Evaluation Data**.
  - FEMNIST [12] test data split.
- **Training Data**.
  - FEMNIST [12] training data split.
- **Ethical Considerations**: The model is trained on FEMNIST which contains handwritten characters, which does not pose significant ethical concern.
- **Caveats and Recommendations**: This model is trained with low resolution images of handwritten characters which is not suited for other form of recognition tasks such as general object recognition.

Figure 21: Model card for CNN from FedOPT [78].

Model Card: BERT

- **Model Details**.
  - Person or organization developing model: Devlin et al. from Google AI introduced this model in [18].
  - Model date: 2018.
  - Model version: BERT-Base.
  - Model type: Encoder-only Transformer-based architecture.
  - Information about training algorithms, parameters, fairness constraints or other applied approaches, and features: Trained using masked language model and next sentence prediction tasks. The pretraining is done on BooksCorpus [102] and English Wikipedia [22].
  - Paper or other resource for more information: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [18].
  - Citation details: Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2018 [18].
  - License: Apache License 2.0.
  - Where to send questions or comments about the model: Open an issue in FedML repo github.com/FedML-AI/FedML/issues.
- **Intended Use**.
  - Primary intended uses: Natural language understanding tasks, including question answering, sentiment analysis, and named entity recognition.
  - Primary intended users: Machine learning and Natural Language Processing researchers and engineers.
  - Out-of-scope use cases: Tasks with non-textual data. Text generation.
- **Factors**.
  - Text length (due to fixed context length), language or dialect of text.
  - The model is trained on web-curated datasets and may contain racist, sexist, xenophobic, and other discriminatory content.
- **Metrics**.
  - Test accuracy.
- **Evaluation Data**.
  - Fine-tuning: 20 News [55] test split.
- **Training Data**.
  - Pre-training: BooksCorpus [102] and English Wikipedia [22].
  - Fine-tuning: 20 News [55] training split.
- **Ethical Considerations**: The model is trained on web-curated datasets which may contain racist, sexist, xenophobic, and other discriminatory content.
- **Caveats and Recommendations**: Bert is a base model intended for further fine-tuning. The users should also add moderation mechanisms before the deployment.

Figure 22: Model card for BERT.

Model Card: Pythia

- **Model Details**.
  - Person or organization developing model: EleutherAI.
  - Model date: 2023.
  - Model type: Decoder-only Transformer-based Language Model
  - Information about training algorithms, parameters, fairness constraints or other applied approaches, and features: Pythia models were trained on the Pile dataset [27]. The pile is a 825GiB general-purpose dataset in English. It contains texts from 22 diverse sources, roughly broken down into five categories: academic writing (e.g. arXiv), internet (e.g. CommonCrawl), prose (e.g. Project Gutenberg), dialogue (e.g. YouTube subtitles), and miscellaneous (e.g. GitHub, Enron Emails).
  - Paper or other resource for more information: "Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling" [7].
  - Citation details: Biderman et al. "Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling", International Conference on Machine Learning, 2023 [7].
  - License: Apache 2.0.
  - Where to send questions or comments about the model: Open an issue in the pythia repo github.com/EleutherAI/pythia/issues.
- **Intended Use**.
  - Primary intended uses: research on the behavior, functionality, and limitations of large language models. This suite is intended to provide a controlled setting for performing scientific experiments.
  - Primary intended users: Machine learning researchers and engineers.
  - Out-of-scope use cases: The Pythia Suite is a based model that requires fine-tuning and alignment before the deployment. Pythia models are English-language only, and are not suitable for translation or generating text in other languages.
- **Factors**.
  - The model is trained on the Pile dataset [27] which is web-curated and may contain racist, sexist, xenophobic, and other discriminatory content.
- **Metrics**.
  - Test loss, Lambada (OpenAI), PIQA, WinoGrande, WSC, ARC, ARC, SciQ, and LogiQA scores.
- **Evaluation Data**.
  - The Pile dataset [27] test data split.
- **Training Data**.
  - The Pile dataset [27] training data split.
- **Ethical Considerations**: The training dataset is web-curated and may contain discriminatory content. Since the pythia models are not aligned with human value, they may generate discriminatory and harmful content.
- **Caveats and Recommendations**: Pythia is a base model intented for further fine-tuning and alignment. The users should align the model with human value before the deployment.

Figure 23: Model card for Pythia.