Model Hijacking Attack in Federated Learning

Zheng Li*, Siyuan Wu*, Ruichuan Chen, Paarijaat Aditya,
Istemi Ekin Akkus, Manohar Vanga, Min Zhang, Hao Li†, Yang Zhang.

***Abstract*—Machine learning (ML), driven by prominent paradigms such as centralized and federated learning, has made significant progress in various critical applications. However, its remarkable success has been accompanied by various attacks. Recently, the model hijacking attack has shown that ML models can be hijacked to execute tasks different from their original tasks, which increases both accountability and parasitic computational risks.**

Nevertheless, thus far, this attack has only focused on centralized learning. In this work, we broaden the scope of this attack to the federated learning domain, where multiple clients collaboratively train a global model without sharing their data. Specifically, we present the first-of-its-kind hijacking attack against the global model in federated learning, namely HijackFL. The adversary aims to force the global model to perform a different task (called hijacking task) from its original task without the server or benign client noticing. To accomplish this, unlike existing methods that use data poisoning to modify the target model's parameters, HijackFL searches for pixel-level perturbations based on their local model (without modifications) to align hijacking samples with the original ones in the feature space. When performing the hijacking task, the adversary applies these perturbations to the hijacking samples, compelling the global model to identify them as original ones and predict them accordingly. Extensive experiments demonstrate HijackFL significantly outperforms baselines, e.g., 92.75% vs. 10%. We further investigate the factors that affect its performance and discuss possible defenses to mitigate its impact. Code is available at https://github.com/zhenglisec/HijackFL.

***Index Terms*—Machine Learning, Federated Learning, Security, Model Hijacking, Dataset Manipulation.**

I. Introduction

MACHINE learning (ML) has progressed rapidly in the past decade. Centralized learning and federated learning [1], [13], [16], [17], as the two most representative paradigms, have played a crucial role in this advancement. In centralized learning, models are trained on data collected at a central server. Federated learning, on the other hand, allows for collaborative training across multiple devices, preserving local privacy and data ownership.

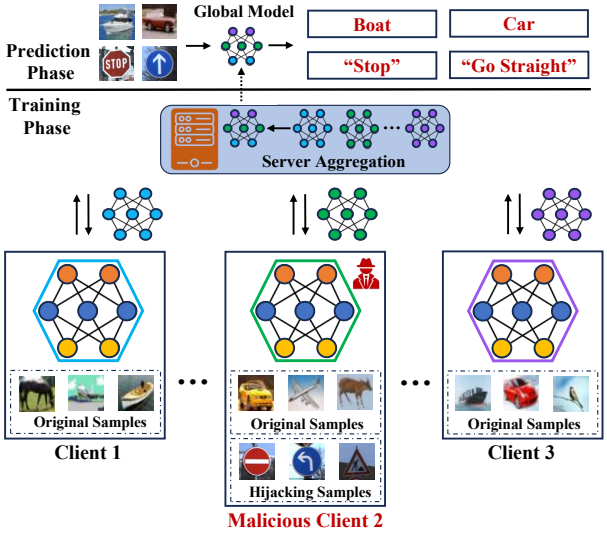


Fig. 1: The overview of model hijacking attack in federated learning. The original task is CIFAR-10, while the hijacking task is GTSRB.

Despite being popular, ML models are vulnerable to various security and privacy attacks. In this paper, we focus on a new type of attack, the model hijacking attack, where the adversary aims to force the target model to perform a completely different task during deployment or inference, known as the hijacking task. For instance, the target model is designed for the CIFAR-10 task, while an adversary aims to repurpose it for a different task, such as GTSRB. Successful model hijacking attacks can cause severe consequences. First, they constitute parasitic computing. By surreptitiously utilizing the victim's infrastructure to execute private tasks, the adversary effectively transfers the burdens of deployment and maintenance to the model provider. Second, this unauthorized exploitation raises severe accountability risks. Since model providers are responsible for the system's outputs, they are obligated to detect any discrepancies between legitimate and malicious usage. When providers fail to identify the anomalous inputs that trigger these unauthorized tasks, they are forced to bear the legal and regulatory liabilities for the resulting illegal services.

Motivation. In NDSS'22, Salem et al. [21] presented the first model hijacking attack against machine learning models, or more precisely, centralized learning models. As the data provider, the adversary can manipulate the target model's training dataset by integrating the hijacking dataset into the original one, i.e., the data poisoning approach. Thus, models trained on such a poisoned dataset will have their parameters

Zheng Li is with the School of Cyber Science and Technology, Shandong University, Qingdao, China; the State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, China; and the Shandong Key Laboratory of Artificial Intelligence Security, Shandong University, Qingdao, China (Email: zheng.li@sdu.edu.cn).

Siyuan Wu is with the Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China, and the University of Chinese Academy of Sciences, Beijing, China (Email: wusiyuan2023@iscas.ac.cn).

Ruichuan Chen, Paarijaat Aditya, Istemi Ekin Akkus, and Manohar Vanga are with Nokia Bell Labs (Email: {ruichuan.chen, paarijaat.aditya, istemi_ekin.akkus, manohar.vanga}@nokia-bell-labs.com).

Min Zhang and Hao Li are with the Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China (Email: zhangmin@iscas.ac.cn, lihao@iscas.ac.cn).

Yang Zhang is with the CISPA Helmholtz Center for Information Security (Email: zhang@cispa.de).

*These authors contributed equally to this work.

†Corresponding author: Hao Li.

modified, i.e., hijacking effects are injected, leading to changes in their behavior. Si et al. [24] also utilize the data poisoning approach to hijack NLP models. However, the data poisoning approach is only applicable to hijack centralized learning models.

In this study, we consider the model hijacking attack in the federated learning (FL) paradigm. FL involves multiple clients for training, creating an attack surface where malicious clients could hijack the *global model*. However, two unique properties hinder existing hijacking attacks that modify the global model through data poisoning (verified in Section V-A). First, malicious clients need to modify their local model parameters first, causing differences from benign model parameters, which can be detected by the central server. Second, FL allows collaborative training, meaning malicious clients can only participate in a few training rounds. In most times, only benign clients contribute to the global model, leading to the hijacking effect being eliminated. To overcome this, we aim to answer, "Is it possible to hijack a cooperatively trained global model without modifying it?"

Contributions. We introduce the first model hijacking attack against the FL global model, called HijackFL. Specifically, HijackFL achieves the goal of hijacking by adding pixel-level perturbations, denoted as "cloaks," onto the hijacking samples. Using their local model (fetched from the global model), the adversary optimizes these cloaks to ensure that the hijacking samples closely resemble the original samples within the feature space. Furthermore, due to the high similarity between the local model and final deployed global model, these cloaks will exhibit a high degree of transferability. When attacking the global model in deployment, the hijacking samples with cloaks will be treated as original samples by the model and predicted to original classes. Finally, malicious clients map these predicted original classes to hijacking classes based on a predefined class mapping. Since HijackFL does not modify the local model or inject hijacking effects into the global model, it effectively overcomes the challenges faced by existing attacks.

We conduct extensive evaluations on four benchmark image datasets and three widely used models. Empirical results show that HijackFL has no side effect in the utility of the global model and, furthermore, outperforms existing attacks [21], [24]. For instance, when the global model is ResNet-18 and the original dataset is CIFAR-10, HijackFL can achieve an attack success rate of 75.45% for the GTSRB hijacking dataset. In contrast, existing attacks only achieve an attack success rate of around 10%. Furthermore, we conduct a comprehensive ablation study to analyze the factors that may affect the attack performance and also explore two possible defenses to mitigate our attack.

Abstractly, our contributions can be summarized as follows:

- We propose the first-of-its-kind model hijacking attack against the global model in FL domain, called HijackFL.
- Without modifying the local/global model, HijackFL adds cloaks to the hijacking samples so that these samples have features that are highly similar to those of the original samples, thus allowing the global model to categorize them into the original class.
- Extensive evaluations show that HijackFL achieve significant performance regarding both utility and attack success rate.

II. Preliminaries and Related Works

A. Federated Learning

Federated Learning (FL) [1], [13], [16], [17] is a machine learning paradigm that allows multiple clients to collaboratively train a model without sharing their training data. Formally, considering an FL training process involves n clients, each client holds a set of private training data. At each training round t, the central server selects m clients and sends them the current global model G^t. Each selected client, e.g., ith of m clients, updates this model to a new local model F_i^{t+1} by training on its private data, and sends the difference $F_i^{t+1} - G^t$ back to the central server. The central server then aggregates the received model parameters from these m clients, e.g., averaging these parameters (FedSGD [23]), to obtain the new global model G^{t+1}:

$$G^{t+1} = G^t + \frac{\eta}{n} \sum_{i=1}^{m} \left(F_i^{t+1} - G^t \right) \tag{1}$$

Here, the η is a global learning rate set by the central server, controlling the fraction of the global model that is updated. The central server and all parameters share the same model architecture, and only the model parameters $F_i^{t+1} - G^t$ are communicated regularly between them. Normally, the training does not stop until the global model converges, a malicious client thus always has an opportunity to share model updates with the central server.

B. Model Hijacking Attack

Model hijacking attack is a new class of training time attacks against ML models. In centralized learning, Salem et al. [21] assume the adversary acts as a data provider and follows a similar implementation as data poisoning attacks, i.e., poisoning the training dataset to modify the target model's parameters. Two attack methods are proposed in Salem et al. [21]. The first is to directly mix the original dataset with the hijacking dataset. This achieves the upper bound on attack performance yet causes a drawback: it can be detected by the model owner as the original and hijacking datasets are visually different. Thus, Salem et al. introduce an extra model to transform the hijacking dataset into a format visually similar to the original dataset. Subsequently, Si et al. [24] presented an attack similar to the second one mentioned above, but targeting NLP models. Both studies demonstrate the applicability of the model hijacking attack to the centralized learning domain.

Note. It is important to note that model hijacking differs from data poisoning, backdooring, and especially adversarial perturbation attacks. Data poisoning [7], [12], [25] aims to *jeopardize the models' utility*, backdooring [10], [14] links *the trigger inserted in the original sample* (e.g., a white square at the corner) to specific model output, and adversarial perturbation attacks primarily seek to induce a human-model

cognitive mismatch by causing misclassification with imperceptible noise. Though the backdoor attack can be considered a specific instance of model hijacking to some extent, hijacking a model is to execute *an entirely different task*, regardless of the original one. Crucially, both backdooring and adversarial perturbation attacks are confined to the original label space, aiming primarily to induce misclassification rather than extending the model's capabilities. This implies that achieving the hijacking objective is more challenging. Further, the adversary is free to determine the hijacking task in the model hijacking attack.

C. Threat Model in FL

We define the threat model by specifying the identity, knowledge, capability, and objective of the participants. We assume the central server and honest clients are benign and strictly follow the standard FL protocol. The adversary is modeled as a malicious client who, while adhering to the parameter exchange protocol to evade detection, maintains full control over their local training process. This includes holding an additional private dataset and optimizing localized, task-specific components to leverage the global model's feature extractor.

The adversary's primary objective is defined as Model Hijacking, analogous to attacks in centralized learning [21], [24], and is driven by the economic benefits of parasitic computing. By surreptitiously occupying the shared parameter space to execute unauthorized private tasks (e.g., GTSRB) alongside the original task (e.g., CIFAR-10), the attacker effectively transfers the substantial burdens of model training and infrastructure maintenance to the victim. This unauthorized exploitation allows the adversary to obtain high-end AI capabilities at a negligible marginal cost. Crucially, this behavior violates the model's functional integrity and creates severe accountability risks. By forcing the model to execute hidden, potentially illegal tasks without consent, the attacker leaves the model provider to bear the legal and regulatory liabilities arising from the system's output. We provide a concrete real-world scenario in Appendix to illustrate the multi-stakeholder harms.

A successful model hijacking attack in the FL domain should satisfy the following two requirements. We refer to the victim global model in deployment as the hijacked global model.

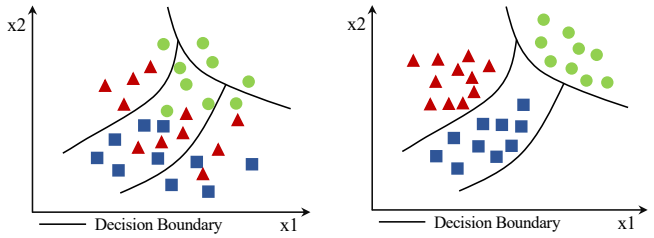
Requirement 1. *The hijacked global model should have a similar performance as the clean global model on its original dataset.*

Requirement 2. *The hijacked global model should correctly classify most samples in the hijacking dataset.*

III. Methodology

A. Overview

ML models are trained to extract and identify features from input samples, which they employ for prediction. However, their feature identification can be misled by adversarial example attacks [18], [19], [20], where slight perturbations to input samples alter the model's feature perception. Our attack exploits this vulnerability to mislead the global model's perception of features extracted from hijacking samples toward the original samples.



(a) Without Cloak (b) With Cloak

Fig. 2: An illustration of cloaking the hijacking samples, which can then be classified by the global model. The background is the decision boundary of the original task. The colored points represent the features of hijacking samples.

At a certain training round, the adversary *updates their local models by fetching the global model* from the central server. Leveraging the local model, the adversary then learns slight perturbations (called cloaks) for the hijacking samples, enabling the local model to extract features close to those from the original samples. If the global model remains relatively stable in subsequent training rounds, the cloaks learned on the local model can be transferred to the *final global model*. This transferability enables the final global model to also extract features from the hijacking samples that are similar to the original samples. The final global model thinks it works correctly, as it perceives the features of input samples within the "original dataset." Through this process, the adversary successfully accomplishes the goal of hijacking the final global model in deployment. We provide an illustration in Figure 2. As depicted in Figure 2a, the decision boundary of the local/global model fails to distinctly classify the hijacking samples into separate regions. After adding the cloaks, the hijacking samples will become clearly separable into different regions (Figure 2b).

We emphasize that the decision boundary remains unchanged because the local/global model is not modified, but only the learned cloaks are added to the hijacking samples. This strategic shift prevents the two limitations of existing methods based on data poisoning approach: (1) being detected by the server and (2) the hijacking effect being removed. We note that cloak computation requires only a single, well-chosen training round rather than continuous participation. To select this timing, the adversary retains a subset of the original task data and tracks the global model using loss or accuracy metrics to estimate the optimal attack window (i.e., already converged). Thus, the attack is robust to random client sampling because it uses a one-shot trigger. Under typical federated learning settings, the probability that the attacker is never sampled during the critical phase is negligible. Even when executed before model convergence, the attack still achieves effective hijacking rather than failing. We further analyze the impact of suboptimal timing in Section V-E.

TABLE I: List of notations.

Notation	Description
y	A class of the original dataset (original class)
h	A class of the hijacking dataset (hijacking class)
x_h	A hijacking sample of class h
δ	A cloak perturbation for a hijacking sample
$x_h \oplus \delta$	Cloaked version of x_h
$\bar{h}$	A hijacking class beyond y
$\mathcal{X}_h$	All hijacking samples of class h
Φ	Feature extractor (part of the local model F).
$\Phi(x_h)$	A feature extracted from x_h
Φ_y	A anchor feature of the class y with 100% confidence

B. Computing Cloak Perturbations

But how do we determine what perturbations, namely cloaks, to apply to the hijacking samples? An effective cloak could cause the global model to associate the hijacking samples with features similar to the original samples. Intuitively, the closer the features extracted from the hijacking samples are to the original samples, the higher the probability that the global model will accurately identify the hijacking samples. In the following, we describe the detailed design to computing cloaks. For presentation purposes, we summarize the notations in Table I.

Cloaking to Minimize Feature Deviation. Formally, given a hijacking sample x_h and a original class y, our ideal design is to find a cloak perturbation δ that minimizes the deviation between the sample's feature $\Phi(x_h \oplus \delta)$ and the anchor feature Φ_y:

$$\min_{\delta} \mathit{Dist}\left(\Phi\left(x_h \oplus \delta\right), \Phi_y\right), \tag{2}$$

Where *Dist* calculates the distance of two feature vectors. Once the adversary succeeds in finding a cloak perturbation δ and adds it to the hijacking sample x_h, the cloaked sample $x_h \oplus \delta$ will be identified and extracted with anchor feature Φ_y, which the model then predicts as the original class y.

Class-specific Cloaking. When generating cloaks for a hijacking task, the adversary will create class-specific cloaks based on their held hijacking dataset, i.e., δ is class-dependent. Specifically, depending on the held hijacking dataset, the adversary optimizes the cloak for a particular hijacking class; therefore, samples within this class can be extracted to the same or similar features. This ensures that the cloak exhibits a high degree of generalizability, allowing it to map new, unseen real-world hijacking samples from that class to similar features, thus predicting them to the same original class. Therefore, instead of producing individual cloaks for each hijacking sample, we propose creating a single, generalizable cloak for each hijacking class. For each hijacking class h, all samples $\mathcal{X}_h$ share a common cloak δ_h. To achieve this, the adversary will pair any hijacking sample $x_h \in \mathcal{X}_h$ with an anchor feature Φ_y. The optimization problem is formulated as follows:

$$\min_{\delta_h} \mathit{Dist}\left(\Phi\left(x_h \oplus \delta_h\right), \Phi_y\right), \quad \text{subject to } \forall x_h \in \mathcal{X}_h. \tag{3}$$

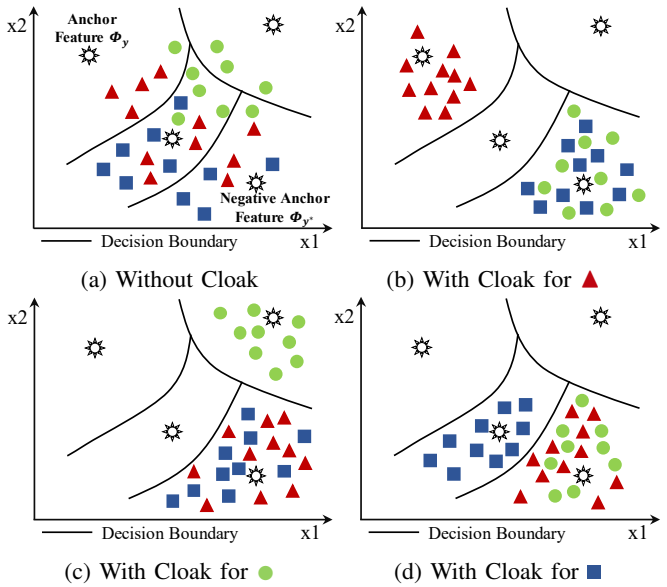


Fig. 3: An illustration of the class-specific cloaking. The background is the decision boundary of the original task. The colored points represent the features of hijacking samples. The negative anchor feature Φ_{y^*} is fixed in the lower right region.

This class-specific cloak optimization starts with $\delta = 0$ (no perturbation) and iteratively updates δ_h on all hijacking samples $\mathcal{X}_h$. The adversary then finds a cloak perturbation δ_h and adds it to any hijacking sample $x_h \in \mathcal{X}_h$. The cloaked samples will be predicted to the original class y. In this way, the adversary establishes a class mapping $h \Rightarrow y$. To establish another class mapping, the adversary performs the above optimization again and finally searches for a new cloak.

The above implementation, however, tends to search for overlearned cloaks so that the model only responds to cloaks and ignores the hijacking samples they cloak. In other words, given an unseen hijacking sample for query, adding a different cloak to it will be predicted as a different original class. This poses a challenge to the adversary to choose a suitable cloak for it. To address this, we introduce the negative anchor feature Φ_{y^*} (Figure 3a), which is fixed for any hijacking samples. As illustrated in Figure 3, the cloak δ_h not only establishes a class mapping $h \Rightarrow y$, but also maps all hijacking samples beyond class h to the original class y^*, i.e., $\bar{h} \Rightarrow y^*$. Now, given an unseen hijacking sample x, it can be accurately classified as the original class y only when added with its associated cloak. However, if added with any other cloaks, it will *always* be classified as the original class y^*. This design enables the adversary to easily determine which cloak to add to a given query sample, as only one cloak will not result in the specific original class y^*.

Thus, besides Equation 3, we further pair any hijacking samples beyond class h, i.e., $x_{\bar{h}} \in \mathcal{X}_{\bar{h}}$, with the negative anchor feature Φ_{y^*} and minimize the distance between $\Phi(x_{\bar{h}} \oplus \delta_h)$

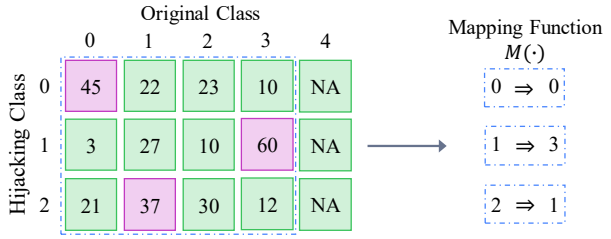


Fig. 4: The greedy-based class mapping. The number of hijacking samples is the same for each hijacking class (i.e., each row). Each cell's value indicates the count of hijacking samples from that hijacking class mapped to the original class. NA indicates this original class is used for negative anchor features and not assigned to a particular hijacking class.

and Φ_{y^*}:

$$\min_{\delta_h} \left(\mathit{Dist}\left(\Phi\left(x_h \oplus \delta_h\right), \Phi_y\right) + \lambda\, \mathit{Dist}\left(\Phi\left(x_{\bar{h}} \oplus \delta_h\right), \Phi_{y^*}\right)\right), \quad \text{subject to } \forall x_h \in \mathcal{X}_h \text{ and } \forall x_{\bar{h}} \in \mathcal{X}_{\bar{h}}. \tag{4}$$

The coefficient λ balances the two *Dist*, and is set to 1.2 in our implementation. The second *Dist* actually serves as a penalty for the cloak perturbation, to reduce overlearning.

C. HijackFL Pipeline

After introducing the core component of HijackFL, we now present the entire attack pipeline. The attack pipeline can be divided into four stages: Class Mapping, Anchor Features Computing, Cloaks Computing, and Executing. Note that the first three steps occur during a specific mid-training round, where the adversary exploits only their hijacking dataset and local model (fetched from the global model), without needing the original dataset. The final step happens when the adversary attacks the deployed global model.

In the following section, we consider a simple case where *the adversary's hijacking dataset comprises three classes ($h \in 0, 1, 2$), and the global model's original dataset comprises four classes ($y \in 0, 1, 2, 3$).*

Class Mapping. During this stage, the adversary needs to predefine a class mapping between the hijacking classes h and the original classes y. To this end, we define a hard-coded mapping function $M(\cdot)$ that maps a class from h to y. In the above case, $M(\cdot)$ may be defined to assign the first 3 original classes in a one-to-one mapping to the 3 hijacking classes, as adopted by Salem et al. [21].

Nevertheless, it's plausible that hijacking samples from certain classes may exhibit a stronger inclination to be associated with a particular original class, rather than being equally inclined towards all original classes (verified in Section Section V-D). Motivated by this, we propose a greedy-based class mapping approach aimed at facilitating the optimization of cloaks.

Specifically, we feed equal-sized hijacking samples from each class to the local model and record the frequency of their mapping to each original class. See an illustration in Figure 4. Notably, we exclude the last original class, which serves as y^* for negative anchor feature (used later), and focus solely on the remaining original classes. We then identify the hijacking class that maps to the largest number of original classes, e.g., $1 \Rightarrow 3$ shown in Figure 4. Subsequently, we exclude the selected hijacking and original classes, repeating this process with the remaining classes until all hijacking classes are assigned. This results in sequences such as $0 \Rightarrow 0$ and ultimately $2 \Rightarrow 1$. Thus, the class mapping function $M(\cdot)$ is established as $[0 \Rightarrow 0, 1 \Rightarrow 3, 2 \Rightarrow 1]$. See further details in Appendix Figure 17.

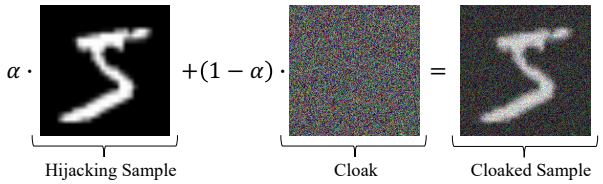


Fig. 5: The convex combination of the hijacking sample and the cloak.

Anchor Features Computing. After determining the class mapping $M(\cdot) = [0 \Rightarrow 0, 1 \Rightarrow 3, 2 \Rightarrow 1]$ and negative original class $y^* = 4$. The adversary is currently preparing anchor features that significantly contribute to predicting each assigned original class. To do this, for the assigned original class y, we initialize a random sample r from Gaussian noise and feed it the local model F to obtain a probability $F_y(r)$ of its class y. We then maximize the probability by optimizing the random sample as follows:

$$\min_r -\log F_y(r) \tag{5}$$

We optimize this loss with Adam and terminate until the probability is over a threshold. After we obtain an optimized sample, we feed it into the local model and extract its features from the intermediate layer, i.e., the anchor feature Φ_y we aim to obtain. Note that the threshold should be set very close to 1; only then can we claim that the searched anchor feature indeed significantly contributes to the prediction of class y. In our implementation, we set the threshold to 0.99. In this way, the adversary access the anchor features $\Phi_{y\in[0,1,2,3]}$: 3 anchor features and 1 negative anchor feature.

Cloaks Computing. For each hijacking class h, we initialize a cloak δ_h from Gaussian noise and transform it using a sigmoid function σ to ensure its values align with the hijacking samples, i.e., $255 \times \sigma(\delta_h) \in [0, 255]$. We then gather all hijacking samples of this class and take a convex combination as follows:

$$x_h \oplus \delta_h = \alpha x_h + (1-\alpha)\delta_h \quad \text{subject to } \forall x_h \in \mathcal{X}_h. \tag{6}$$

We use a parameter $\alpha \in [0, 1]$ to balance the weight/influence of x_h and δ_h on cloaked samples. Figure 5 present an illustration (see more visual examples in Appendix Figure 16). In our implementation, we set α to 0.5. We also study the effect of α by varying it in Section V-E.

The adversary then randomly selects equal-size hijacking samples beyond class h, i.e., $x_{\bar{h}}$, and obtains their cloaked versions by adding the same cloak δ_h. We then feed both $x_h \oplus \delta_h$

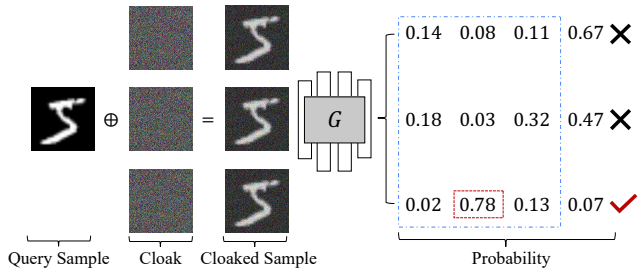


Fig. 6: The final execution of HijackFL against a remotely deployed global model.

and $x_{\bar{h}} \oplus \delta_h$ to the local model's feature extractor Φ, and minimize the two feature distances: $Dist\left(\Phi\left(x_h \oplus \delta_h\right), \Phi_y\right)$ and $Dist\left(\Phi\left(x_{\bar{h}} \oplus \delta_h\right), \Phi_{y^*}\right)$, as described in Equation 4. Lastly, to ensure the cloaked sample pixels remain in the correct range ($[0, 255]$), we transform the optimized cloak values into $[0, 255]$ by $255 \times \sigma(\delta_h)$. See the algorithm in Appendix Algorithm 1.

Executing. The adversary can now execute the attack on the remotely deployed global model. Given a new hijacking sample, the adversary cannot directly determine which cloak should be added to this sample. Instead, as shown in Figure 6, the adversary obtains 3 cloaked samples by adding 3 cloaks to the given sample respectively. The adversary then queries the global mode for each cloaked sample to obtain its 4-dimensional probability. The adversary observes only the first 3 dimensions of the probability, and selects the class with the highest probability as the final prediction, Finally, the adversary can determine its hijacking class by reversing the class mapping $M^{-}(\cdot) = [0 \Rightarrow 0, 3 \Rightarrow 1, 1 \Rightarrow 2]$.

IV. Experimental Setup

A. Datasets

We employ four well-established computer vision benchmark datasets: MNIST [2], CIFAR-10 [3], GTSRB [4], and TinyImageNet-100 [5] (see details in Appendix Section A). Note that due to the high imbalance of GTSRB's 43 classes, we select the top ten classes with the highest number of samples to create a more balanced dataset.

We use CIFAR-10 and TinyImageNet-100 as the original datasets, while MNIST and GTSRB are the hijacking datasets. The different combinations of these datasets are in Table II (See dataset sizes in Appendix Table XI and Table XII). Since the last original class should be fixed for negative pairs, when the original dataset is CIFAR-10, we randomly sample 9 classes from MNIST and GTSRB as the hijacking dataset. This is actually an inherent limitation of the model hijacking attack, i.e., the hijacking dataset should have less class labels than the original dataset. We will discuss this further in Section VI-E. Lastly, following Salem et al. [21], we rescale all datasets to the image size of 32×32, and convert the grayscale MNIST to three-channel images by repeating the same values in all channels.

TABLE II: Different combinations for the original and hijacking datasets.

TaskSet	Original Dataset	Hijacking Dataset
I	CIFAR-10 (10)	MNIST (9)
II	CIFAR-10 (10)	GTSRB (9)
III	TinyImageNet-100 (100)	MNIST (10)
IV	TinyImageNet-100 (100)	GTSRB (10)

B. Model Structures

As aforementioned, we focus on the computer vision classification tasks. Thus, following Salem et al. [21], we also use the popular image classification model as our target model, namely ResNet-18 [11], MobileNet-V2 [22], and ShuffleNet-V2 [15]. Each of these model architectures enjoys its own unique property and has been applied in a wide range of applications. Besides, all clients and the central server share the same model architecture.

C. Federated Learning Settings

We now detail the global settings of the server and the general local settings of the benign/malicious clients.

Global Settings. Following prior work [6], we set the number of training rounds to 200, and the total number of clients contributing to the global model is $n = 50$. In each round, the central server selects $m = 5$ clients (we also study selecting 10 and 20 clients in Section V-E), and each client is chosen with equal probability, ensuring that all 50 clients have been involved in one round after 10 training rounds. Furthermore, the central server averages the received updates from the selected 5 clients in each training round. The global learning rate is set to $\eta = 10$.

Local Settings. Each client controls the same-sized subset of the full original dataset, e.g., one client locally controls 2000 samples of the TinyImageNet-100 dataset (we also discuss non-IID data in Section VI-A). In each round, all the selected clients train the local model using the same training settings, i.e., an optimizer of SGD [8], a learning rate of 0.1, and a local training epoch of 2.

D. Hijacking Attack Settings

We configure the adversary to update their local model by fetching it from the global model at the 150th training round (hijacking round). The adversary then performs the first three steps: class Mapping, anchor feature computation, and cloak computation on this local model. The optimization of anchor features involves 500 iterations with a learning rate of 0.005. Similarly, cloak computation optimization involves 100 iterations with a learning rate of 0.005. These settings are determined based on observing convergence around these iterations during the optimization process.

E. Baseline Attacks

The core idea of existing work [21], [24] is to utilize data poisoning to achieve the goal of hijacking. We generalize

this data poisoning approach to the federated learning setting as a baseline attack. In addition, we further adapt the data poisoning approach to model poisoning based on the FL backdoor idea proposed by Bagdasaryan et al. [6].

Data Poisoning. The adversary can poison the training dataset, achieving the goal of hijacking when the model is trained on this poisoned data. Specifically, Salem et al. [21] introduce two attack methods. The first involves mixing the original dataset with the hijacking dataset and then using this poisoned dataset to train the target model. This achieves the upper bound on attack performance, yet causes a drawback: it can be easily detected by the model owner since samples in the original and the hijacking datasets can be significantly different. Thus, Salem et al. [21] introduce an additional model to transform the hijacking dataset into one that is visually similar to the original dataset, which is then mixed with the original dataset to train the target model. Si et al. [24] adopt a similar approach by using an additional model to transform the hijacking dataset. In this study, we apply both of these methods to the FL scenario, called Data Poison (naive) and Data Poison (transform). We emphasize that both methods are essentially data poisoning attacks aimed at altering the target model's parameters, while our attack avoids such modifications.

Model Poisoning. Model poisoning (also called model replacement) is first proposed in backdooring FL models by Bagdasaryan et al. [6]. This approach exploits the fact that FL enables the malicious client to directly affect the global model, eventually replaced by the poisoned model. Specially, the adversary can substitute the new global model G_{t+1} with a malicious model X in Equation 1:

$$X = G^t + \frac{\eta}{n} \sum_{i=1}^{m} \left(F_i^{t+1} - G^t\right) \tag{7}$$

As the global model converges, these deviations start to cancel out, i.e., $\sum_{i=1}^{m-1} \left(F_i^{t+1} - G^t\right) \approx 0$. Thus, the adversary can submit the model update as follows:

$$F_i^{t+1} = \frac{n}{\eta} X - (\frac{n}{\eta} - 1) G^t - \sum_{i=1}^{m-1} (F_i^{t+1} - G^t) \approx \frac{n}{\eta}(X - G^t) + G^t \tag{8}$$

This adversary scales up the weights of the malicious model X by $\gamma = \frac{n}{\eta}$ to ensure that the backdoor survives the averaging and the global model is replaced by X. In this work, we adapt the two aforementioned data poisoning methods to model poisoning, called Model Poison (naive) and Model Poison (transform). Specifically, we make a strong assumption for the adversary that they know the exact FL settings, i.e., $n = 50$ and $\eta = 10$. Thus, we set the scale-up parameter $\gamma = \frac{n}{\eta} = 5$.

F. Evaluation Metrics

Following prior works [21], [24], we adopt two metrics, namely utility and attack success rate.

Utility. Utility measures how closely the performance of the hijacked global model matches that of a clean (non-hijacked) global model on the original testing dataset (Requirement 1).

Attack Success Rate. The attack success rate measures the hijacked global model accuracy on the hijacking testing dataset (Requirement 2). A higher attack success rate indicates a more powerful attack.

V. Experimental Results

A. Attack Performance

Utility. We start by evaluating the utility of the hijacked global model on the original testing dataset. Table III illustrates the utility of both the hijacked global models and the clean ones trained solely with original datasets. We observe that data poisoning (naive) and model poisoning (naive) induce slight side effects on the global model's utility to different degrees. For instance, clean ResNet-18 achieves 88.03% utility on TaskSet-I, while hijacked ResNet-18 achieves 86.39% utility by model poison (naive). Conversely, data poison (transform), and particularly model poison (transform), exhibit a more significant negative impact on the global model's utility. For example, on TaskSet-III, clean MobileNet-V2 achieves 44.72% utility, whereas hijacked MobileNet-V2 achieves 38.84% and 29.94% utility by data poison (transform) and model poison (transform), respectively. These findings align with those of Salem et al. [21], where transforming the hijacking dataset leads to relatively poorer performance. We attribute these observations to the fact that transforming the hijacking dataset visually aligns it more closely with the original dataset, thereby complicating model optimization and resulting in decreased utility. In particular, since the model poison further amplifies poisoned local model parameters by the scale-up $\gamma = \frac{n}{\eta}$, it leads to a particularly obvious decrease in utility.

Encouragingly, HijackFL does not modify the local model, but only exchanges clean model parameters with the central server. Thus, HijackFL does not affect the utility performance of the global model. Overall, HijackFL can guarantee superior utility performance over baseline attacks.

Attack Success Rate.

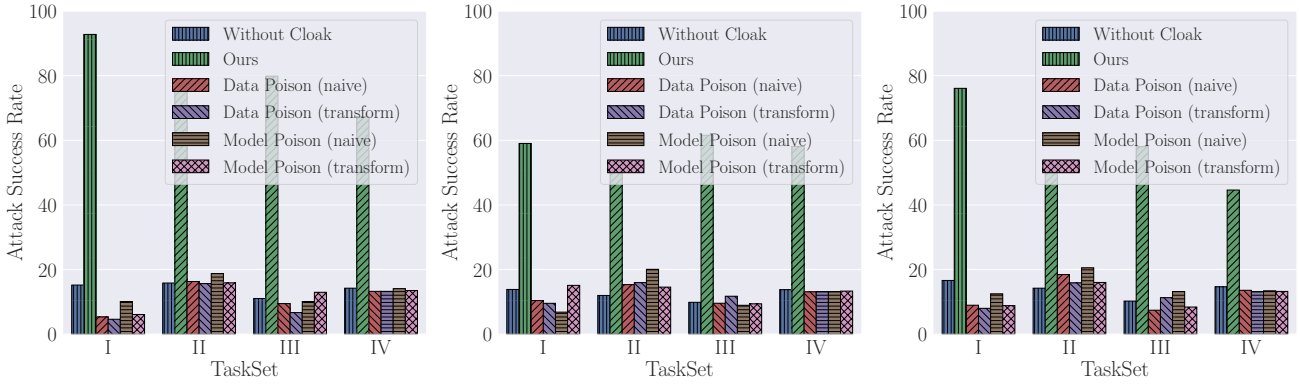
We present the results in Figure 7. As we can see, HijackFL achieves a very high level of attack success rate and is far superior to baseline attacks. For example, on ResNet-18, HijackFL for TaskSet-I achieves an attack success rate of 92.75%, while all baseline attacks achieve only around 10%. The reason is that the baseline attacks achieve their goal by submitting poisoned model parameters, thereby injecting the hijacking effect into the global model. However, as the global model undergoes more FL training rounds and averages only benign local model parameters, the hijacking effect will be reduced or even removed. However, since HijackFL does not rely on injecting the hijacking effect into the global model, it has a high probability of achieving remarkable attack performance even as the global model undergoes more FL training rounds.

B. Fluctuation of Utility

Note that all evaluations, except those in this section, are performed on the final global model: the adversary mount cloak computation/data poison/model poison at the hijacking round (i.e., 150th) and evaluates their attacks on the final global model trained over the entire 200 rounds.

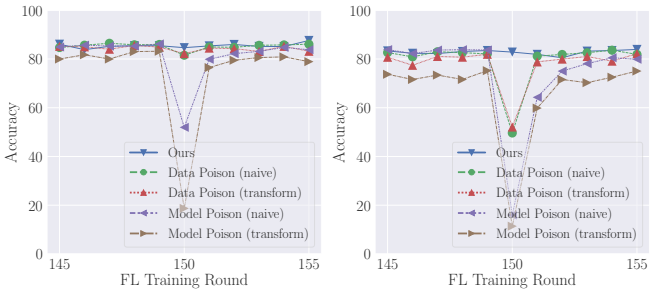
TABLE III: The utility of the clean global models and the hijacked global models on the clean test set. Note that current FL techniques struggle to achieve high utility on some benchmark datasets. These *low utilities* are also observed in other papers [9], [28].

Model Architecture	TaskSet	Clean Model	Hijacked Global Model				
			Data Poison (naive)	Data Poison (transform)	Model Poison (naive)	Model Poison (transform)	Ours
ResNet-18	I	0.8803	0.8822	0.8639	0.8639	0.8774	0.8803
	II	0.8803	0.8803	0.8713	0.8809	0.8483	0.8803
	III	0.5460	0.5362	0.5390	0.5456	0.3806	0.5460
	IV	0.5460	0.5344	0.5328	0.5436	0.3102	0.5460
MobileNet-V2	I	0.8592	0.8459	0.8041	0.8554	0.7466	0.8592
	II	0.8592	0.8597	0.8247	0.8529	0.8121	0.8592
	III	0.4472	0.4592	0.3884	0.4350	0.2994	0.4472
	IV	0.4472	0.4598	0.4218	0.4564	0.2564	0.4472
ShuffleNet-V2	I	0.8746	0.8716	0.8489	0.8674	0.8174	0.8746
	II	0.8746	0.8726	0.8765	0.8730	0.8438	0.8746
	III	0.5178	0.5188	0.4822	0.5150	0.3334	0.5178
	IV	0.5178	0.5132	0.5028	0.5252	0.3752	0.5178



(a) ResNet-18 (b) MobileNet-V2 (c) ShuffleNet-V2

Fig. 7: The attack success rates of HijackFL, the baseline attacks of data poison, and model poison.



(a) ResNet-18 (b) MobileNet-V2

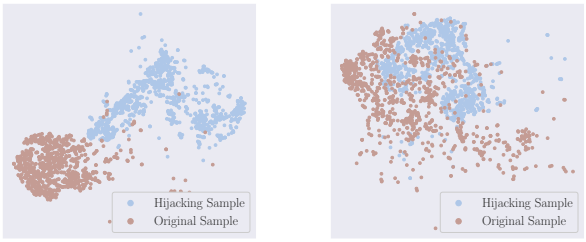
Fig. 8: The utility of the global model before, during, and after the hijacking round (i.e., 150th).

In this section, we investigate the impact of these attacks on the global model at the hijacking round. Figure 8 illustrates the utility of the global model before, during, and after the hijacking round. We can observe a noticeable fluctuation in utility when baseline attacks execute the poison operation. Such fluctuations can easily alert the server, leading to detection by the central server. This observation stems from the fact that baseline attacks modify local model parameters, which are then transmitted to the central server. Consequently, the global model is affected, resulting in decreased accuracy on the original dataset. In subsequent rounds, when only benign local model parameters are submitted to the central server, the adversary's hijacking impact is eliminated. As a result, utility levels rebound to a high standard. This also explains why these baseline attacks only achieve a 5% $\sim$ 22% attack success rate when targeting the final global model, as depicted in Figure 7. In contrast, in HijackFL, the adversary simply trains the local model on their original dataset as usual and submits these clean model parameters to the server. As a result, the adversary can always guarantee the utility of the global model.

C. Feature Visualization

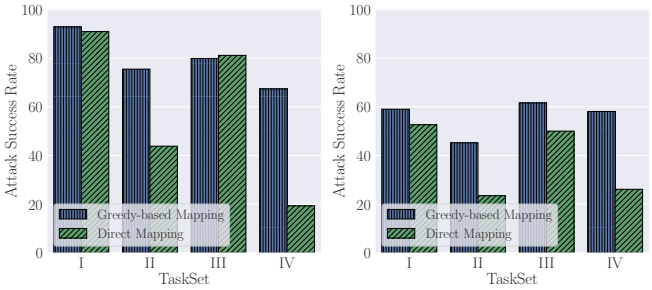
Recall that the key idea of HijackFL is to add cloaks to the hijacking samples, which allows for the identification and extraction of features that are similar to the original samples. Besides the quantitative analysis, in this section, we will verify the effectiveness of HijackFL from a visualization aspect.

Concretely, in TaskSet-I, where CIFAR-10 is the original dataset, and MNIST is the hijacking dataset, we randomly select some hijacking samples from a single hijacking class and an equivalent number of original samples from its mapped original class. We feed these samples into the final global model and extract features from the middle layer of the model. Then, we utilize t-SNE [26] to process these features into a 2D space and visualize them in Figure 9a. As we can see, the



(a) Without Cloak (b) With Cloak

Fig. 9: The visualization of hijacking/original sample features mapped to 2D space using t-SNE.



(a) ResNet-18 (b) MobileNet-V2

Fig. 10: The attack success rate of HijackFL under the impact of the class mapping, i.e., greedy-based class mapping and direct class mapping.

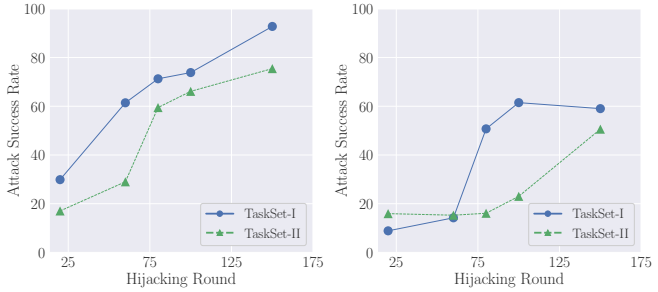
features from hijacking and original samples have been clearly mapped into separate regions.

In contrast, after we add the corresponding cloak to these hijacking samples, we then feed the cloaked samples and the original samples to extract their features and map these features in 2D space using t-SNE. Remarkably, the visualization shown in Figure 9b demonstrates a high degree of overlap between the features of cloaked samples and the original samples. This intriguing finding suggests that the cloaked samples are now associated with features similar to the original samples, effectively leading the global model to classify these cloaked samples to this original class.

D. Impact of Class Mapping

Recall that our attack consists of four steps, where the first stage is class mapping. In this stage, the adversary predefines a class mapping between the hijacking class and the original class. Since we believe that the hijacking samples of certain classes will be more inclined to be associated with a certain original class. Therefore, we introduce a greedy-based class mapping to facilitate the optimization of effective cloaks. In this section, we investigate the impact of the class mapping on the attack performance, i.e., whether it is really as we conjectured.

In particular, we discard our proposed greedy-based class mapping strategy and instead assign the first several original classes to the hijacking classes in a one-to-one mapping,



(a) ResNet-18 (b) MobileNet-V2

Fig. 11: The attack success rate of HijackFL under the impact of the hijacking round.

similar to the strategy adopted by Salem et al. [21]. For example, in TaskSet-III, where TinyImageNet-100 serves as the original dataset and MNIST as the hijacking dataset, so we directly assign the first 10 classes of TinyImageNet-100 to the 10 classes of MNIST in a one-to-one mapping (called direct mapping). We then keep our (HijackFL) other settings unchanged and evaluate its attack performance. Figure 10 shows a comparison of HijackFL with greedy-based class mapping and direct class mapping. We can find that in most cases, greedy-based class mapping outperforms direct class mapping in terms of attack performance. For example, in TaskSet-II on ResNet-18, the attack with greedy-based class mapping achieves an attack success rate of 75.45%, while the attack with direct class mapping achieves an attack success rate of only 43.85%. However, we do find an exception where the direct mapping achieves a similar level of performance as greedy-based attacks in TaskSet-III on ResNet-18. We attribute this to the fact that the hijacking samples of the first few classes are already more inclined to be associated with the first few original classes. Thus, the direct class mapping also achieves a similar attack performance as the greedy-based class mapping.

In summary, the above results show that the greedy-based class mapping we designed has excellent performance, and also verify our claim that hijacking samples of certain classes will indeed be more inclined to be associated with a certain original class.

E. Ablation Study of Hyperparameters

Hijacking Round. We now explore the impact of the hijacking round, denoting the FL training round where the adversary mounts cloak computation. In the above evaluations, we default the adversary to mount cloak computation at the 150th training round (note that only once). However, in real-world scenarios, the adversary may not have control over their participation round, as the central server might follow specific protocols, such as randomly selecting clients. Hence, it is essential to investigate how the hijacking round impacts our HijackFL.

To evaluate the effect of attack timing, we configure the adversary to start cloak computation at rounds 10, 50, 80, 100, and 150 in a 200-round training process. As shown in

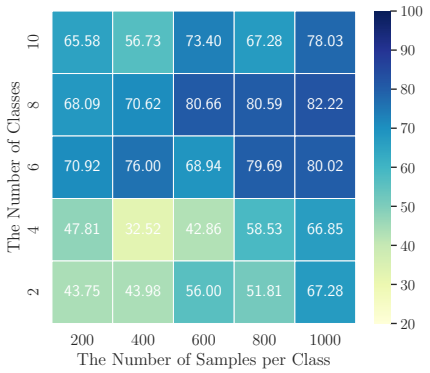


Fig. 12: The attack success rate under the impact of the complexity of the hijacking task. The y-axis represents the number of classes, and the x-axis represents the number of samples per class.

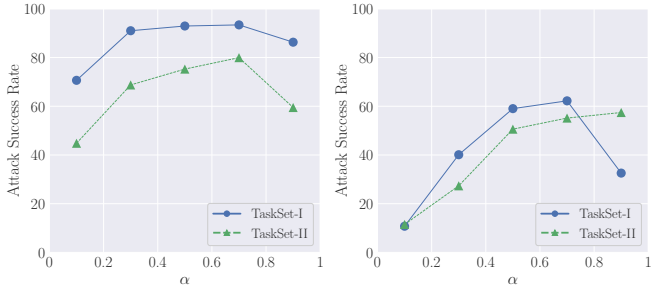
Figure 11, performance peaks when the global model nears convergence. However, the attack remains effective even when launched at much earlier rounds. This tolerance to timing reduces the impact of random client sampling in two ways. First, the probability that the adversary is never selected during later, optimal rounds is negligible. Second, even when the attack is triggered well before convergence, HijackFL still learns a valid hijacking mapping rather than failing. These results show that our method remains effective throughout training and mitigates the risks posed by server-controlled sampling.

Complexity of Hijacking Task. We here study the impact of hijacking task complexity. Specifically, we consider the hijacking task complexity from two perspectives: the number of classes and the number of samples per class. For the original dataset TinyImageNet-100 and the hijacking dataset GTSRB, we reconstruct the GTSRB dataset by adjusting the number of classes from 2 to 10. Additionally, we vary the number of GTSRB samples per class within the range of 200 to 1000. We conduct extensive experiments to simultaneously tune these two hyper-parameters and report the results in Figure 12. Through investigation, we make the following observations.

- The attack performance tends to increase with the number of samples per class.
- The attack performance initially increases with the number of classes and then decreases, peaking around 6 or 8 classes.

The first observation is rooted in the fact that having more hijacking samples per class results in the learning of a more effective, general, and stable cloak, consequently leading to a higher attack performance. Concerning the second observation, we speculate that it is linked to intricate interactions between the global model architectures and the complexity of the original task. We leave the in-depth exploration of such intricate interactions as a future work.

Number of Hijacking Task. We now discuss the impact of the number of hijacking tasks: hijacking the global model with multiple datasets. Importantly, owing to our innovative design,



(a) ResNet-18 (b) MobileNet-V2

Fig. 13: The attack success rate of HijackFL under the impact of α

HijackFL distinguishes itself by not interacting with the global model's parameters, in contrast to the baseline attacks of data poison and model poison. Consequently, the adversary can mount cloak computation based on the same global model, irrespective of the number of hijacking datasets. This approach enables the adversary to possess distinct sets of cloaks for different hijacking datasets, offering flexibility when launching attacks on the final global model as they can select the suitable set of cloaks.

In contrast, it is expected that the baseline attacks of data poison and model poison will be influenced by the number of hijacking tasks. More hijacking datasets are likely to result in submitted model updates deviating further from a clean one, thereby causing a larger utility drop in the global model.

Balance Parameter α. Recall that we employ a convex combination of the hijacking samples and the cloak: $x_h \oplus \delta_h = \alpha x_h + (1-\alpha)\delta_h$. In this expression, the parameter $\alpha \in [0,1]$ governs the balance between the weight or influence of x_h and δ_h on the cloaked samples. In this section, we investigate the impact of α. Note that we use a default value of 0.5 for α

Throughout this paper, we use the default value of 0.5 for α. We now vary the parameter α from 0 to 1 and present the results in Figure 13. We clearly observe a trend where the attack performance initially increases and then decreases in most cases. The peak attack performance is attained when α falls within the range of 0.5 to 0.7. This observation motivates the adversary to either carefully select the optimal α or simply use a value within the range of 0.5 to 0.7.

Number of Cloak. In our previous evaluations, we employ a single generalized cloak for all hijacking samples within the same class. In this section, we explore the impact of the number of cloaks. In particular, we generate only one cloak for all hijacking samples, irrespective of their respective hijacking classes. Table IV shows evaluation results on ResNet-18 and MobileNet-V2 with all TaskSets. We can clearly observe that employing just one cloak for the entire hijacking classes fails to effectively achieve the hijacking goal. This outcome is understandable, as it becomes considerably more challenging for a single cloak to establish class mappings from various hijacking classes to various original classes compared to the simpler task of mapping one hijacking class to one original class.

TABLE IV: Comparison between HijackFL that uses only one cloak for the entire hijacking dataset and HijackFL that employs one cloak for each hijacking class (i.e., multi cloaks).

TaskSet	ResNet-18		MobileNet-V2	
	One Cloak	Multi Cloak	One Cloak	Multi Cloak
I	0.1246	0.9275	0.1246	0.5903
II	0.1200	0.7545	0.1200	0.5060
III	0.1135	0.7984	0.1135	0.6156
IV	0.1111	0.6733	0.1111	0.5808

TABLE V: The attack success rate of HijackFL under different number of clients in each training round.

TaskSet	ResNet-18			MobileNet-V2		
	5	10	20	5	10	20
I	0.9275	0.9279	0.8958	0.5903	0.7840	0.8975
II	0.7545	0.8007	0.7820	0.5060	0.7840	0.7567
III	0.7984	0.8829	0.9115	0.6156	0.7260	0.8037
IV	0.6733	0.6733	0.6940	0.5808	0.5841	0.7071

We further consider the feasibility of generating a specific cloak for each hijacking sample. We assert that this approach is impractical for several reasons. Firstly, optimizing a cloak for each individual sample limits its effectiveness to that sample alone, rendering it ineffective for new, unseen hijacking instances. Moreover, in hijacking datasets with numerous samples, generating a unique cloak for each becomes unwieldy for the adversary, making selection for new samples challenging. Thus, adversaries should optimize cloaks based on a data domain (e.g., hijacking samples within the same class) with the aim of achieving generalizability.

Number of Clients. In the previous evaluation, the server selects 5 clients per round by default. Here, we investigate the impact of varying the number of clients participating in each round. Specifically, among 50 clients, we increase the number to 10 and 20 clients randomly chosen to participate in each round. All other settings remain the same as described in Section IV. As shown in Table V, HijackFL consistently achieves a high ASR when 10 and 20 clients are involved in each training round. Notably, we observe a general trend where an increase in the number of clients correlates with a higher attack success rate. This suggests a potential advantage for HijackFL, as real-world scenarios likely involve many more clients participating in training.

VI. Discussion

A. Non-IID Data Among Clients

Previous evaluations have assumed a default setting where the original dataset is evenly distributed among clients. However, in the real world, data is usually non-IID among clients, with different sample sizes and class distributions.

In this reality, we emphasize that this does not affect HijackFL. Recall that HijackFL searches for pixel-level perturbations based on the local model (i.e., fetched from the global model). In other words, as long as the adversary can receive the global model locally, they can compute the cloak perturbation.

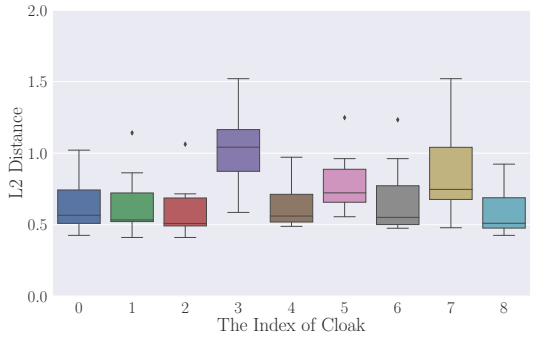


Fig. 14: The L2 distance between the cloaked sample's feature and the negative anchor feature. The x-axis represents the index of different cloaks added to the hijacking sample.

TABLE VI: The attack success rate of HijackFL against various anomaly defenses.

TaskSet	ResNet-18				MobileNet-V2			
	No	Defence-I	Defence-R	Defence-B	No	Defence-I	Defence-R	Defence-B
I	0.9275	0.1107	0.8106	0.1111	0.5903	0.1109	0.5775	0.1088
II	0.7545	0.4390	0.6883	0.1148	0.5060	0.1707	0.4715	0.1148
III	0.7984	0.1009	0.6185	0.0990	0.6156	0.1009	0.5017	0.0985
IV	0.6733	0.2571	0.5775	0.1031	0.5808	0.2778	0.4147	0.1035

This process is independent of the local dataset, so non-IID data does not negatively affect the attack performance.

B. Possible Defenses

Since detecting attacks during the FL training phase is impossible, we explore two defenses in the FL prediction phase: feature-based anomaly detection and adversarial example detection.

Feature-based Anomaly Detection. The intuition behind HijackFL's execution is straightforward: the adversary adds all cloaks to a hijacked sample and submits these cloaked versions to the final global model. Only one cloak will map the sample to its original class y, while the rest will map it to a negative class y^*. Consequently, one cloaked sample will have a feature significantly further from the negative anchor feature than the others, as shown in Figure 14.

Based on above, we propose a feature-based anomaly detection method: (1) Generate anchor features for each class; (2) Extract features from cloaked samples; (3) Compute the L2 distance between each sample and its class anchor; (4) Flag the set as a hijacking attempt if any distance exceeds a threshold (see Table IX). Table VI presents results for two scenarios. In the ideal case (Defense-I), where hijacking samples are queried together, the defense is effective as features can be compared directly. In the realistic case (Defense-R), the adversary mixes hijacking and clean samples or staggers queries, making feature comparison harder and allowing high attack success.

Adversarial Example Defenses. HijackFL adds cloaks like adversarial perturbations, so we apply Feature Squeezing [27], which detects adversarial inputs by compressing them and comparing prediction changes. Significant differences indicate adversarial behavior, allowing rejection based on a threshold.

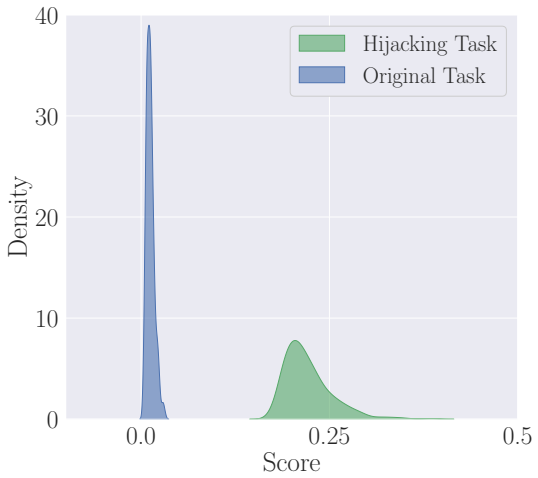


Fig. 15: Kernel density estimation of TV scores for the original task and the hijacking task under Task-I with ResNet-18.

TABLE VII: The defensive performance of adversarial example defense, i.e., Feature Squeezing.

TaskSet	ResNet-18				MobileNet-V2			
	Threshold 1		Threshold 2		Threshold 1		Threshold 2	
	Utility	ASR	Utility	ASR	Utility	ASR	Utility	ASR
I	0.5736	0.1223	0.8641	0.8934	0.6126	0.1397	0.8423	0.5304
II	0.5776	0.1267	0.8633	0.7128	0.6090	0.1125	0.8428	0.4260
III	0.2390	0.1179	0.4966	0.5771	0.3140	0.1081	0.4482	0.5417
IV	0.2402	0.1206	0.4966	0.5543	0.3108	0.0894	0.4484	0.4452

TABLE VIII: Comparison of Computational Cost (Hours) Between HijackFL and Direct Training Using MobileNetV2

		TaskSet III	**TaskSet IV**
HijackFL	Class Mapping	0.0035	0.0008
	Anchor Features	0.0103	0.0102
	Cloaks Computing	2.5612	0.8132
Directly Training		0.5910	0.4284

Since Feature Squeezing also affects clean inputs, we report both utility and attack performance (Table VII). A low threshold (Threshold 1) detects hijacking samples but causes many false positives, harming utility. A high threshold (Threshold 2) preserves utility but fails to detect attacks, allowing HijackFL to remain effective.

Pre-inference Defenses. A practical defense against parasitic computing is to validate input legitimacy before inference to block unauthorized requests. Notably, HijackFL does not require visual similarity between hijacking inputs and inputs from the original task. As a result, although parasitic inputs can produce valid predictions, their statistical properties in raw pixel space often differ significantly from legitimate data.

To quantify this difference, we use Total Variation (TV) as an illustrative metric. As shown in Figure 15, the TV distributions of the original and hidden tasks are clearly separable. Inputs from the hidden task typically have higher TV values, indicating stronger high-frequency components or unnatural textures. Based on this observation, we propose a lightweight defense. If an input's TV value deviates substantially from the normal range, the server suppresses standard inference and returns a stochastic or default output. The effectiveness of this mechanism (Defense-B) is demonstrated in Table VI.

Combining this approach with behavior-based anomaly detection can further strengthen oversight. Commercial exploitation of parasitic capabilities often appears as high-frequency and low-diversity batch requests that deviate from normal user behavior. We therefore recommend deploying an anomaly monitoring system that tracks meta-features such as request frequency and input similarity. This system can automatically flag suspicious accounts and trigger manual compliance audits of their usage patterns.

C. Comparison with Centralized Setting

Technically, HijackFL can extend to centralized settings. However, it requires access to model parameters to extract anchor feature representations for the hijacking classes. In centralized scenarios, these parameters are usually protected by black-box APIs, which makes this assumption unrealistic. In contrast, federated learning distributes the full global model to all clients. This design gives malicious clients direct access to the model's representation space and allows cloak optimization without extra intrusion. As a result, HijackFL exposes a key vulnerability of federated learning: model transparency, which supports collaborative training, also enables covert resource hijacking.

D. Computational Cost of Cloak

Regarding efficiency, cloak generation incurs a local computational cost (See Table VIII), but this cost is a one-time offline investment. In contrast to independent deployment, which requires ongoing maintenance and infrastructure expenses, HijackFL transfers these costs entirely to the victim. As a result, the initial overhead is amortized over many online inference queries, allowing the adversary to exploit high-performance AI capabilities with negligible marginal cost.

E. Limitation

Task Scale and Efficiency. A key limitation of HijackFL is the scale and complexity of the hijacking task. First, the hijacking task usually involves fewer classes than the primary task, because one original class must be reserved for negative samples. Second, the attack relies on existing feature representations and does not update model parameters, so the hidden task is limited by the fixed capacity of the primary feature space. In addition, the current design requires multiple queries to infer the final label, which reduces efficiency as the number of hijacking classes grows. Future work will study universal cloak methods and improved relabeling strategies to enable single-query inference for larger and more complex hijacking tasks.

Noticeable Visual Perturbation. The second limitation is that the cloaked samples queried by the adversary to the global model may exhibit noticeable visual differences from

TABLE IX: The threshold of the feature-based anomaly defense..

TaskSet	ResNet-18	MobileNet-V2
I	0.6	0.6
II	0.6	0.6
III	0.6	0.6
IV	0.6	0.6

TABLE X: The threshold of Feature Squeezing.

TaskSet	ResNet-18		MobileNet-V2	
	Threshold 1	Threshold 2	Threshold 1	Threshold 2
I	3	15	3	15
II	3	15	3	15
III	20	60	20	60
IV	20	60	20	60

the original samples, making them susceptible to detection by the defense mechanisms. One potential solution is to not only minimize the deviation of features between the cloaked samples and original samples but also minimize the deviation of pixels in the image space. This approach aims to make the hijacking samples, added with cloaks, resemble original samples in both image space and feature space. However, we speculate that introducing this additional objective for cloaks may diminish its overall effectiveness. We will explore a more powerful attack against FL models to address this limitation.

VII. Conclusion

In this work, we present the first model hijacking attack against federated learning, namely HijackFL. In this attack, an adversary (i.e., a malicious client) repurposes a global model designed for a specific task to perform an adversary-defined hijacking task. The core design of HijackFL is to introduce pixel-level perturbations (cloaks) to hijacking samples, enabling their extraction and identification in a manner similar to the original samples within the feature space. Consequently, these samples are accurately classified into the corresponding original classes. We conduct extensive evaluations on four benchmark computer vision datasets and three widely used model architectures. Empirical findings demonstrate that our attack attains a high level of attack success rate without compromising model utility, meeting the two requirements for hijacking a federated learning model. Furthermore, we explore two possible defenses that can detect hijacking samples in the FL prediction phase.

Acknowledgments

This work was supported by National Natural Science Foundation of China under Grant (No. 62502276), Key R&D Program of Shandong Province, China (No. 2025CXPT085).

Appendix

A. Datasets

We employ four well-established computer vision benchmark datasets: MNIST [2], CIFAR-10 [3], GTSRB [4], and TinyImageNet-100 [5]. Here's a brief introduction to each dataset:

MNIST. MNIST is a gray image dataset consisting of handwritten digits. It contains 60,000 training images and 10,000 test images, with each image representing a single digit from 0 to 9.

CIFAR-10. CIFAR-10 is a dataset comprising 60,000 color images divided into 10 classes. Each image in CIFAR-10 belongs to one of the following categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck.

GTSRB. GTSRB is an image collection consisting of 43 traffic signs. It consists of over 51,839 color images, with 39,209 used for training and 12,630 used for testing. Due to the high imbalance of each class, we select the top ten classes with the highest number of samples to create a more balanced dataset.

TinyImageNet-100. TinyImageNet-100 is a subset of the larger ImageNet dataset, containing 100 object classes with 500 training images and 50 validation images per class.

B. The Convergence of Cloak Optimization

In this section, we delve into the convergence properties of cloak optimization. Equation 4 defines an optimization problem aimed at minimizing the weighted sum of distances between two features. Specifically, we employ the L2 distance as our loss function to quantify these distances.

Loss Function Convexity. The L2 loss function is a squared loss function. Since the squared function is convex, the L2 loss function inherits this property. Convex functions exhibit desirable optimization properties, ensuring that the optimal solution is globally optimal and avoids local minima. This helps the optimization algorithm to converge to the global optimum.

Gradient Continuity. This gradient is continuous because the derivative of the squared function is a linear function that can be derived everywhere. The continuity of the gradient ensures that the gradient information is reliable in the process of finding the optimal solution of the optimization algorithm, and there will not be a situation where the gradient does not exist or is not continuous.

Convergence. Since the L2 loss function is convex and the gradient is continuous, optimization algorithms based on gradient descent can converge to the global optimum. The continuous gradient guides the optimization process toward the optimal solution along the descent direction until convergence. Importantly, convexity prevents the algorithm from getting trapped in local optima.

In summary, the L2 distance as a loss function has good mathematical properties, i.e., convexity and continuity of the gradient, which makes the optimization algorithm based on gradient descent able to converge to the global optimal solution. This is one of the important reasons why the L2 loss function is widely used in machine learning and optimization problems.

C. A Real-World Scenario

To illustrate the impact of HijackFL, we analyze a medical consortium collaborating on a retinopathy screening model. An adversary hijacks this model for unauthorized skin lesion classification via offline-trained cloaks. This parasitic behavior inflicts multi-dimensional harm: the server bears unauthorized operational costs and resource contention, leading to service delays for legitimate users and legal liability for hosting uncertified functions. Meanwhile, honest clients' contributions are misappropriated, exposing the consortium to accountability risks. Furthermore, by circumventing legal oversight and bypassing mandatory certifications, attackers can offer unauthorized services at significantly lower prices, thereby disrupting market order and stifling industrial innovation.

Algorithm 1: Cloaks Computing.

Input: Feature extractor Φ; Anchor features $\Phi_{y\in[0,1,2,3]}$; Hijacking samples of different classes $\mathcal{X}_{h\in[0,1,2]}$; Class mapping $M(\cdot)=[0\Rightarrow 0, 1\Rightarrow 3, 2\Rightarrow 1]$; Iteration T; Hyperparameter λ.

Output: A list **Cloaks**;

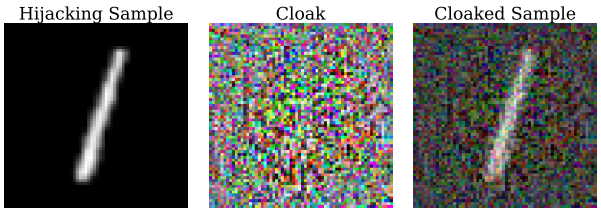
initialize an empty list: **Cloaks** = [];
set $y^* = 3$;
select negative anchor feature: Φ_{y^*};
for $h \in [0,1,2]$ **do**
  initialize cloak δ_h;
  select anchor feature $\Phi_{M(h)}$;
  for $t \leftarrow 0$ *to* T **do**
    select x_h from $\mathcal{X}_h$;
    select negative samples $x_{\bar{h}}$ from $\mathcal{X}_{\bar{h}}$;
    $loss_1$ = *Dist* $\left(\Phi\left(x_h \oplus \delta_h\right), \Phi_{M(y)}\right)$;
    $loss_2$ = *Dist* $\left(\Phi\left(x_{\bar{h}} \oplus \delta_h\right), \Phi_{y^*}\right)$;
    δ_h = $argmin(loss_1 + \lambda loss_2)$
  end
  add δ_h to **Cloaks**;
end
return **Cloaks**;

TABLE XI: Different combinations for the original and hijacking datasets used in *FL training phase*.

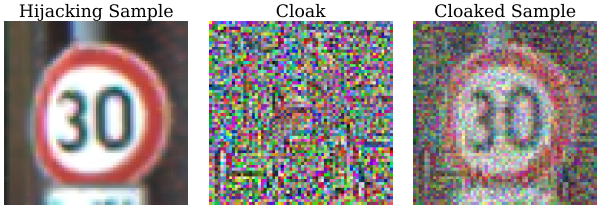
TaskSet	Original Dataset	Hijacking Dataset	Dataset Size	
I	CIFAR-10 (10)	MNIST (9)	60000	54000
II	CIFAR-10 (10)	GTSRB (9)	60000	15120
III	TinyImageNet-100 (100)	MNIST (10)	50000	60000
IV	TinyImageNet-100 (100)	GTSRB (10)	50000	16110

TABLE XII: The size of original and hijacking datasets used in *FL prediction/testing phase*.

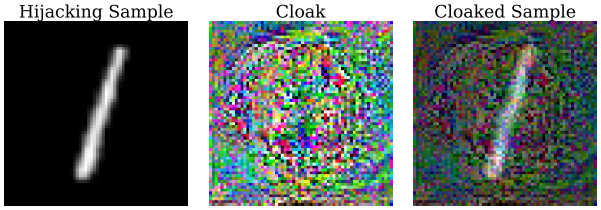
TaskSet	Original Dataset	Hijacking Dataset	Dataset Size	
I	CIFAR-10 (10)	MNIST (9)	10000	9000
II	CIFAR-10 (10)	GTSRB (9)	10000	4320
III	TinyImageNet-100 (100)	MNIST (10)	10000	10000
IV	TinyImageNet-100 (100)	GTSRB (10)	10000	4800



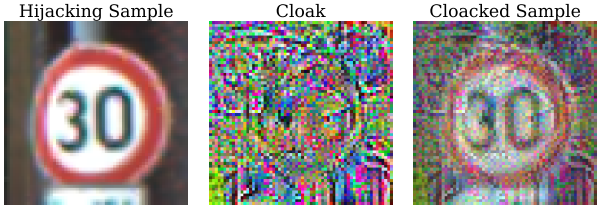
(a) TaskSet-I



(b) TaskSet-II



(c) TaskSet-III



(d) TaskSet-IV

Fig. 16: The generated cloaks and corresponding samples for TaskSets I–IV

REFERENCES

[1] https://decentralizedml.com/.
[2] http://yann.lecun.com/exdb/mnist/.
[3] https://www.cs.toronto.edu/~kriz/cifar.html.
[4] http://benchmark.ini.rub.de/?section=gtsrb.
[5] https://www.kaggle.com/c/tiny-imagenet.
[6] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2938–2948. PMLR, 2020.
[7] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning Attacks against Support Vector Machines. In *International Conference on Machine Learning (ICML)*. icml.cc / Omnipress, 2012.
[8] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *International Conference on Computational Statistics (COMPSTAT)*, pages 177–186. Physica-Verlag, 2010.
[9] Liang Gao, Huazhu Fu, Li Li, Yingwen Chen, Ming Xu, and Cheng-Zhong Xu. FedDC: Federated Learning with Non-IID Data via Local Drift Decoupling and Correction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10102–10111. IEEE, 2022.
[10] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Grag. Badnets: Iden-

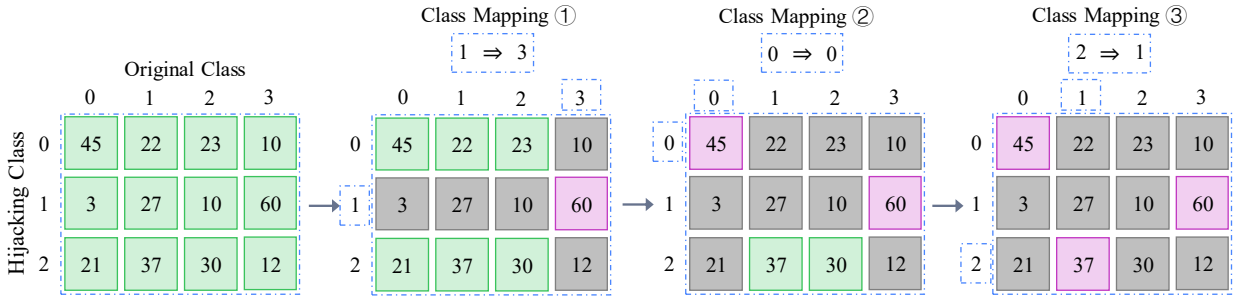


Fig. 17: Class Mapping Steps

tifying Vulnerabilities in the Machine Learning Model Supply Chain. *CoRR abs/1708.06733*, 2017.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

[12] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In *IEEE Symposium on Security and Privacy (S&P)*, pages 19–35. IEEE, 2018.

[13] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated Learning: Strategies for Improving Communication Efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning (PMPML)*. NIPS, 2016.

[14] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[15] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *European Conference on Computer Vision (ECCV)*, pages 122–138. Springer, 2018.

[16] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1273–1282. PMLR, 2017.

[17] Solmaz Niknam, Harpreet S. Dhillon, and Jeffrey H. Reed. Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges. *IEEE Communications Magazine*, 2020.

[18] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. SoK: Towards the Science of Security and Privacy in Machine Learning. In *IEEE European Symposium on Security and Privacy (Euro S&P)*, pages 399–414. IEEE, 2018.

[19] Nicolas Papernot, Patrick D. McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks Against Machine Learning. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 506–519. ACM, 2017.

[20] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The Limitations of Deep Learning in Adversarial Settings. In *IEEE European Symposium on Security and Privacy (Euro S&P)*, pages 372–387. IEEE, 2016.

[21] Ahmed Salem, Michael Backes, and Yang Zhang. Get a Model! Model Hijacking Attack Against Machine Learning Models. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2022.

[22] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520. IEEE, 2018.

[23] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and Communication-Efficient Federated Learning from Non-IID Data. *CoRR abs/1903.02891*, 2019.

[24] Wai Man Si, Michael Backes, Yang Zhang, and Ahmed Salem. Two-in-One: A Model Hijacking Attack Against Text Generation Models. *CoRR abs/2208.11180*, 2022.

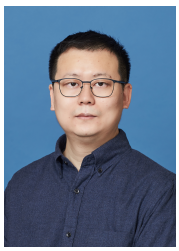
[25] Mingjie Sun, Jian Tang, Huichen Li, Bo Li, Chaowei Xiao, Yao Chen, and Dawn Song. Data Poisoning Attack against Unsupervised Node Embedding Methods. *CoRR abs/1810.12881*, 2018.

[26] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 2008.

[27] Weilin Xu, David Evans, and Yanjun Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[28] Rui Ye, Mingkai Xu, Jianyu Wang, Chenxin Xu, Siheng Chen, and Yanfeng Wang. FedDisco: Federated Learning with Discrepancy-Aware Collaboration. In *International Conference on Machine Learning (ICML)*, pages 39879–39902. PMLR, 2023.

Zheng Li is a Professor at the School of Cyber Science and Technology, Shandong University, Qingdao, China. He received the Ph.D. degree in Computer Science from CISPA Helmholtz Center for Information Security, Germany, in 2023, and the M.S. and B.S. degrees in Computer Science from Shandong University, China, in 2020 and 2017, respectively. His research focuses on trustworthy machine learning, with an emphasis on privacy attacks (e.g., membership inference), security threats (e.g., backdoor and data poisoning), and technical defenses against unethical AI deployments. He has published in top-tier venues including IEEE S&P, USENIX Security, CCS, and NeurIPS.



Hao Li received his Ph.D. from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2011. He is currently a Professor at the same institute. He has published dozens of papers in top-tier conferences and journals, including ACM CCS and USENIX Security. His research interests include artificial intelligence security and privacy, access control, and database security.