

FedSlice: Protecting Federated Learning Models from Malicious Participants with Model Slicing

Abstract—Federated learning (FL) is a new crowdsourcing development paradigm for the DNN models, which is also called “software 2.0”. In practice, the privacy of FL can be compromised by many attacks, such as free-rider attacks, adversarial attacks, gradient leakage attacks, and inference attacks. Conventional defensive techniques have low efficiency because they deploy heavy encryption techniques or rely on TEE. To improve the efficiency of protecting FL from these attacks, this paper proposes FedSlice to prevent malicious participants from getting the whole server-side model while keeping the performance goal of FL. FedSlice breaks the server-side model into several slices and delivers one slice to each participant. Thus, a malicious participant can only get a subset of the server-side model, preventing them from effectively conducting effective attacks. We evaluate FedSlice against these attacks and results show that FedSlice provides effective defense: the server-side model leakage is reduced from 100% to 43.45%, the success rate of adversarial attacks is reduced from 100% to 11.66%, the average accuracy of membership inference is reduced from 71.91% to 51.58%, and the data leakage from shared gradients is reduced to the level of random guesses. Besides, FedSlice only introduces less than 2% accuracy loss and about 14% computation overhead. To the best of our knowledge, this is the first paper to discuss defense methods against these attacks to the FL framework.

I. INTRODUCTION

Deep learning models, also dubbed as “software 2.0”, are one of the major focusing points of software engineering researchers today [57], [44], [24], [26], [16], [36], [68], [62], [33], [68]. Federated learning (FL), which allows multiple participants to train a deep learning model collaboratively, is a new crowdsourcing strategy for deep learning models [50], [31], [70], [47]. Unlike traditional software crowdsourcing in which the key security concern is the integrity of the software under development, FL concerns not only the integrity of the model but also the privacy of participants. Although FL protects participant privacy by limiting the participant data to local devices, recent research showed that FL may still leak participant privacy [74], [72], [54], [50].

Possible Attacks. This paper focuses on four attacks against federated learning: free-rider attacks [41], [23], adversarial attacks [45], gradient leakage attacks [74], [72], and inference attacks [54], [50]. Free-rider attacks mean a malicious participant can steal the aggregated model without contributing to the training process. The stolen model harms the intellectual property of the task requester and causes economic loss. For adversarial attacks, malicious participants may use the shared model to generate adversarial samples to mislead the server-side model. In gradient leakage attacks, the private training data can be reconstructed by other participants. Inference

attacks can recover the sensitive information of other participants’ training data. These four attacks compromise the integrity of the model and the participant’s privacy. It is important to develop defensive techniques for these attacks.

Status Quo and Limitation. Researchers have noticed the threat of these four attacks and proposed various solutions. However, these solutions are hard to deploy in practice due to low efficiency. One solution is to encrypt model weights so that participants know nothing of the distributed model [49], [43], [10]. However, as reported in [60], cryptographic ML algorithms are more than $1,000\times$ slower than ordinary protocols. Another solution is to perform model updates inside the trusted execution environments (TEEs) [20], [60], [48]. However, TEEs are about $36\times$ slower than the untrusted hardware [60], [55]. Besides, TEEs require dedicated hardware support, which is not always available for federated devices [3], [1], [2], [4].

Existing techniques have low efficiency because they aim to protect the machine learning model from both the malicious server and participants simultaneously. Heavy encryption techniques or dedicated hardware, such as TEE, are necessary to defend against the malicious server because it is the dominator of the FL process and has the highest privilege over the model. For example, the server can see and manipulate the model updates from all participants. It is difficult to prevent the server from hacking the model in this scenario. However, both encryption and using TEE introduce high overhead and slow down the speed of FL models [60], [20], [60], [48].

Key Idea. This paper aims to efficiently protect FL models in a different but more practical scenario, in which the server is trustworthy but a few participants are malicious [47], [9], [67], [67], [67]. This scenario is more practical because, in many realistic FL applications, the server belongs to reputable companies or organizations with financial resources. Violating the participants’ privacy harms the reputation of these companies and may result in punishment with a high penalty. Thus, the server is less motivated to compromise the participants’ privacy and attack the trained model. On the contrary, participants are less reliable and are more likely to conduct attacks on FL. As FL needs many participants, it is hard to control the identification of participants. An adversary can pretend to be an honest participant, join the FL training process, and attack the trained model. In this case, malicious participants are less privileged than the server and have less control over the training process.

Specifically, this paper aims to efficiently protect FL models by achieving two sub-goals. First, we want to defend against the four types of attacks mentioned before. Second, we aim

to avoid using TEE and encryption because these techniques can slow down the training speed of an FL model by $36\times$ to $1000\times$. To achieve our goal, we build FedSlice, a new FL framework that defends against malicious participants when the server is trustworthy. To the best of our knowledge, FedSlice is the first efficient technique to defend against these four attacks without TEE and encryption.

Insight. The insight of FedSlice comes from the *discriminative model distribution* strategy, which does not share the whole model with all participants unconditionally. Instead, a participant only has access to the part of the model that is relevant to her so that she cannot infer privacy information about other participants nor compromise the integrity of the whole model. This strategy is fundamentally different from existing solutions [40], [46], [5], [9], [67], [13], which distribute a shared model indiscriminately to all participants. Since our approach avoids distributing a shared model to all participants, we can avoid using TEE or encryption techniques, allowing more efficient model protection.

Technical Challenges. Enabling the discriminative distribution strategy is particularly challenging because it requires aggregating heterogeneous models across the server and client sides. To ensure that each participant receives a model only relevant to his privacy, models distributed to different participants must be different. The server-side model also needs to be different from the participant models. However, existing FL techniques require a unified model between the server and all participants because they fuse the models by matching weights on the exact same positions in different models [46], [5], [40]. Such merging techniques cannot be applied to merge heterogeneous models because they cannot find matching weights. To achieve discriminative model distribution, we need to address two technical challenges: (1) we need to design an effective way to automatically generate heterogeneous models for a large number of participants; (2) we need to develop an efficient technique to effectively aggregate heterogeneous models.

Unfortunately, none of the existing techniques can address these two technical challenges to the best of our knowledge. (1) Neural architecture search [21] can generate heterogeneous models automatically, but the search process is data-intensive and time-consuming. Besides, the architectures of searched models are highly diversified, making the models difficult to combine later [25]. (2) Knowledge distillation [30] can transfer the knowledge between heterogeneous models, but the training time is unacceptable [18]. This technique requires training all updated models to aggregate the knowledge. When the number of participants increases to hundreds or thousands, a realistic scenario for FL applications, the distillation time exposes and becomes impractical [11].

Our Solution. FedSlice addresses the two challenges mentioned above with techniques based on model slicing, which was inspired by traditional program slicing. The high-level idea of model-slicing-based techniques is first to have a central model as the template to produce heterogeneous models and then aggregate the generated models back into the central

model. The proposed method addresses the first challenge because heterogeneous models can be generated by permuting and combining the basic layers of the template model. This model generation process does not need data and can be completed within five minutes. For the second challenge, as the heterogeneous models are generated from the same template model, existing FL aggregation techniques can combine the building layers to fuse the models back into the template model.

FedSlice can be integrated into current FL aggregation strategies, such as McMahan [46], Li [40], and Asad [5]. Therefore, FedSlice can be deployed into existing FL applications without modifying their fundamental models and aggregation strategy.

Evaluation. We conducted experiments on six representative tasks. The results show that FedSlice can effectively defend these four attacks with marginal accuracy loss and computation overhead. For the free-rider attacks, FedSlice decreases the model leakage from 100% to 43.45%. For adversarial attacks initiated from participants, FedSlice decreases the success rate from above 99% to 11.66%. For membership inference attacks, the attack accuracy is reduced from 71.91% to 51.58% (a random guess has 50% accuracy). For deep gradient leakage attacks, the mean squared error between the recovered data and the original training data is increased from 0.00013 to 1.52 (a white noise has 2.0 MSE). The average accuracy loss of FedSlice is smaller than 2%. The training time of FedSlice is 14% more than the unprotected model. Compared to model encryption (more than $1,000\times$ slower) and TEE-based techniques ($36.1\times$ slower), the cost of FedSlice is low.

To summarize, our paper makes the following contributions:

- We propose a slicing-based method that protects FL models against four existing attacks, which include free-rider attacks, adversarial attacks, membership inference attacks, and deep gradient leakage attacks.
- We implement our method as a prototype, FedSlice, that achieves $31.6\times$ and $877.2\times$ higher training speed compared to TEE and encryption-based methods.
- We conduct extensive experiments with six large-scale datasets against the four attacks and demonstrate the effectiveness and scalability of FedSlice to protect the functionality and privacy of the server-side model.

Data Availability: Our tool and data are available on the Internet ¹.

II. BACKGROUND

A. Security Risks of Federated Learning

Although FL ensures that data does not leave the local device, researchers still found that it suffers from various security issues.

Free-rider attacks are launched by malicious participants who want to steal the FL model without contributing to it [41], [23]. A free-rider can forge the locally-trained model

¹<https://anonymous.4open.science/r/FedSliceICSEartifact/README.md>

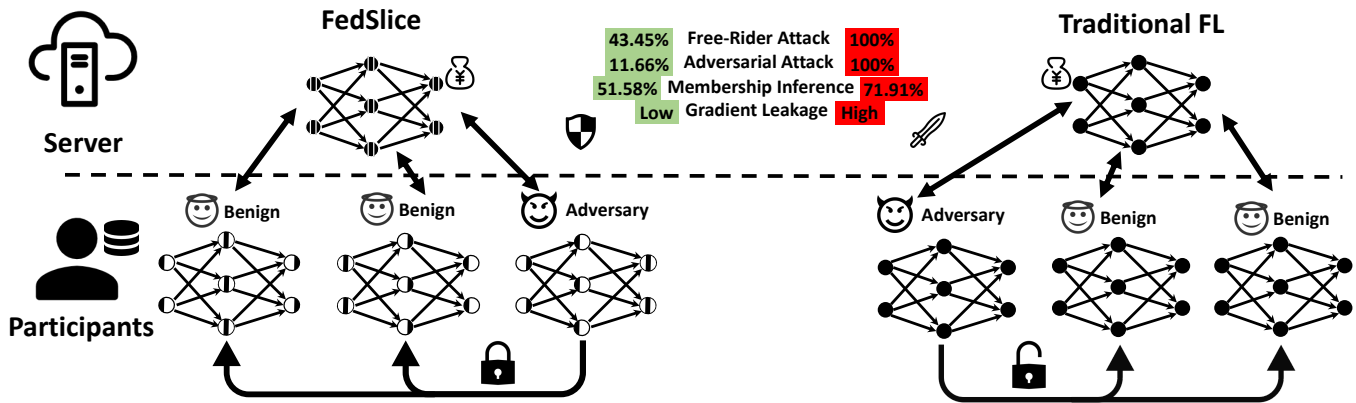


Fig. 1: The comparison between FedSlice and traditional federated learning. Traditional FL distributes the entire server-side model to all participants but FedSlice only distributes a (different) model slice to each participant.

by adding random noise to model weights. It is difficult for the server to stop free-riders because it distributes a monolithic model to the participants at each round. The free-rider can get the valuable model just by waiting for the server to deliver the model.

Adversarial attacks aim to fool the target deep learning system with carefully crafted input samples that look similar to standard samples in the human’s eyes. The adversarial attack is considered a severe security threat to DNN models, including models trained by federated learning [45], [14]. The malicious samples are usually generated from a surrogate model, and the attack success rate strongly depends on the similarity between the surrogate model and the target model. The attack is more effective if the attacker knows the target model and mimics its structure.

Inference attacks mainly consist of membership inference and attribute inference. Membership inference determines whether a data sample is used by other participants, and attribute inference outputs sensitive attributes of a given sample. A malicious participant can utilize the output score, loss value [54], or internal gradient [50] of the shared model to infer private information. Although membership inference and attribute inference goals are different, the techniques are similar, so we mainly discuss membership inference in this paper.

Deep gradient leakage means that the shared model updates can expose detailed information about the training data. Given the shared model, a malicious attacker can recover the ground truth label and input sample [74], [72]. The recovered sample is realistic (pixel-level accurate). Thus this vulnerability poses a significant threat to the private data of honest participants.

B. Motivation and Challenges

Inspiration. Figure 1 shows the high-level idea of FedSlice and its difference between conventional FL processes. Traditional FL unconditionally distributes the server-side model to all participants (as shown in the right part of Figure 1). The adversary participant receives the shared server-side model and can perform various attacks. Unlike conventional FL, FedSlice,

as shown in the left part of Figure 1, partitions the server-side model into several slices and distributes one slice to each participant. As the adversary participant only has partial information about the server-side model, he can not perform effective attacks.

The design of FedSlice is inspired by access control in operating system [61], [12]. Access control ensures that authorized users and applications can only do what is inside the permission and nothing more than that. This technique protects system resources (such as memory and files) and user applications to enforce confidentiality and integrity. Similarly, we want to restrict the accessible information to one participant by controlling the distributed model so that the malicious participant doesn’t know enough to perform attacks.

Technical Challenges. Designing FedSlice faces the following two challenges:

(1) *How to separate the server-side model into slices that participants can train on local devices.* The way of encoding knowledge and the explainability of the operating mechanism between DNNs and traditional software is different. For traditional software, developers manually specify the knowledge into code lines, each line having a specific meaning. Thus, a software program can be decomposed and recomposed into program slices [65] or slice combinations [27], [7]. On the contrary, the behavior of DNN is constructed through enormous annotated data, and each weight does not have a definite meaning. Although there are advanced neural network slicing, only small models can apply such techniques. It’s because they perform fine-grained weight analysis, and the introduced overhead is exponential to the amount of analyzed weights [52], [71].

(2) *How to effectively combine heterogeneous models from participants and reduce accuracy loss?* To discriminatively distribute models to different participants, the distributed models must be heterogeneous, *e.g.*, various participants receive different models. However, existing FL aggregation techniques cannot handle heterogeneous models and achieve high server-side model accuracy. Although there are techniques, such as model distillation [30], that may distill the knowledge of heterogeneous models into homogeneous models, they are

not efficient due to the large amount of data required. For example, distilling a model on the CIFAR10 dataset requires about 4,000 samples per class [6]. With this amount of data, The task requester can directly train a new model without FL.

C. Threat Model

Assumption. We consider a typical federated learning framework, which includes one server and multiple participants. The server wants to utilize participants’ data to train a central server model and award the participants with money. According to regulations, participants do not need to upload data and only share a locally-trained model. We assume that the server is trustworthy, but participants may be malicious. We also assume that malicious participants have all privileges to the model they have received. They can arbitrarily analyze or modify the received model. We assume that the server can leverage some **public** dataset to assist slice aggregation. For example, the server can train the aggregated model with the ImageNet dataset. However, the server does *not* have access to participants’ private data. This assumption is common in prior literatures [13], [38], [15], [73], [42]. In other words, the participant-side models should have inadequate functionalities compared to the server-side model and contain as little as possible private information about other participants’ training data.

III. APPROACH

This section will introduce the overview of FedSlice. After that, we will illustrate the detailed description of FedSlice.

A. Overview

FedSlice includes three steps: ensemble model construction, slice distribution, and slice aggregation. The first step is only performed once during the framework setup. The slice distribution and aggregation may be iterated multiple times until a desired performance or time limitation. Figure 2 depicts a sample demo of the overall pipeline. We briefly summarize each stage as follows:

Ensemble model construction is the setup phase of our framework. According to expert knowledge, the task requester must first select a “template model” (a pre-defined model architecture). For example, the task requester can choose a ResNet20 [29] model to perform image classification tasks. This stage enlarges each template model’s layer by duplicating one layer to several parallel “branches”. Such branches have the same operations but have different weights. Then, a fusion layer is added between layers to aggregate the output of different branches. The duplicated branches and the added fusion layer form the “ensemble model”. The task requester maintains this ensemble model until the end of the training phase. As shown in the upper left part of Figure 2, the template model has three layers. Each of the first two layers performs three operations, and the last layer performs two operations. FedSlice duplicates each layer into three branches and adds one fusion layer between layers.

Slice distribution constructs diverse “model slices” from the ensemble model by a slice-branch mapping rule. A slice of the ensemble model is a combination of branches and has the same structure as the template model. For each of the template model’s layers, a slice includes one of the ensemble model’s branches. The mapping rule defines which branches are used to construct the slices and is initialized at the setup phase. During the training phase, FedSlice distributes different slices to different participants. Participants use private data to train the slices and upload the slices to the server. The mapping rule is displayed in the upper right part of Figure 2, and five constructed slices are distributed to five participants.

Slice aggregation fuses the uploaded slices into the ensemble model. FedSlice first updates the branches with an inverse mapping rule. This inverse mapping is derived from the slice-branch mapping rule and defines which slices are used to update the branches. How to update the branches is defined by a given per-weight aggregation function. This function can be arbitrarily selected from the prior work [46], [40], [5], [39]. The lower part of Figure 2 shows the inverse mapping and how each branch is updated from the slices. After updating all the branches, FedSlice trains the ensemble model with the public dataset. This joint training aims to train the fusion layers and enhance the collaboration between branches.

Rationality. FedSlice can mitigate the threats in Section II-A meanwhile reducing the performance loss. First, the ensemble model can efficiently achieve high accuracy with little data because the participants train branches with private data. The server only needs to fine-tune the weights of the branches and the fusion layers. Second, in our design, each participant only receives a slice of the server-side model, and none of the participants know the ensemble model’s complete structure. The size of a slice is approximately $1/n$ of the whole model (n is the number of branches per layer). Thus, the confidentiality of the ensemble model is protected, and only the server owner has full access to the model. Third, each slice is composed of different branches. Thus slices are mutually heterogeneous. The probability of two participants receiving the same slice is $1/n^l$ (l is the number of layers). In the experimental setting, where l is four and n is ten, this probability is less than 10^{-6} . The chance for the malicious participant to receive the same model as another participant is little or no. Thus he can not perform an effective attack because the attacks require the integral model from other participants.

B. Ensemble Model Construction

In the beginning, the task requester first chooses a template model. The upper left corner of Figure 2 shows an example in which the template model contains three layers. Each of the first two layers performs three operations: convolution, batch normalize, and ReLU. The last layer performs two operations: pooling and linear. Ensemble model construction aims to construct an ensemble model from the template model. The task requester maintains the ensemble model and then slices it into different model slices. To construct the ensemble

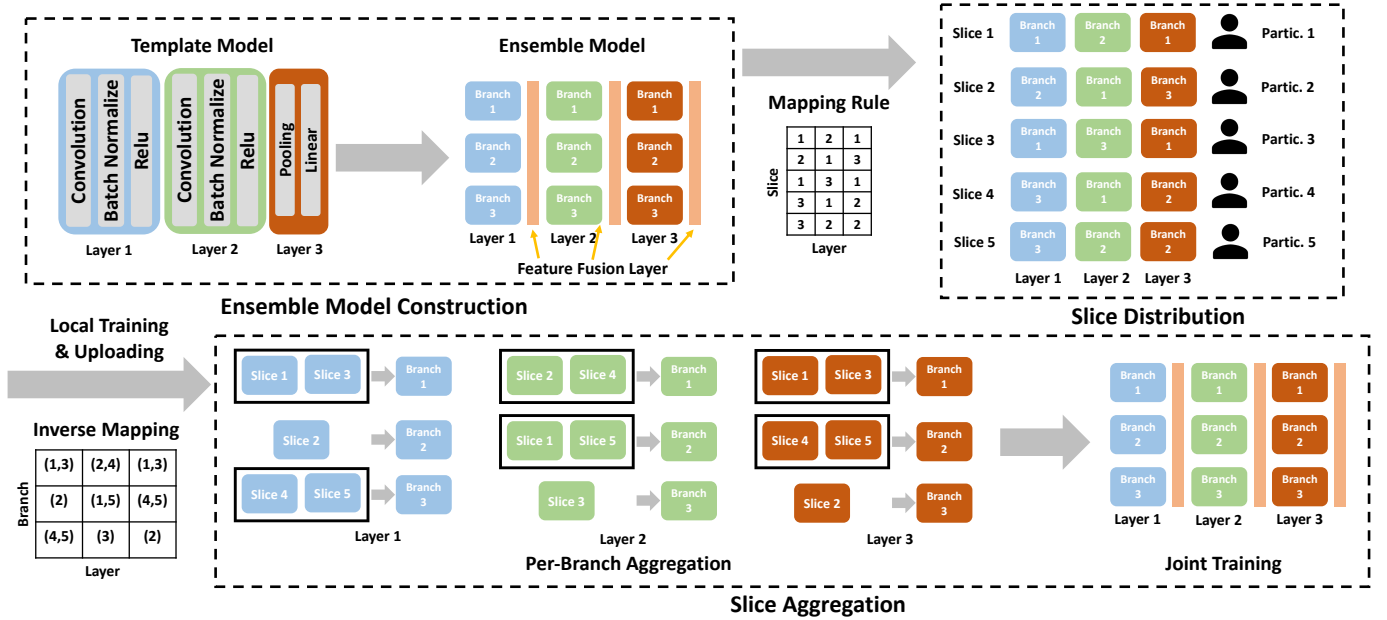


Fig. 2: The overall pipeline of FedSlice consists of three stages: ensemble model construction, slice distribution, and slice aggregation. In the demo, the template model contains three layers (denoted in blue, green, and red boxes) and each layer has three branches (denoted by the number in each box). “Partic.” is the abbreviation of “participant”.

model, the task requester builds different branches for each layer and adds a fusion layer between layers.

Branches Replication. We first build n parallel branches for each layer. The branches share the same architecture but have different weights. The server model uses all branches for inference, while participants only receive a subset of branches. Different branches learn knowledge from different participants. This knowledge is later integrated into the ensemble model. At the setup stage, branches are initialized by different weights. In the upper left part of Figure 2, FedSlice constructs three branches for each layer, and the squares show the branch IDs (branch 1, branch 2, e.t.c).

Fusion Layer. To fuse the information of branches, we append a feature fusion layer after each layer of the ensemble model. The fusion layers take the outputs of prior branches as input and produce the input for branches of the next layer. This design is similar to the requirement that different software components have a unified interface to facilitate the development process. Even though the function and implementation of different components vary, the interface consistency ensures that the components can be easily replaced and don’t need to change other modules.

The choice of the fusion layers should consider two aspects: the ease of training and the protection of model confidentiality. On the one hand, the server can not collect much data to support complex models, so the structure of fusion layers should be concise. On the other hand, a superficial fusion layer (such as feature average) may leak the server model because there are fewer server-specific parameters. We choose to apply batch normalization after the averaged feature maps of all branches (which we call “FeatBN”) as the fusion layer. Other choices include only averaging all feature maps

of prior branches (which we call “FeatAvg”) and applying convolution operation after the averaged feature maps (which we call “FeatConv”). FeatBN is the best choice among the three candidates to balance the two factors according to the experiment in Section IV-D.

C. Slice Distribution

In this stage, the task requester decomposes the ensemble model into **model slices** and distributes slices to participants according to a **mapping rule**. The input of this stage is the ensemble model from the pre-mentioned section, and the output is a set of model slices. This stage is the core component of FedSlice. In each communication round, slices are assembled following the mapping rule and are sent to the participants. The goal of slice distribution is to ensure each participant only receives a partial server-side model and thus can not perform effective attacks. Next, we will introduce the two components of this stage: model slices and slice mapping rules.

Each slice is a subset of the server-side model. The upper right corner of Figure 2 shows five model slices for five participants. For example, participant 1 has slice 1 that consists of three branches: the first branch of layer 1, the second branch of layer 2, and the first branch of layer 3. Each slice has the same structure as the template model (the upper left corner of Figure 2). Thus the slices can be trained on the local devices with existing training pipelines (such as cross-entropy loss and gradient decent optimization).

Slice Mapping Rule. This rule records the mapping relationship between the branches and slices. Formally, suppose the i -th branch of the j -th layer composes the slice p , the mapping rule is denoted as $map[p][j] = i$. The slice p can be

denoted as:

$$Slice_p = \{map[p][j] | j = 1, \dots, n\}. \quad (1)$$

As shown in the upper right part of Figure 2, the slices are constructed following the mapping rule. For example, the first row of the mapping rule defines how to construct the slice 1.

The mapping rule map is generated at the startup phase and fixed during the training process. We utilize a random strategy to determine the slice-branch mapping relationship, *i.e.* one slice randomly selects a branch from each layer. This random-combination scheme has two advantages. First, it encourages branches from a different layer to collaborate, *i.e.* so that they can be better integrated into the server-side model. Second, it reduces the privacy leakage of participant data because different participants get different slices (various combinations of branches).

D. Slice Aggregation

After participants train the model slices with local data, they upload the slices to the server. The input of this stage is the uploaded slices, and the output is the aggregated ensemble model. Conventional model aggregation techniques can not aggregate model slices because they require the uploaded and ensemble models to have the same architecture. However, in our case, the uploaded slice is a subset of the ensemble model. This stage aggregates the slice knowledge into the ensemble model to solve this problem by updating branches and training the ensemble model. FedSlice first performs per-branch aggregation to collect slice knowledge into branches. Then FedSlice trains the ensemble model to update the fusion layers and improve collaboration between branches.

Per-Branch Aggregation. The input of this stage is the uploaded slices, and the output is the aggregated branches. First, per-branch aggregation requires a pre-defined per-weight aggregation function. Researchers have proposed various per-weight aggregation functions (such as per-weight aggregation [46]). This function can be arbitrarily selected from the prior work [46], [40], [5], [39].

Then FedSlice builds an inverse mapping rule from the slice mapping rule. This rule defines which slices are used to update each branch. Contrary to the mapping rule (which records which branch is distributed from a layer to a slice), the inverse mapping rule records the slices each branch composes. Formally, FedSlice constructs the inverse map by

$$i_map[i][j] = \{p | map[p][j] = i\}. \quad (2)$$

The inverse map defines *which* slices are used to update one branch, and the per-weight aggregation function defines *how* the branches are updated. Each branch is updated by the per-weight aggregation function using the slices in the inverse map.

The lower left part of Figure 2 displays the inverse mapping of the demo case. Taking the first branch of layer 1 as an example, the inverse mapping records (1, 3) (the array’s first column of the first row). It means this branch is updated by slice 1 and slice 3. It is because, in the mapping rule, slice

1 and 3 are composed of branch 1 (the blue squares of the upper right part of Figure 2). As shown in the lower part of Figure 2, branches are updated by the inverse mapping rule.

Joint Training. FedSlice jointly trains the whole server model with a public dataset to converge the ensemble model. Without this step, the ensemble model cannot converge when participants’ data is highly diverse. In this scenario, the updated slices have a large deviation from each other, causing the aggregated branches difficult to produce valid outputs. This step also trains the weights of the fusion layers to aggregate the output features of the prior layer’s branches. Because we choose FeatBN as fusion layers (as stated in Section III-B), this step does not need a large amount of data.

IV. EVALUATION

Research Questions. In this section, we want to answer the following research questions:

- **RQ 1:** Can FedSlice reduce the successful rate of the four attackers in Section II-A?
- **RQ 2:** How is the training efficiency of FedSlice compared with unprotected baselines?
- **RQ 3:** What is the accuracy loss of FedSlice?
- **RQ 4:** How do hyper-parameters influence the performance of FedSlice?

Implementation Details. In our experiment, we set the number of branches in each layer n as ten and select FeatBN as the fusion layer. We conduct the experiments of FedSlice on a server with two GeForce GTX 1080Ti GPUs, two Intel Xeon CPUs with 16 cores, and 64GB of memory.

A. Defense Effectiveness

The goal of FedSlice is to protect FL models from the four attacks mentioned in Section II-A. Therefore, we evaluate how effectively can FedSlice counter the four attacks in this section.

1) **Setup: Dataset.** We evaluate FedSlice on six representative datasets. The dataset selection follows prior FL benchmark FedML [28] and Leaf [11]. We choose three computer vision datasets (EMNIST, CIFAR10, and CIFAR100) as they are commonly used by prior FL literatures [39], [28]. We choose one natural language process dataset (Shakespeare [11]) to study the effectiveness of FedSlice on different tasks. We also choose two large-scale datasets designed for FL evaluation: FEMNIST and Celeba [11]. FEMNIST contains handwriting digits and alphabets from more than 3,000 participants, and Celeba contains human faces of more than 4600 different people.

Data Partition. We use the default partition of the original setting for the already partitioned datasets (Celeba, FEMNIST, and Shakespeare). For other datasets, we partition the data into 100 participants and follow the prior α -degree of non-IID to simulate both IID and non-IID distribution [22].

We simulate the public dataset by randomly selecting 5% of data from each dataset. This selection follows prior setting [13], [38], [15], [73], [42]. The model trained with this amount of data has low accuracy. Thus the task requester

TABLE I: The performance of server-side model protection of FedSlice in terms of the accuracy. The metrics of FedSlice include the performance of the server-side model (the higher the better) and the average performance of participant models (the lower the better). We list the average relative values of FedSlice w.r.t the unconditional sharing baseline at the bottom row.

		McMahan [46]			Asad [5]			Li [40]		
		Baseline	FedSlice		Baseline	FedSlice		Baseline	FedSlice	
			Server ↑	Partic. ↓		Server↑	Partic.↓		Server↑	Partic.↓
EMNIST	IID	79.82	81.74	69.19	79.40	80.65	67.48	83.09	80.19	69.62
	Non-IID	79.04	80.04	61.04	79.44	80.52	59.10	83.04	80.20	69.59
CIFAR10	IID	51.07	50.52	12.87	55.58	51.39	10.36	54.53	51.07	14.48
	Non-IID	48.54	47.20	11.73	50.29	49.05	13.76	52.76	50.04	13.66
CIFAR100	IID	26.50	26.50	1.32	28.54	27.10	1.17	24.21	22.92	7.78
	Non-IID	25.52	26.59	3.7	24.56	25.30	1.13	22.51	23.73	3.93
FEMNIST		73.82	75.02	3.72	72.94	68.66	5.10	76.51	76.34	23.77
Shakespear		41.28	39.52	27.85	42.15	39.95	33.07	39.53	36.72	21.90
Celeba		88.89	86.03	53.73	82.26	84.51	64.75	91.45	89.34	66.98
Average of Relative Value		-	-0.20%	40.63%	-	-1.94%	42.02%	-	-3.11%	47.70%

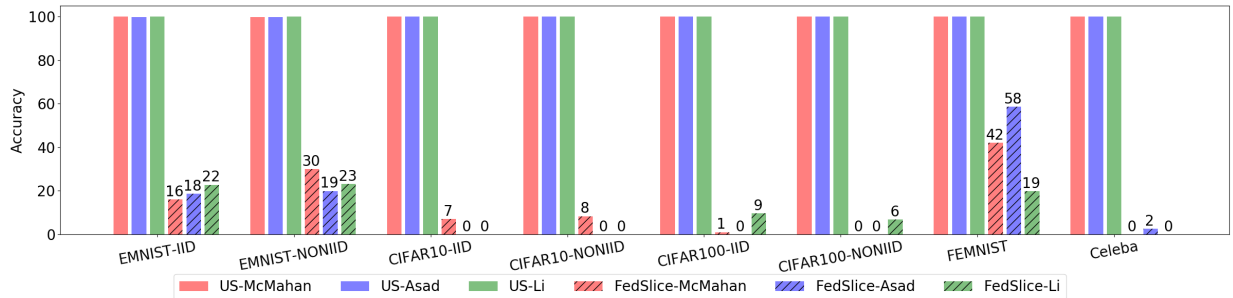


Fig. 3: The effectiveness of adversarial attack mitigation. Attack success rate (ASR) is reported to measure how much adversarial examples generated by participant models can confuse the server-side model.

needs the private data from the participants to help improve the accuracy [38].

Models. We select four representative models as baselines. For EMNIST and FEMNIST, we use a simple CNN model (LeNet) according to the benchmark suggestion [11], [28]. For CIFAR10, CIFAR100, and Celeba we use a more complex CNN model with residual connections (ResNet20 [29] and ResNet18 [29]) following previous work [35]. We choose an RNN model suggested by the Leaf [11] benchmark for Shakespeare dataset.

Baseline Approaches We select three representatives “unconditional sharing” baselines to compare with FedSlice. The three baselines are McMahan [46], Asad [5], and Li [40], which are among the most used FL aggregation methods [39], [28], [64]. These three methods have been used in industry and academia [63], [32], [38], [58].

2) *Free-Rider Attacks: Metrics.* Similar to previous work, we use the average accuracy of all the participant’s models to measure the model leakage [41], [23]. A higher participants’ accuracy represents more model leakage, and a lower value means better protection of the server-side model’s functionality. Since the absolute accuracy of different datasets and baseline techniques are distinct, we also utilize relative values

for better comparison across different datasets. Let ACC_S be the accuracy of the server-side model and ACC_p be the accuracy of participant p , the relative leakage is defined as $rLKG = \frac{1}{|P|} \sum_{p \in P} ACC_p / ACC_S$.

Results. The model leakage can be observed by comparing the “Server” column and the “Partic.” column of Table I. The “Server” column shows the accuracy of the server-side model, and the “Partic.” column shows the averaged accuracy of the participants’ models. A more significant gap between the “Server” column and the “Partic.” column represents a lower functionality that the server-side model leaks.

For the three baselines, FedSlice’s average leakage of server functionality is 40.63%, 42.02%, and 47.70%. It means that FedSlice can reduce the leaked functionality of the server-side model to an average of 43.45%. The leaked functionality of baselines is 100% because they send the shared model to participants.

3) *Adversarial Attacks: Metrics.* We follow prior work [19] to report the attack success rate (ASR), which computes how many adversarial examples generated from the local model can mislead the output of the server-side model. We implement a strong adversarial attack (PGD attack [45]) to evaluate ASR

in the worst case. The hyper-parameters of the PGD attack follow the default setting of the original paper [45].

Results. Figure 3 shows the defense result of computer vision datasets against strong PGD attacks. We list the unconditional sharing baselines (the first three histograms for each dataset) and FedSlice (the subsequent three histograms) together. McMahan [46], Asad [5], and Li [40] are represented in red, blue, and green, respectively.

For all three baselines, the attack success rate (ASR) reaches more than 99% (not labeled explicitly for simplicity), meaning that the security risk of adversarial attacks is high. On the contrary, the ASR of FedSlice is reduced substantially. For EMNIST and FEMNIST, the ASR is below 30% in six out of nine cases. For other datasets, the ASR is below 10%, and there are 60% cases (nine out of fifteen) with an ASR of zero. Averagely, FedSlice achieves an ASR of 11.66%. The comparison between baselines and FedSlice demonstrates that distributing model slices can effectively defend against adversarial attacks from the participant and protect the server-side model’s security.






Although the malicious participant can perform a black-box adversarial attack, the cost is higher, and the effectiveness is lower than the PGD attack. According to literatures [8], [17], the attacker has to query the target model thousands of times to test the decision boundary of the server-side model, which is slow and expensive.

4) *Membership Information Attacks: Metrics.* We evaluate FedSlice with four state-of-the-art membership inference attacks: the neural network (NN) attack [56], the Top3 attack, the Loss attack [54], and the Gradient attack [50]. All four attacks use the adversary participant’s local model and infer other honest participants’ membership information. We report four metrics for each attack (precision, recall, F1 score, and accuracy) to comprehensively evaluate the attack performance. The selected metrics follow prior literatures [34], [50]. For each metric, a higher value represents more privacy leakage. Note that membership attack is a binary classification task. The lower limit is a random guess, with the lowest value of all the aforementioned metrics as 0.5.

Results. Table III displays the performance of four attacks against baselines and FedSlice. For baseline techniques, all four attacks can effectively infer the membership information. The attack performances of the Loss attack and the Gradient attack are generally higher because these two attacks use more information (data label and gradient information). Averagely, the precisions of all attacks are above 0.74, meaning that about three-fourths of the predicted positive samples are truly the training members. This high precision threatens the privacy of the participants’ training data.

On the contrary, according to Table III, the success rate of FedSlice is substantially reduced. The performance of all four attack techniques resembles that of a random guess. The highest F1 is 0.54, and the highest accuracy is 0.55, which are all lower than the baselines. The average accuracy is reduced from 71.91% to 51.58%. The average F1 score is reduced from 0.71 to 0.48. It means the adversary participant extracts nearly

TABLE II: The effectiveness to reduce deep gradient leakage.

	Ground Truth	Baseline		FedSlice	
		DLG	iDLG	DLG	iDLG
MSE	-	0.00012	0.00014	1.75	1.49
Sample					

no private information.

We also studied why our approach could reduce the success rate of membership inference. Take the loss attack as an example. Prior literature concluded that membership information is mainly embedded into the loss magnitude [50]. Member data samples usually have low loss magnitudes, and non-member data samples have high magnitudes. It is because, during the training phase, the DNN weights are optimized to minimize the loss of the member samples. For FedSlice, both member and non-member samples have high loss magnitude because the adversary’s slice is never trained on honest participants’ data. The adversary participant can not extract usable private information from similar magnitudes.

5) *Deep Gradient Leakage Reduction: Metrics.* We choose two state-of-the-art attacks that use leaked gradient to recover participants’ training samples [53]: deep gradient leakage (DGL) [74] and improved deep gradient leakage (iDGL) [72]. We display both quantitative results and qualitative results of each attack. We show the Mean Squared Error (MSE) for quantitative results, which is consistent with the original attack evaluation [74], [72]. MSE computes the pixel-value difference between the original image and the recovered image. The range of MSE is [0, 2], where 0 means the recovered image is identical to the original one, and 2 means the recovered image is completely random noise. We display two randomly selected images for qualitative results to compare the attack effect between different techniques.

Results. Table II displays both quantitative and qualitative results. For baselines, the MSE of both attacks is smaller than 0.0002, which is similar to the lower limit. For FedSlice, the MSE is around 1.50, which is four magnitudes larger than the MSE of baselines and is close to the upper limit. The bottom of Table II shows the qualitative results. The left column is the ground truth sample, and the others are recovered samples. The samples recovered from baselines resemble the ground truth. There are only several countable pixels for the flower image that are different from the ground truth image (the central part of the flower and the right side of the image). On the contrary, the samples of FedSlice are random noise, meaning that the attacker can not infer adequate input information from the distributed model slices.

Answer to **RQ 1:** FedSlice can reduce the model leakage, F1 score, ASR, and MSE for the four attacks mentioned in Section II-A.

B. Model Efficiency

FedSlice is designed to avoid heavy encryption and TEE. In this section, we evaluate how much training time could be saved by avoiding TEE and encryption.

We recorded the training time on all datasets to compare the training efficiency between FedSlice and baselines. Figure 4 shows the training time on CIFAR10 in minutes. It can be observed that FedSlice marginally increases the training overhead. Averagely, FedSlice takes 14.3% longer time than the unconditional sharing baseline. Compared to the TEE-based solution (up to $36.1\times$ slower [55] than baseline), FedSlice is $31.6\times$ faster. Compared to cryptographic solution (more than $1,000\times$ slower than baseline [60]), FedSlice is $877.2\times$ faster. The FedSlice is 3.7% to 33.2% slower on other datasets. We skip the detailed figures due to the space limit.

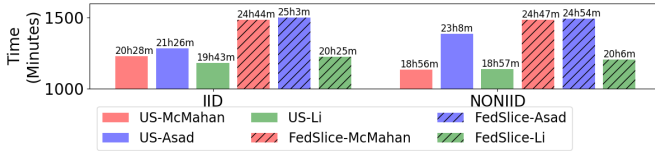


Fig. 4: The comparison of FL training time on CIFAR10.

Answer to **RQ 2**: FedSlice improves the training speed by avoiding TEE and encryption.

C. Accuracy Loss

In this section, we want to evaluate whether FedSlice protects FL models by harming the model’s accuracy. To do this, we measure the accuracy of the server-side model of FedSlice and unconditional sharing.

Results. We can observe the accuracy loss of FedSlice by comparing the “Baseline” column and the “Server” column of Table I. The “Baseline” column is the accuracy of baseline models, and the “Server” column is the accuracy of the FedSlice models. To rigorously compare baselines with FedSlice, we compute the statistical significance with Wilcoxon signed-rank test [66]. For each baseline, the null hypothesis is that there is no accuracy difference between the baseline and FedSlice. The computed p-values for the three baselines are 0.78, 0.31, and 0.02. It means at a confidence level of 5%, we receive the null hypothesis for McMahan [46] and Asad [5] and reject the hypothesis for Li [40]. For Li [40], the averaged accuracy loss is 3.11%. These observations mean that FedSlice does not remarkably harm the accuracy of the server-side model.

Answer to **RQ 3**: For two of the three baselines, FedSlice does not reduce the server-side model accuracy with statistical significance.

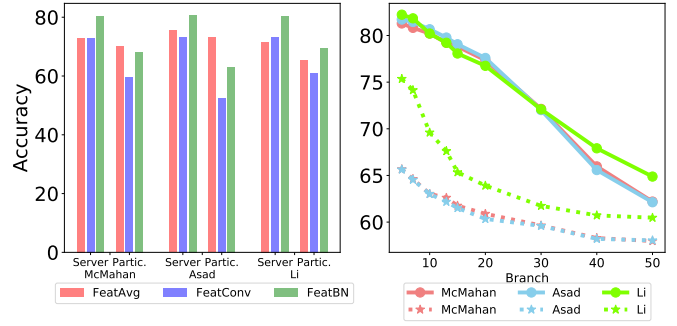


Fig. 5: Ablation study on the fusion module and the number of branches on the EMNINST dataset.

D. Effect of Hyper-Parameters.

We then study hyper-parameters’ effect on the two characteristics of FedSlice: the accuracy of the server-side model and the model leakage defense. The hyper-parameters include the choice of fusion layers and the number of branches. For the choice fusion layers, we evaluate “FeatAvg”, “FeatConv”, and “FeatBN”, as discussed in Section III-B. For the number of branches, we set the range from 2 to 50. We select EMNIST and all three baselines to perform the experiments due to the page limit. We also did exploratory experiments on other datasets, and the results are similar.

The left figure in Figure 5 shows the results of the choice of fusion layers. For each baseline, we plot the accuracy of the server-side model and participant model together. In the figure, FeatBN achieves the highest server accuracy in all cases, demonstrating that it can effectively fuse knowledge from different participants. The participants’ accuracy of FeatBN is lower than FeatAvg, but higher than FeatConv. This result means FeatBN can protect model functionality better than FeatAvg but worse than FeatConv. However, we still choose FeatBN as the fusion layer because it has higher server-side accuracy.

The right figure in Figure 5 displays the model sensitivity concerning the number of branches. The solid lines with circles represent the server-side model, and the dotted lines with stars represent the participants’ model. Figure 5 shows that as the number of branches increases, the accuracy of the server-side model and the participant-side models decreases. We select the number of branches as ten because it achieves high server-side accuracy while maintaining a significant gap between the server-side and participant-side models.

Answer to **RQ 4**: FedSlice chooses the hyper-parameters that can better balance the server-side model’s accuracy and model leakage protection.

E. Threats to Validity.

Internal Validity. The training hyper-parameters may be one internal validity in the experiment. Such hyper-parameters include learning rates, training rounds, and epochs for each local device. We mitigate this threat by using the recommended

TABLE III: Protection against membership inference attacks on CIFAR100. For each case, we use precision, recall, F1 score, and accuracy as metrics. The last two rows show the average value over all baselines.

		NN				Top3				Loss				Gradient			
		Prec	Recall	F1	Acc	Prec	Recall	F1	Acc	Prec	Recall	F1	Acc	Prec	Recall	F1	Acc
McMahan [46]	US	0.81	0.71	0.69	0.71	0.83	0.76	0.75	0.76	0.81	0.81	0.81	0.81	0.84	0.78	0.77	0.78
	FedSlice	0.52	0.52	0.51	0.52	0.51	0.51	0.51	0.51	0.54	0.53	0.48	0.53	0.53	0.53	0.53	0.53
Asad [5]	US	0.84	0.78	0.77	0.78	0.82	0.73	0.71	0.73	0.85	0.82	0.81	0.82	0.85	0.80	0.79	0.80
	FedSlice	0.50	0.50	0.50	0.50	0.51	0.51	0.50	0.51	0.56	0.55	0.51	0.55	0.55	0.55	0.54	0.55
Li [40]	US	0.59	0.58	0.57	0.58	0.60	0.59	0.58	0.59	0.65	0.64	0.63	0.64	0.63	0.63	0.63	0.63
	FedSlice	0.50	0.50	0.46	0.50	0.50	0.50	0.47	0.50	0.49	0.49	0.38	0.49	0.50	0.50	0.42	0.50
Average	US	0.74	0.69	0.68	0.69	0.75	0.69	0.68	0.69	0.77	0.76	0.75	0.76	0.77	0.74	0.73	0.74
	FedSlice	0.51	0.51	0.49	0.51	0.51	0.51	0.49	0.51	0.53	0.52	0.46	0.52	0.53	0.53	0.50	0.53

settings of the public benchmark [28], [11] and keeping the parameters assistant across all experiments.

External Validity. The choice of evaluated datasets and models may be one threat to external validity. We mitigate this threat using the diverse and recommended choices of public benchmark [28], [11]. The evaluated datasets include both computer vision tasks and natural language processing tasks. Another threat is the number of simulated participants and data distribution. To mitigate this threat, we simulate a large number of participants (100 to more than 4K) and evaluate both IID and non-IID distribution to demonstrate the effectiveness of FedSlice under real-world settings.

Construct Validity. The choice of evaluation metrics may be one threat to construct validity. We mitigate this threat by selecting the same metrics from prior work [28], [11], [19], [37], [54], [50], [74], [72].

V. DISCUSSION

Collusion of Malicious Clients. One possible threat against FedSlice is that several participants collude to attack the server model. The expected number of colluded participants relates to the ensemble model structure, which is a high-dimension variant of the coupon collector problem [51]. For a simple situation in which the shared model is split into three layers and ten branches, the expectation of the number of colluding participants is 38.75. Therefore, although multiple participants may conspire to compromise the FedSlice, the number of colluding participants is not trivial.

Limitations. One limitation is that the FedSlice requires a small portion of public data to train the ensemble model. However, the server may not have the computation ability to train the model on the server-side. Another limitation is the size of the ensemble model is larger than the meta-model. Although it is not difficult to store such a model on the server side, deploying it on edge devices may be difficult. We will address the limitations in future work.

VI. RELATED WORK

Security Issues of FL. A few recent papers point out that some ill-intended participants may pretend to contribute data and steal the server-side model for free (free-rider attacks) [23], [41]. Existing defense techniques mainly focus on detecting such dishonest participants [41]. FedSlice uses an orthogonal technique that protects the server-side model

by avoiding sharing the model with all participants. FedSlice and dishonest participant detection can be applied together to protect the trained model better.

Privacy Risk of FL. Researchers have proposed several attacks, *i.e.* gradient leakage [74], [72], membership leakage [47], [50] and attribute leakage [47]. Zhu *et al.* restored the training data from the shared gradient, and the restored data is pixel-wise accurate for images [74]. Nasr *et al.* found that the shared gradients and the history of model updates expose even more membership information [50]. Melis *et al.* also displayed that FL can leak attributes unrelated to the target task, such as whether a person wears glasses in the computer vision task and the doctor’s specialty in the health-related reviews [47]. Conventional solutions to these attacks are TEE or encryption based, while FedSlice leverages a novel technique that is more efficient.

Other Attacks to FL. Besides attacks mentioned in Section II-A, FL may also suffer label flipping [69] and poisoning attacks [59]. Recently researchers proposed several byzantine-robust aggregation algorithms to defend against label flipping and poisoning attacks from malicious participants [9], [67], [13]. These byzantine-robust solutions can not defend against the four attacks discussed in this paper because they deliver the server-side model to all participants and can not avoid information leakage. On the contrary, FedSlice focuses on the model distribution stage and avoid sending the server-side model to the participants.

Besides, byzantine-robust solutions are orthogonal with FedSlice, and the two solutions can be integrated to provide a higher security level. These solutions can be regarded as a per-weight aggregation function and be used to aggregate uploaded branches.

VII. CONCLUSION

This paper aims to protect FL from the four attacks mentioned in Section II-A without using heavy encryption and TEE-based techniques. To achieve this goal, we propose FedSlice, a federated learning framework that ensures each participant only receives a slice of the server-side model, which prevents them from performing effective attacks. We evaluate FedSlice on six datasets and five models. The experiment results show that FedSlice effectively defends the four attacks with less than 2% accuracy loss and about 14% computation overhead.

REFERENCES

- [1] Arm cortex-a55. https://en.wikipedia.org/wiki/ARM_Cortex-A55.
- [2] Arm cortex-a75. https://en.wikipedia.org/wiki/ARM_Cortex-A75.
- [3] Jetson nano and trustzone. <https://forums.developer.nvidia.com/t/jetson-nano-and-trustzone/149051>.
- [4] Raspberry pi 4 trustzone support. <https://forums.raspberrypi.com/viewtopic.php?t=311331>.
- [5] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Fedopt: Towards communication efficiency and privacy preservation in federated learning. *Applied Sciences*, 10(8):2864, 2020.
- [6] Umar Asif, Jianbin Tang, and Stefan Harrer. Ensemble knowledge distillation for learning improved and efficient networks. *arXiv preprint arXiv:1909.08097*, 2019.
- [7] Árpád Beszédés, Csaba Faragó, Z Mihaly Szabo, János Csirik, and Tibor Gyimóthy. Union slices for program maintenance. In *International Conference on Software Maintenance, 2002. Proceedings.*, pages 12–21. IEEE, 2002.
- [8] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII*, volume 11216 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2018.
- [9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 118–128, 2017.
- [10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.
- [11] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.
- [12] Hakki C. Cankaya. Access control from an OS security perspective. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security, 2nd Ed*, pages 7–9. Springer, 2011.
- [13] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*.
- [14] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57. IEEE Computer Society, 2017.
- [15] Hongyan Chang, Virat Shejwalkar, Reza Shokri, and Amir Houmansadr. Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer. *arXiv preprint arXiv:1912.11279*, 2019.
- [16] Boyuan Chen, Mingzhi Wen, Yong Shi, Dayi Lin, Gopi Krishnan Rajbahadur, and Zhen Ming Jiang. Towards training reproducible deep learning models. In *Proceedings of the 44th International Conference on Software Engineering*, pages 2202–2214, 2022.
- [17] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. Hop-skipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1277–1294. IEEE, 2020.
- [18] Sijie Cheng, Jingwen Wu, Yanghua Xiao, and Yang Liu. Fedgems: Federated learning of larger server models via selective knowledge fusion. *arXiv preprint arXiv:2110.11027*, 2021.
- [19] Ting-Wu Chin, Cha Zhang, and Diana Marculescu. Renofeat: A simple transfer learning method for improved adversarial robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3243–3252, June 2021.
- [20] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.
- [21] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.
- [22] 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.
- [23] Yann Fraboni, Richard Vidal, and Marco Lorenzi. Free-rider attacks on model aggregation in federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1846–1854. PMLR, 2021.
- [24] Xuanqi Gao, Juan Zhai, Shiqing Ma, Chao Shen, Yufei Chen, and Qian Wang. Fairneuron: Improving deep neural network fairness with adversary games on selective neurons. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*.
- [25] Yanjie Gao, Yonghao Zhu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. Resource-guided configuration space reduction for deep learning models. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*.
- [26] Jianmin Guo, Quan Zhang, Yue Zhao, Heyuan Shi, Yu Jiang, and Jianguang Sun. Rnn-test: Towards adversarial testing for recurrent neural network systems. *IEEE Transactions on Software Engineering (TSE)*, 2021.
- [27] Robert J Hall. Automatic extraction of executable program subsets by simultaneous dynamic program slicing. *Automated Software Engineering*, 2(1):33–53, 1995.
- [28] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.
- [29] Kaiping 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.
- [30] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.
- [31] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 603–618. ACM, 2017.
- [32] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-iid data quagmire of decentralized machine learning. In *International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020.
- [33] Pei Huang, Yuting Yang, Minghao Liu, Fuqi Jia, Feifei Ma, and Jian Zhang. ϵ -weakened robustness of deep neural networks. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*.
- [34] Bo Hui, Yuchen Yang, Haolin Yuan, Philippe Burlina, Neil Zhenqiang Gong, and Yinzhi Cao. Practical blind membership inference attack via differential comparisons. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.
- [35] Yerlan Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 2022-08-18.
- [36] Pin Ji, Yang Feng, Jia Liu, Zhihong Zhao, and Zhenyu Chen. Asrtest: automated testing for deep-neural-network-driven speech recognition systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 189–201, 2022.
- [37] Klas Leino and Matt Fredrikson. Stolen memories: Leveraging model memorization for calibrated white-box membership inference. In Srđjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1605–1622. USENIX Association, 2020.
- [38] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.
- [39] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: vision, hype and reality for data privacy and protection. *arXiv preprint arXiv:1907.09693*, 2019.
- [40] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous

- networks. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020.
- [41] Jierui Lin, Min Du, and Jian Liu. Free-riders in federated learning: Attacks and defenses. *arXiv preprint arXiv:1911.12560*, 2019.
- [42] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33:2351–2363, 2020.
- [43] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 35(4):70–82, 2020.
- [44] Zixi Liu, Yang Feng, Yining Yin, and Zhenyu Chen. Deepstate: selecting test suites to enhance the robustness of recurrent neural networks. In *Proceedings of the 44th International Conference on Software Engineering*, pages 598–609, 2022.
- [45] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
- [46] 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.
- [47] 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.
- [48] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 94–108, 2021.
- [49] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124, 2011.
- [50] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 739–753. IEEE, 2019.
- [51] Peter Neal. The generalised coupon collector problem. *Journal of Applied Probability*, 45(3):621–629, 2008.
- [52] Rangeet Pan and Hridesh Rajan. On decomposing a deep neural network into modules. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann, editors, *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 889–900. ACM, 2020.
- [53] Xudong Pan, Mi Zhang, Yifan Yan, Jiaming Zhu, and Zheming Yang. Exploring the security boundary of data reconstruction via neuron exclusivity analysis. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3989–4006, 2022.
- [54] Ahmed Salem, Yang Zhang, Mathias Humbert, Mario Fritz, and Michael Backes. MI-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Network and Distributed Systems Security Symposium 2019*. Internet Society, 2019.
- [55] Tianxiang Shen, Ji Qi, Jianyu Jiang, Xian Wang, Siyuan Wen, Xusheng Chen, Shixiong Zhao, Sen Wang, Li Chen, Xiapu Luo, et al. {SOTER}: Guarding black-box inference for general neural networks at the edge. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 723–738, 2022.
- [56] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.
- [57] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering*, pages 338–349, 2022.
- [58] Canh T Dinh, Nguyen Tran, and Josh Nguyen. Personalized federated learning with moreau envelopes. *Advances in Neural Information Processing Systems*, 33:21394–21405, 2020.
- [59] 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.
- [60] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
- [61] Haoyu Wang, Yuanchun Li, Yao Guo, Yuvraj Agarwal, and Jason I. Hong. Understanding the purpose of permission use in mobile apps. *ACM Trans. Inf. Syst.*, 35(4):43:1–43:40, 2017.
- [62] Jialai Wang, Han Qiu, Yi Rong, Hengkai Ye, Qi Li, Zongpeng Li, and Chao Zhang. Bet: black-box efficient testing for convolutional neural networks. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*.
- [63] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.
- [64] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.
- [65] Mark Weiser. Program slicing. *IEEE Transactions on software engineering*, (4):352–357, 1984.
- [66] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.
- [67] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR, 2018.
- [68] Boxi Yu, Zhiqing Zhong, Xinran Qin, Jiayi Yao, Yuancheng Wang, and Pinjia He. Automated testing of image captioning systems. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*.
- [69] Mengmei Zhang, Linmei Hu, Chuan Shi, and Xiao Wang. Adversarial label-flipping attack and defense for graph neural networks. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 791–800. IEEE, 2020.
- [70] Wanrong Zhang, Shruti Tople, and Olga Ohrimenko. Leakage of dataset properties in multi-party machine learning. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2687–2704. USENIX Association, 2021.
- [71] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. Dynamic slicing for deep neural networks. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann, editors, *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 838–850. ACM, 2020.
- [72] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. iDLG: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.
- [73] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.
- [74] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.