



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

Ziqi Zhang*, Yuanchun Li[†], Bingyan Liu[‡], Yifeng Cai*, Ding Li*[§], Yao Guo*[§], Xiangqun Chen*

*Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University

[†]Institute for AI Industry Research (AIR), Tsinghua University

[‡]School of Computer Science, Beijing University of Posts and Telecommunications

{ziqi_zhang, caiyifeng, ding_li, yaoguo, cherry}@pku.edu.cn, liyuanchun@air.tsinghua.edu.cn, bingyanliu@bupt.edu.cn

Abstract—Crowdsourcing Federated learning (CFL) is a new crowdsourcing development paradigm for the Deep Neural Network (DNN) models, also called “software 2.0”. In practice, the privacy of CFL 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 Trusted Execution Environments (TEEs). To improve the efficiency of protecting CFL from these attacks, this paper proposes FedSlice to prevent malicious participants from getting the whole server-side model while keeping the performance goal of CFL. 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 CFL framework.

I. INTRODUCTION

Deep Neural Network (DNN) models, also dubbed as “software 2.0” [19], [37], are one of the major focusing points of software engineering researchers today [15], [35], [77], [32], [77], [83], [87]. Crowdsourcing Federated Learning (CFL), which allows the task requester (e.g. large companies) to use participants’ private data to collaboratively train DNN models, is a new crowdsourcing strategy for deep learning models [56], [31], [54]. The task requester pays the participants for the usage of private data, and the participants receive a monetary incentive. Unlike traditional software crowdsourcing, in which the key security concern is the integrity of the software under development, CFL concerns not only the integrity of the model but also the privacy of participants. Although CFL protects participant privacy by limiting the participant data to local devices, recent research showed that CFL may still leak participant privacy [88], [85], [62], [56].

Possible Attacks. This paper focuses on four attacks against federated learning: free-rider attacks [45], [22], adversarial

attacks [52], gradient leakage attacks [88], [85], and inference attacks [62], [56]. Free-rider attacks mean a malicious participant can steal the server-side 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 [55], [10]. However, as reported in prior work [71], cryptographic ML algorithms are more than 1,000× slower than ordinary protocols. Another solution is to perform model updates inside the Trusted Execution Environments (TEEs) [18], [71], [84]. However, TEEs are about 36× slower than the untrusted hardware [71], [64]. Besides, TEEs require dedicated hardware support, which is not always available for federated devices [4], [1], [5].

Existing techniques have low efficiency because they aim to simultaneously protect the machine learning model from the malicious server and participants. Heavy encryption techniques or dedicated hardware, such as TEEs, are necessary to defend against the malicious server because it is the dominator of the CFL 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 CFL models [71], [18].

Key Idea. This paper aims to efficiently protect CFL models in a different but more practical scenario in which the server is trustworthy, but a few participants are malicious [54], [9]. This scenario is more practical because, in many realistic CFL applications, the server belongs to reputable companies or organizations with financial resources. Violating the participants’

[§]Corresponding author.

privacy harms the reputation of these companies and may result in a 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 CFL. As CFL needs many participants, it is hard to control the identification of participants. An adversary can pretend to be an honest participant, join the CFL 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 CFL models by achieving two sub-goals. First, we want to defend against the four attacks mentioned before. Second, we aim to avoid using TEEs or encryption because these techniques can slow down the training speed of an CFL model by $36\times$ to $1000\times$. To achieve our goal, we build FedSlice, a new CFL 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 TEEs or 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 a part of the model so that she cannot infer private information about other participants nor compromise the integrity of the whole model. This strategy is fundamentally different from existing solutions [43], [53], [6], [9], [12], [48], which distribute a shared model indiscriminately to all participants. Since our approach avoids distributing a shared model to all participants, we can avoid using TEEs or encryption techniques, allowing more efficient model protection.

Our Solution. FedSlice implements the discriminative model distribution strategy 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. FedSlice has three advantages. First, FedSlice can generate heterogeneous models 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. Second, as the heterogeneous models are generated from the same template model, existing CFL aggregation techniques can combine the building layers to fuse the models back into the template model. Third, FedSlice can be integrated into current CFL aggregation strategies, such as McMahan [53], Li [43], and Asad [6]. Therefore, FedSlice can be deployed into existing CFL 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 by 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.

II. BACKGROUND

A. Security Risks of Federated Learning

Although CFL 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 CFL model without contributing to it [45], [22]. A free-rider can forge the locally-trained model 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 by waiting for the server to deliver it.

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 [52], [13]. 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.

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 [62], or internal gradient [56] of the shared model to infer privacy. Although membership inference and attribute inference have different goals, the techniques are similar, so we mainly discuss membership inference in this paper.

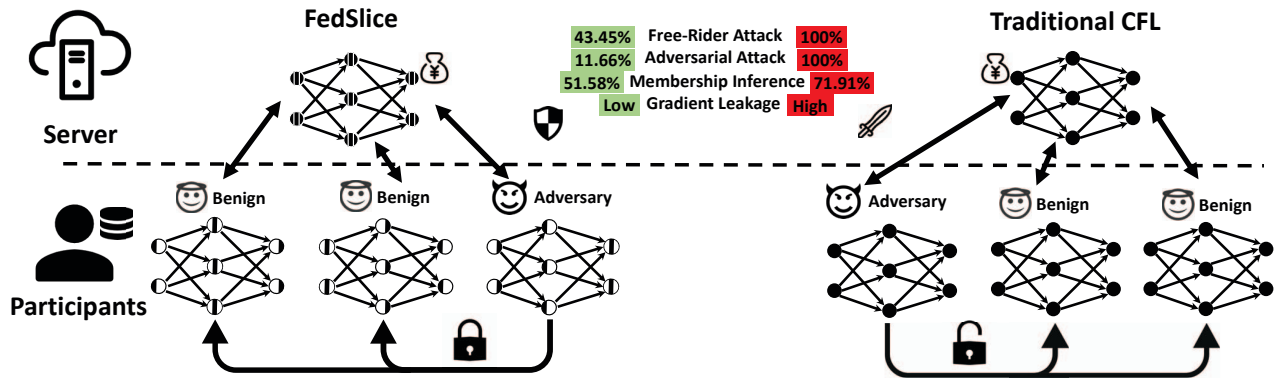


Fig. 1: The comparison between FedSlice and traditional federated learning. Traditional CFL distributes the entire server-side model to all participants but FedSlice only distributes a (different) model slice to each participant.

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 realistic (pixel-level accurate) input samples [88], [85]. Thus this vulnerability poses a significant threat to the private data of honest participants.

B. Motivation and Challenges

Motivation. Figure 1 shows the high-level idea of FedSlice and its difference compared to the conventional CFL process. Traditional CFL 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 CFL, FedSlice, as shown in the left part of Figure 1, implements a discriminative model distribution strategy. FedSlice 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 cannot perform effective attacks.

The design of FedSlice is inspired by access control in operating system [72]. 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 does not know enough privacy to perform attacks.

Technical Challenges. Enabling the discriminative distribution strategy is particularly challenging because it requires aggregating heterogeneous models across the server and participant 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 CFL 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 [53], [6], [43]. 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 aggregate heterogeneous models.

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 [74] or slice combinations [26], [8]. 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 techniques, 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 [58], [82]. One possibly viable solution is neural architecture search [20]. This technique 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 [24].

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 CFL 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 a large amount of required data. For example, distilling a model on the CIFAR10 dataset requires about 4,000 samples per class [7]. With this amount of data, the task requester can directly train a new model without CFL.

Besides, 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 CFL applications, the distillation time explodes and becomes impractical [11].

C. Threat Model

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 literature [12], [40], [14], [86], [46]. In other words, the participant-side models should have inadequate functionalities compared to the server-side model and contain as little privacy as possible 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. 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. 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 have trained 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.

Intuition. The intuition behind the branches is to replicate different copies of a model layer so that 1) different copies contain knowledge of different participants and 2) the copies of different layers can be combined to form an integrated model with the same architecture as the template model. The intuition behind the fusion layer is to effectively aggregate branch outputs. As Section IV-D shows, naive aggregation (e.g., feature average) leads to suboptimal performance of the server-side model.

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,

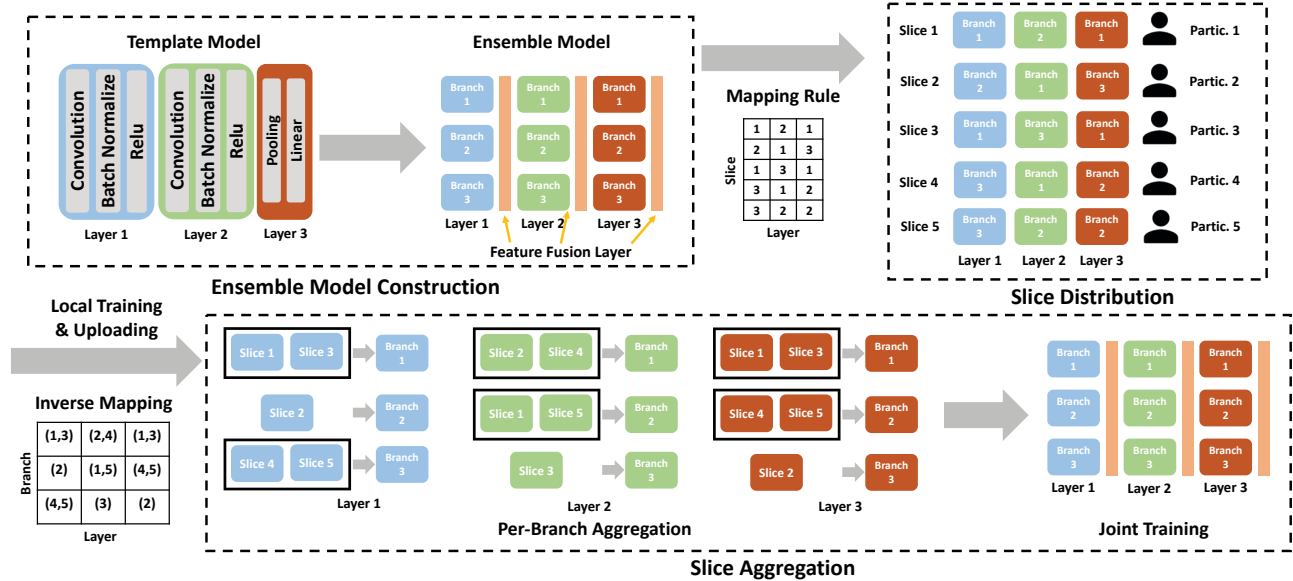


Fig. 2: The overall pipeline of FedSlice consists of three stages: ensemble model construction, slice distribution, and slice aggregation. In the demo, the template model contains three layers (denoted in blue, green, and red boxes) and each layer has three branches (denoted by the number in each box).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

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 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 one, branch two, etc.).

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. 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.

Model Slice. 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 one has slice one that consists of three branches: the first branch of layer one, the second branch of layer two, and the first branch of layer three. 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 descent optimization).

Slice Mapping Rule. This rule records the mapping relationship between the branches and slices. Suppose the i -th

branch of the j -th layer composes the slice p , and 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 slice one.

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 different layers to collaborate 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 average [53]). This function can be arbitrarily selected from the prior work [53], [43], [6], [41].

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 (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 indicated by the inverse map.

The lower left part of Figure 2 displays the inverse mapping of the demo case. Taking the first branch of layer one 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 one and slice three. It is because, in the mapping rule, slice one and slice three are composed of branch one (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.

Evaluation protocol. We conducted our experiments through FedML [28], a widely-used commercial open-source library and benchmark for federated machine learning. FedML provides a well-designed evaluation protocol for comparing CFL approaches. Therefore, we directly use the standard procedure of FedML in our evaluation. Specifically, we configured FedML in the Standalone Simulation mode and simulated the participants and the server on a desktop. During the CFL training procedure, FedML first simulates each participant to train local models one by one. Then it simulates the central server to aggregate the participants' models. FedML repeats these two steps for several rounds to train the CFL model. For more details on how FedML simulates the participants, please refer to the FedML paper [28].

A. Defense Effectiveness

The goal of FedSlice is to protect CFL 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 CFL benchmark FedML [28] and Leaf [11]. We choose three computer vision datasets (EMNIST, CIFAR10, and CIFAR100) as they are

commonly used by prior CFL literatures [41], [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 CFL evaluation: FEMNIST and Celeba [11]. FEMNIST contains handwriting digits and alphabets from more than 3,000 participants, and Celeba includes human faces of more than 4600 people.

Data Partition. Data distribution is critical in federated learning because, in real-world deployment, it may affect the performance of CFL models [86]. We use the default partition strategy 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 (independent and identically distributed) and non-IID distribution [21]. IID means that the distributions across all participants are the same, and non-IID means that each participant has a unique data distribution. For IID simulation, FedML iterates all participants and randomly assigns one sample to each participant until all samples are assigned. For non-IID simulation, we modified FedML following previous work [21] since FedML does not provide a default Non-IID partitioning strategy. For each participant, we randomly select a major label, and half of this participant’s data is sampled from the major label’s data. The rest half of the participant’s data is uniformly sampled from the data that does not belong to the major label.

We simulate the public dataset by randomly selecting 5% of data from each dataset. This selection follows prior setting [12], [40], [14], [86], [46]. The model trained with this amount of data has low accuracy. Thus the task requester needs the private data from the participants to help improve the accuracy [40].

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 [34]. We choose an RNN model suggested by the Leaf [11] benchmark for Shakespeare dataset.

Baseline Approaches We select three representative “unconditional sharing” baselines to compare with FedSlice. The three baselines are McMahan [53], Asad [6], and Li [43], which are among the most used CFL aggregation methods [41], [28]. These three methods have been used in industry and academia [40], [67].

2) *Free-Rider Attacks: Attack Protocol.* We implemented the attack proposed by Fraboni *et al.* and Lin *et al.* [45], [22]. In our implementation, the malicious participants do not update real gradients but fake gradients. After the training procedure, the malicious participants use the shared model for profit. The amount of stolen model functionality is quantified by the accuracy of malicious participants’ local models.

Metrics. We use the average accuracy of all the participant’s models to measure the model leakage [45], [22]. A higher participant’s 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 $rLKG = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} ACC_p / ACC_S$.

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 performance of FedSlice is similar to a fully protected baseline. For example, for the LEAF dataset, the participants’ accuracy (about 60%) is similar to that of prior literature (65%) [9]. On the contrary, the leaked functionality of unconditional sharing baselines is 100% because they send the shared model to participants.

3) *Adversarial Attacks: Attack Protocol.* The malicious participant joins the training procedure as benign participants. After training, when the server-side model is utilized for inference, the malicious participant runs the adversarial attack with the local model to generate adversarial samples and uses these samples to mislead the server-side model.

Metrics. We report the attack success rate (ASR) to evaluate the defense effectiveness. ASR 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 [52]) to evaluate ASR in the worst case. The hyper-parameters of the PGD attack follow the default setting of the original paper [52].

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 [53], Asad [6], and Li [43] 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.

The performance of FedSlice is similar to the fully pro-

TABLE I: The performance of server-side model protection of FedSlice in terms of 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 [53]			Asad [6]			Li [43]		
		Baseline	FedSlice		Baseline	FedSlice		Baseline	FedSlice	
			Server \uparrow	Partic. \downarrow		Server \uparrow	Partic. \downarrow		Server \uparrow	Partic. \downarrow
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%

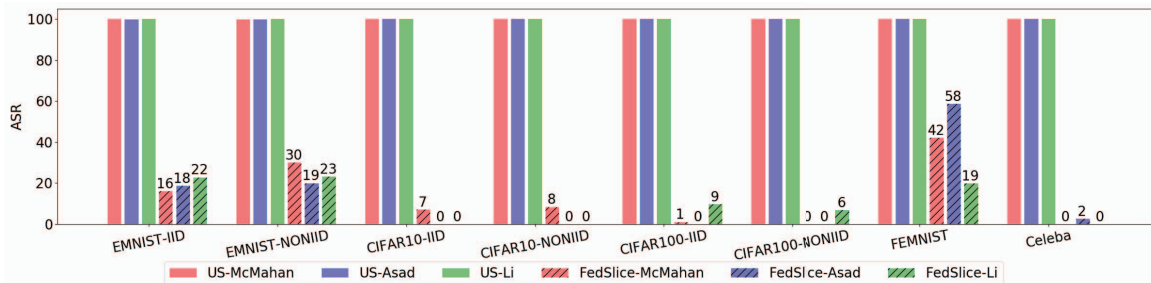


Fig. 3: The effectiveness of adversarial attack mitigation. Attack success rate (ASR) is reported to measure how much adversarial examples generated by participant models can confuse the server-side model.

tected baselines because both FedSlice and the fully protected baselines can prevent the adversary from getting an accurate enough model from local data, which is necessary for existing approaches to generate valid adversary samples. For example, white-box adversary attacks require an attack model with an accuracy higher than 90% for CIFAR10 and 75% for CIFAR100, while FedSlice limit the accuracy of locally trained models below 20%. The fully protected baselines have low accuracy because local data is not enough to train high-performance models.

4) *Membership Information Attacks: Attack Protocol.* In our simulation, the malicious participants join the training procedure as benign participants. After training, the adversary uses the shared model and unknown input samples to determine if other participants use these samples during the training procedure.

Metrics. We select four state-of-the-art membership inference attacks: the neural network (NN) attack [65], the Top3 attack, the Loss attack [62], and the Gradient attack [56]. The four attacks use different information from the shared model. The NN attack uses the logit output to infer membership. The Top3 attack uses the highest three confidences of model output. The Loss attack uses the loss value of the unknown sample on the shared model. The Gradient attack uses the gradient vector

of the last layer’s input to infer membership information. To comprehensively evaluate the attack performance, we report four metrics for each attack (precision, recall, F1 score, and accuracy). The selected metrics follow prior literature [33], [56]. For each metric, a higher value represents more privacy leakage. Note that the membership attack is a binary classification task. The lower limit is a random guess, with the lowest value of all the metrics mentioned above as 0.5.

Results. Table II displays the performance of four attacks against baselines and FedSlice. For baseline techniques, all four attacks can effectively infer the membership information. The performance of the Loss attack and the Gradient attack is generally higher because these 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 II, 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

TABLE II: 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 [53]	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 [6]	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 [43]	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

0.71 to 0.48. It means the adversary participant extracts nearly no private information. The attack performance against a fully protected baseline is a random guess, and the performance of FedSlice is similar to the fully protected baseline.

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 [56]. 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: Attack Protocol.

The malicious participant uploads the local model and receive the shared model as usual. Then, he computes the averaged gradient by differentiating two versions of the received model. Last, the malicious participant uses existing attacks and the average gradient to recover the training data of other participants.

Metrics. We choose two state-of-the-art attacks that use leaked gradients to recover participants’ training samples [60]: deep gradient leakage (DGL) [88] and improved deep gradient leakage (iDGL) [85]. We display both quantitative results and qualitative results of each attack. We show the Mean Squared Error (MSE) for quantitative results, consistent with the original attack evaluation [88], [85]. 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 III 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 III 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.

TABLE III: 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					

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 (similar to the fully protected baseline), 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, ASR, F1 score, and MSE for the four attacks mentioned in Section II-A.

B. Model Efficiency

FedSlice is designed to avoid heavy encryption and TEEs. 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. In the brackets above the dashed histograms of FedSlice, we show the percentage of the time of FedSlice over the time of the unconditional sharing baselines. 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 [64] than baseline), FedSlice is $31.6\times$ faster. Compared to cryptographic solutions (more than $1,000\times$ slower than baseline [71]), FedSlice is $877.2\times$ faster. The FedSlice is 3.7% to 33.2% slower on other datasets. We skipped the detailed figures due to the space limit.

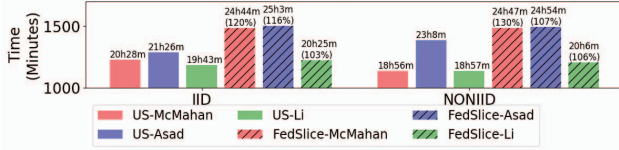


Fig. 4: The comparison of CFL training time on CIFAR10.

Answer to **RQ 2**: FedSlice improves the training speed by avoiding TEEs or encryption.

C. Accuracy Loss

In this section, we want to evaluate whether FedSlice protects CFL 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 [75]. 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 have little or no evidence to reject the null hypothesis for McMahan [53] and Asad [6], but we can reject the hypothesis for Li [43]. For Li [43], 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.

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 were 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

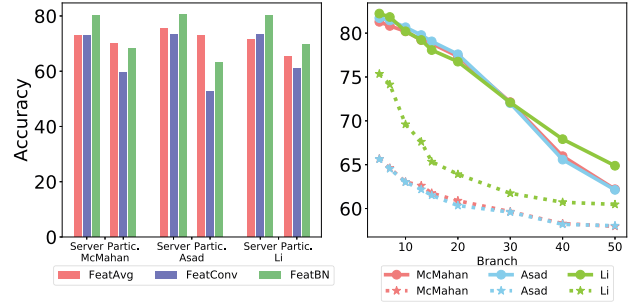


Fig. 5: Ablation study on the fusion module and the number of branches on the EMNIST dataset.

means FeatBN can protect model functionality better than FeatAvg but worse than FeatConv. However, we still choose FeatBN 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 settings of the public benchmark [28], [11] and keeping the parameters consistent 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], [39], [62], [56], [88], [85].

V. DISCUSSION

Collusion of Malicious Participants. 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 [57]. 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 non-trivial.

Experiment Simulation. Although the CFL process is simulated on a desktop, our experiments could still reflect the realistic scenarios of CFL. Our primary goal is to protect CFL models from the four attacks without losing too much accuracy. The related metrics are platform-independent. For the additional training overhead of FedSlice, we compare FedSlice and baselines on the same platform. It is a common practice in prior literature to compare the efficiency and convergence speed of different CFL algorithms on a simulated environment to approximate the realistic performance gap [43], [36], [48]. Besides, we use the standard simulation framework, FedML, which provides a systematic and realistic simulation of CFL environments.

Limitations. FedSlice has two limitations: the constrained application on the CFL and the dependency on the public dataset. FedSlice is designed for the Crowdsourcing Federated Learning scenario [70], [80], in which participants only contribute the data and receive monetary incentives. Therefore, participants do not need the final model in CFL [80]. CFL is widely-used in industry [2], [3], [16]. For the public dataset, FedSlice requires the server to use a public dataset to assist slice aggregation. Given the diverse data available on the Internet, the task requester can arbitrarily select one dataset according to the type of task. Designing a data-free algorithm for slice aggregation is an important future work of FedSlice.

Research Implications. The design of FedSlice provides a new secure crowdsourcing solution for large companies that seek to train commercial DNN models with private data. Different from the traditional CFL framework, FedSlice protects the intellectual property and the security of the companies' DNN model. Compared to the TEE-based solutions, FedSlice reduces the hardware requirement (equipped with a TEE on the edge device) for the participants and enlarges the participant group so that the task requester can utilize more private data.

VI. RELATED WORK

Security and Privacy Issues of CFL. A few recent papers point out that some ill-intended participants may steal the server-side model for free (free-rider attacks) [22], [45]. Existing defense techniques mainly focus on detecting such dishonest participants [45], [47]. FedSlice uses an orthogonal technique and can be integrated with detection-based defenses to safeguard the trained model better.

For privacy risks, researchers have proposed several attacks, *i.e.* gradient leakage [88], [85], membership leakage [54], [56] and attribute leakage [54]. Zhu *et al.* restored accurate training data from the shared gradient [88]. Nasr *et al.* found that the

shared gradients may expose additional membership information [56]. Melis *et al.* found that CFL can expose attributes unrelated to the target task, causing privacy leakage [54]. Conventional defenses are based on TEEs or encryption, while FedSlice leverages a novel technique that is more efficient.

Besides attacks mentioned in Section II-A, CFL may also suffer poisoning attacks [69]. To defend against such attacks, researchers proposed several byzantine-robust aggregation [9], [12]. Such defenses are orthogonal with FedSlice. They can not defend the four attacks discussed in this paper because they deliver the server-side model to all participants. Byzantine-robust solutions can be regarded as a per-weight aggregation function and can be integrated into FedSlice.

Model-based Analysis on DNNs. In recent years, SE researchers have proposed various model-based algorithms to analyze the behaviors of DNNs (also dubbed as "SE4AI"). One popular direction of SE4AI is DNN testing which aims to find incorrect behaviors of DNN models [50], [25], [25], [35], [73], [78], [79], [63]. Coverage is a common metric to guide DNN testing [76], [68], [51] due to its effectiveness in analyzing traditional programs [42], [44]. Other researchers try to fix the incorrect behaviors and repair DNN faults [66], [23]. Improving the robustness against adversarial attacks is an important goal of DNN testing and repair [81], [83], [32]. Some researchers focus on the modularization of DNN models and try to decompose and recombine DNN like traditional programs [61], [58], [59].

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. Our tool and data are available on the Internet ¹.

VIII. ACKNOWLEDGEMENTS

We would like to thank the anonymous ICSE reviewers for their valuable feedback of this paper. This work was partly supported by the National Natural Science Foundation of China (62172009,62141208), and a project sponsored by ZTE Communications.

REFERENCES

- [1] Arm cortex-a55. https://en.wikipedia.org/wiki/ARM_Cortex-A55.
- [2] Google assistant to improve its hearing through federated learning. <https://www.slashgear.com/google-assistant-to-improve-its-hearing-through-federated-learning-28665845/>.
- [3] Google wants to crowdsource data for medial research with its new health studies app. <https://www.androidpolice.com/2020/12/09/google-wants-to-crowdsource-data-for-medial-research-with-its-new-health-studies-app/>.

¹<https://zenodo.org/record/7536416#.Y9yIZOxBwUE>

- [4] Jetson nano and trustzone. <https://forums.developer.nvidia.com/t/jetson-nano-and-trustzone/149051>.
- [5] Raspberry pi 4 trustzone support. <https://forums.raspberrypi.com/viewtopic.php?t=311331>.
- [6] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Fedopt: Towards communication efficiency and privacy preservation in federated learning. *Applied Sciences*, 10(8):2864, 2020.
- [7] Umar Asif, Jianbin Tang, and Stefan Harrer. Ensemble knowledge distillation for learning improved and efficient networks. *arXiv preprint arXiv:1909.08097*, 2019.
- [8] 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.*, 2002.
- [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*, 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*, 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] 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*.
- [13] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP*.
- [14] 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.
- [15] 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*, 2022.
- [16] Bekir Sait Ciftler, Abdullatif Albaser, Nouredine Lasla, and Mohamed Abdallah. Federated learning for RSS fingerprint-based localization: A privacy-preserving crowdsourcing method. In *16th International Wireless Communications and Mobile Computing Conference*, 2020.
- [17] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*.
- [18] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016.
- [19] Malinda Dilhara, Ameya Ketkar, and Danny Dig. Understanding software-2.0: A study of machine learning library usage and evolution. *ACM Trans. Softw. Eng. Methodol.*, 30(4):55:1–55:42, 2021.
- [20] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*.
- [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 2020)*.
- [22] Yann Fraboni, Richard Vidal, and Marco Lorenzi. Free-rider attacks on model aggregation in federated learning. In *International Conference on Artificial Intelligence and Statistics*, 2021.
- [23] 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*.
- [24] 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*.
- [25] 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.
- [26] Robert J Hall. Automatic extraction of executable program subsets by simultaneous dynamic program slicing. *Automated Software Engineering*.
- [27] 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.
- [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] 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*, 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*.
- [32] 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*.
- [33] Bo Hui, Yuchen Yang, Haolin Yuan, Philippe Burlina, Neil Zhenqiang Gong, and Yinzi 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.
- [34] Yerlan Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 2022-08-18.
- [35] 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.
- [36] Jiayin Jin, Jiayang Ren, Yang Zhou, Lingjuan Lyu, Ji Liu, and Dejing Dou. Accelerated federated learning with decoupled adaptive optimization. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022*.
- [37] Andrej Karpathy. Software 2.0. <https://karpathy.medium.com/software-2-0-a64152b37c35>.
- [38] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.
- [39] Klas Leino and Matt Fredrikson. Stolen memories: Leveraging model memorization for calibrated white-box membership inference. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020*.
- [40] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.
- [41] 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.
- [42] Siqi Li, Xiaofei Xie, Yun Lin, Yuekang Li, Ruitao Feng, Xiaohong Li, Weimin Ge, and Jin Song Dong. Deep learning for coverage-guided fuzzing: How far are we? *IEEE Transactions on Dependable and Secure Computing*, 2022.
- [43] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems 2020*.
- [44] Yi Li, Shaohua Wang, and Tien Nguyen. Fault localization with code coverage representation learning. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*.
- [45] Jierui Lin, Min Du, and Jian Liu. Free-riders in federated learning: Attacks and defenses. *arXiv preprint arXiv:1911.12560*, 2019.
- [46] 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*, 2020.
- [47] Bingyan Liu, Yifeng Cai, Ziqi Zhang, Yuanchun Li, Leye Wang, Ding Li, Yao Guo, and Xiangqun Chen. Distfl: Distribution-aware federated learning for mobile scenarios. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2021.

- [48] Bingyan Liu, Yao Guo, and Xiangqun Chen. Pfa: Privacy-preserving federated adaptation for effective model personalization. In *Proceedings of the Web Conference 2021*.
- [49] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 2018.
- [50] 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*, 2022.
- [51] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018.
- [52] 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*.
- [53] 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*, 2017.
- [54] 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)*.
- [55] 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.
- [56] 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)*.
- [57] Peter Neal. The generalised coupon collector problem. *Journal of Applied Probability*, 2008.
- [58] 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*.
- [59] Rangeet Pan and Hridesh Rajan. Decomposing convolutional neural networks into reusable and replaceable modules. In *Proceedings of the 44th International Conference on Software Engineering*, 2022.
- [60] 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)*.
- [61] Binhang Qi, Hailong Sun, Xiang Gao, and Hongyu Zhang. Patching weak convolutional neural network models through modularization and composition. *arXiv preprint arXiv:2209.06116*, 2022.
- [62] 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*.
- [63] Qingchao Shen, Junjie Chen, Jie Zhang, Haoyu Wang, Shuang Liu, and Menghan Tian. Natural test generation for precise testing of question answering software. In *IEEE/ACM Automated Software Engineering (ASE)*, 2022.
- [64] 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)*.
- [65] 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)*.
- [66] 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*, 2022.
- [67] Canh T Dinh, Nguyen Tran, and Josh Nguyen. Personalized federated learning with moreau envelopes. *Advances in Neural Information Processing Systems*, 2020.
- [68] Yongqiang Tian, Zhihua Zeng, Ming Wen, Yepang Liu, Tzu-yang Kuo, and Shing-Chi Cheung. Evaldnn: A toolbox for evaluating deep neural network models. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*.
- [69] 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*, 2019.
- [70] Yongxin Tong, Yansheng Wang, and Dingyuan Shi. Federated learning in the lens of crowdsourcing. *IEEE Data Eng. Bull.*, 2020.
- [71] 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*.
- [72] 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.*, 2017.
- [73] 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*.
- [74] Mark Weiser. Program slicing. *IEEE Transactions on software engineering*, 1984.
- [75] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*. 1992.
- [76] Xiaofei Xie, Tianlin Li, Jian Wang, Lei Ma, Qing Guo, Felix Juefei-Xu, and Yang Liu. Npc: Neuron path coverage via characterizing decision logic of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2022.
- [77] 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*.
- [78] Yuanyuan Yuan, Qi Pang, and Shuai Wang. Unveiling hidden dnn defects with decision-based metamorphic testing. *arXiv preprint arXiv:2210.04942*, 2022.
- [79] Yuanyuan Yuan, Shuai Wang, Mingyue Jiang, and Tsong Yueh Chen. Perception matters: Detecting perception failures of vqa models using metamorphic testing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.
- [80] Rongfei Zeng, Chao Zeng, Xingwei Wang, Bo Li, and Xiaowen Chu. A comprehensive survey of incentive mechanism for federated learning. *CoRR*, abs/2106.15406, 2021.
- [81] Yingyi Zhang, Zan Wang, Jiajun Jiang, Hanmo You, and Junjie Chen. Toward improving the robustness of deep learning models via model transformation. 2022.
- [82] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. Dynamic slicing for deep neural networks. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.
- [83] Ziqi Zhang, Yuanchun Li, Jindong Wang, Bingyan Liu, Ding Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. Remos: Reducing defect inheritance in transfer learning via relevant model slicing. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022*.
- [84] Ziqi Zhang, Lucien KL Ng, Bingyan Liu, Yifeng Cai, Ding Li, Yao Guo, and Xiangqun Chen. Teeslice: slicing dnn models for secure and efficient deployment. In *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, pages 1–8, 2022.
- [85] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. iDLG: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.
- [86] 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.
- [87] Jie Zhu, Leye Wang, and Xiao Han. Safety and performance, why not both? bi-objective optimized model compression toward ai software deployment. *arXiv preprint arXiv:2208.05969*, 2022.
- [88] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, 2019.