Breaking Privacy in Model-Heterogeneous Federated Learning

Atharva Haldankar
Virginia Tech
Blacksburg, VA, USA
ahaldankar@vt.edu

Arman Riasi
Virginia Tech
Blacksburg, VA, USA
armanriasi@vt.edu

Hoang-Dung Nguyen
Virginia Tech
Blacksburg, VA, USA
nhd@vt.edu

Tran Viet Xuan Phuong
University of Arkansas at Little Rock
Little Rock, AR, USA
ptran@ualr.edu

Thang Hoang
Virginia Tech
Blacksburg, VA, USA
thanghoang@vt.edu

ABSTRACT

Federated learning (FL) allows multiple distrustful clients to collaboratively train a machine learning model. In FL, data never leaves client devices; instead, clients only share locally computed gradients with a central server. As individual gradients may leak information about a given client's dataset, secure aggregation was proposed. With secure aggregation, the server only receives the aggregate gradient update from the set of all sampled clients without being able to access any individual gradient. One challenge in FL is the systems-level heterogeneity that is quite often present among client devices. Specifically, clients in the FL protocol may have varying levels of compute power, on-device memory, and communication bandwidth. These limitations are addressed by model-heterogeneous FL schemes, where clients are able to train on subsets of the global model. Despite the benefits of model-heterogeneous schemes in addressing systems-level challenges, the implications of these schemes on client privacy have not been thoroughly investigated.

In this paper, we investigate whether the nature of model distribution and the computational heterogeneity among client devices in model-heterogeneous FL schemes may result in the server being able to recover sensitive data from target clients. To this end, we propose two attacks in the model-heterogeneous FL setting, even with secure aggregation in place. We call these attacks the Convergence Rate Attack and the Rolling Model Attack. The Convergence Rate Attack targets schemes where clients train on the same subset of the global model, while the Rolling Model Attack targets schemes where model parameters are dynamically updated each round. We show that a malicious adversary can compromise the model and data confidentiality of a target group of clients. We evaluate our attacks on the MNIST and CIFAR-10 datasets and show that using our techniques, an adversary can reconstruct data samples with near perfect accuracy for batch sizes of up to 20 samples.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**.

RAID 2024, September 30–October 02, 2024, Padua, Italy

ACM ISBN 979-8-4007-0959-3/24/09
https://doi.org/10.1145/3678890.3678905

KEYWORDS

Heterogeneous Federated Learning, Secure Aggregation, Privacy

ACM Reference Format:
Atharva Haldankar, Arman Riasi, Hoang-Dung Nguyen, Tran Viet Xuan Phuong, and Thang Hoang. 2024. Breaking Privacy in Model-Heterogeneous Federated Learning. In *The 27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2024), September 30–October 02, 2024, Padua, Italy*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3678890.3678905

1 INTRODUCTION

In today's world, a massive amount of data is generated from edge devices, including phones, smartwatches, and various other Internet of Things (IoT) devices [16, 37]. Therefore, it has become critical for machine learning workloads to utilize the data that is distributed across these edge devices. With traditional distributed machine learning, clients upload their data samples to the central server, and the server then trains on that data to generate a global model. However, the traditional approach may not respect the privacy of data samples, which may make it infeasible in practice [37]. In particular, many countries have data privacy and security laws, which make it difficult for corporations to share sensitive information. For example, the General Data Protection Regulation (GDPR) was passed in the European Union (EU) to protect personal data of EU citizens [1].

Due to these privacy concerns, a new decentralized approach to machine learning, named federated learning (FL), was proposed [37]. In FL, clients do not upload their training data to a central server. Instead, they download a global model from the server, perform local updates to the downloaded model, and send those updates (e.g. gradients, model parameters) to the server. The server then aggregates these updates into the global model for the next round of training. Assuming that a particular client's update does not reveal information about the dataset used by that client, FL would provide a reasonable level of privacy. Unfortunately, this assumption does not hold in practice. A number of works have shown that client privacy can be broken when the server has access to individual client updates [9, 23, 62, 66, 69]. To provide stronger privacy guarantees in FL, Bonawitz et al. [10] proposed secure aggregation. With secure aggregation, the clients and server interactively run a protocol such that the server learns the average update across all clients without learning any individual client's update.

Despite the numerous applications that FL enables [5, 6], there are a number of practical challenges that are not addressed by the

original FL protocol. One such challenge is the heterogeneity among client devices. Data space heterogeneity occurs when clients collect data from differing feature or label spaces and statistical heterogeneity occurs when the data collected by clients is not independent and identically distributed (IID) [22, 49]. Furthermore, client devices may have different computational resources, communication bandwidths, and storage capacities. This is referred to as system heterogeneity, and is very common in the IoT setting [35, 40, 64]. With system heterogeneity, devices with less computing power generally train slower than devices with more computing power [2, 38, 55]. Furthermore, devices that are limited in storage may not be able to load the full global model into memory [22].

To address the challenges of having clients with different compute power and communication bandwidths, various techniques have been proposed. For example, client selection schemes ensure that selected clients have sufficient computational resources [30, 42], and asynchronous communication schemes [41, 53, 59] allow the server to aggregate client updates as they come in. Even when these techniques are used, however, clients must have enough compute power and storage capacity to make updates to the full global model. A newer line of work has shown that the local model architecture may differ from the global model architecture [4, 18, 26]. This allows clients to only train on a subset of the global model parameters, based on their computational capabilities. Clients with similar computational capabilities are grouped together into cohorts, and each cohort trains on the same model parameters for a given round. This approach, known as model-heterogeneous FL, has been gaining a lot of research attention [22, 47, 61].

However, the implications of model-heterogeneous FL schemes on client privacy have not been rigorously investigated. For example, it is not clear if the model distribution pattern of these schemes, combined with the heterogeneous compute power of client devices, can be targeted by a malicious server to recover sensitive client data. Furthermore, while secure aggregation has been extensively studied in the model-homogeneous FL setting, to the best of our knowledge, no prior works have investigated how client privacy is impacted in model-heterogeneous FL schemes with secure aggregation in place. Thus, we pose the following question in this paper:

Can the adversarial server exploit characteristics of the model-heterogeneous FL setting, hardened with secure aggregation, in order to break the model and data confidentiality for a group of clients?

1.1 Our Contributions

We answer this question in the affirmative with two new attacks. Our attacks demonstrate that unique aspects of model-heterogeneous FL schemes, like the differing computational power of clients or the pattern of submodel distribution, can leak information to a malicious server. We show that the server can exploit this leakage to reconstruct data samples from targeted clients without needing any auxiliary information about client datasets. Our contributions can be elaborated as follows:

- Our first attack, hereafter referred to as the *Convergence Rate Attack* targets schemes where cohorts are assigned submodels that remain static across rounds. We show that the server is able to exploit the fact that client devices converge at different rates in order to recover individual gradient updates from a target cohort. At a high level, the server chooses a target cohort, p, and waits for all cohorts with at least as much compute power as p to converge. Then, by issuing malicious model parameters to cohort p and leveraging the fact that cohorts with less compute power than p do not contribute to cohort p's submodel, the server is able to extract the approximate plaintext gradient update from cohort p. Section 5.1 discusses this attack.
- Our second attack, hereafter referred to as the *Rolling Model Attack*, targets schemes where clients train on different submodels each round (Section 5.2). These submodels are updated either deterministically or randomly. We first show how the server is able to extract model parameters for a target cohort associated with a single node. Then, we generalize this attack to allow the server to obtain the entire gradient update for the target cohort. We demonstrate that the server can accurately extract a target cohort's update after only two rounds of training. We also investigate how the server can add noise to malicious model parameters issued to clients, in order to prevent clients from detecting that the attack is in progress. We show that even when the server adds a relatively large amount of noise, the attack accuracy remains high. After obtaining a target cohort's gradient update in the clear, we show that the server can reconstruct data samples associated with clients in the target cohort.
- We empirically evaluate the effectiveness of our attacks on real world datasets (MNIST, CIFAR-10) by comparing our reconstructions to the original images (Section 6). For the Rolling Model Attack, we show that in the best case, a server can reconstruct original data samples with near-perfect accuracy for batch sizes of 20 or less. Similarly, for the Convergence Rate Attack, we show that the best reconstructions have Pearson correlation coefficients ≥ 0.87 with respect to the original training samples.
- Finally, we discuss potential mitigations to our proposed attacks, such as decentralization, adding hardware support, differential privacy, and encryption, in Section 7.3. These mitigations limit the amount of trust that clients must place on the server.
- An open source implementation of our code can be found at https://github.com/vt-asaplab/model-hetero-fl-attacks.

2 RELATED WORK

2.1 Model-Heterogeneous FL

A number of works tackle the problem of system heterogeneity. When the local model architecture of clients is the same as the global model architecture, asynchronous updating and client selection schemes may be beneficial. Recent works have shown that it is feasible for clients to train on model architectures that differ from the global model. This technique is known as model-heterogeneous FL. Model-heterogeneous FL can be split into two main approaches. The first approach leverages ensemble distillation in order to train an accurate global model, while the second approach is based on partial training of the global model.

2.1.1 Ensemble Distillation. It was initially proposed as a way to distill knowledge from an ensemble of teacher models to a student model. In FL, various schemes have been proposed to transfer knowledge from local client models to the global model. For example, in schemes like FedDF [36], FedAUX [50], and FedBE [14], the logit outputs from teacher models, held by clients, are used in conjunction with an unlabeled public dataset to train a student model on the server. The authors of FedGKT [24] and FedET [17] extend traditional FL-based ensemble distillation schemes to permit the server-side model to be larger than any single client model.

2.1.2 Partial Training. It is another approach to model-heterogeneous FL. Partial training schemes assign different subsets of the global model to clients, based on their computational resources, and clients train on these submodels. During the aggregation phase, each global parameter is averaged from the clients that contain that parameter in their assigned submodel. In HeteroFL [18], the width of hidden channels are scaled down according to the computational abilities of clients. Similarly, FjORD [26] proposes a mechanism for dropping out neurons in hidden layers for clients with less computational power. Their approach, which they refer to as *Ordered Dropout*, prunes neurons in a structured manner.

Both HeteroFL and FjORD distribute submodels in a static manner, meaning that the model parameters assigned to a particular cohort stay fixed across rounds. However, the parameters associated with client submodels may also vary based on the round. For example, the Federated Dropout scheme [11] zeros out a fixed percentage of randomly selected activations for each fully-connected layer. Then, during the distribution phase, clients receive only the non-zero parameters. Similarly, FedRolex [4] proposes a rotating submodel extraction scheme, where the model parameters assigned to a client are deterministically updated across rounds. The authors of Helios [60] propose a scheme where a fixed percentage of neurons with the highest changing values are kept and a fraction of the other neurons are dropped out. Since the neurons with the highest changing values may differ by round, the model parameters issued to clients are updated each round.

2.2 Gradient Inversion Attacks

2.2.1 Standard FL Setting. A number of works have shown that given a client's gradient update to the global model, information can be leaked about that client's private training dataset. Phong et al. [48] were the first to show that the model parameters associated with a neuron in a densely connected layer can reveal information about the activations of the previous layer. Building on this work, Zhu et al. [69] proposed an iterative optimization method to reconstruct client data from publicly shared gradients. Zhao et al. [66] extended this work by proposing a simple and reliable technique for extracting ground-truth labels from gradients. Geiping et al. [23] and Yin et al. [62] proposed gradient inversion attacks in more realistic settings, and showed that even averaging gradients for varying batch sizes does not guarantee that client privacy is protected. Boenisch et al. [9] proposed an adversarial initialization scheme for the model parameters of the first fully-connected layer. Through this technique, they were able to reconstruct individual data points with high accuracy.

2.2.2 Privacy-Preserving FL Setting. All of the attacks in the standard FL setting assume that the server has direct access to gradients from individual clients. However, recent works have demonstrated that a malicious adversary can compromise the confidentiality of clients even when privacy-preserving techniques, like secure aggregation or differential privacy, are in place. Lam et al. [31] proposed an analytical, matrix factorization approach designed to extract private client gradients, despite secure aggregation being in place. Their attack relied on side channel information, in the form of summary analytics about the participation rate of particular users in the FL process. Pasquini et al. [45] proposed a model inconsistency approach to breaking the guarantees provided by secure aggregation. In their work, the server carefully chooses which model parameters to distribute to clients so that gradients are suppressed from all but one target client. Finally, Boenisch et al. [8] proposed an attack where a malicious aggregator can recover individual client gradients despite both secure aggregation and distributed differential privacy being in place. Their attack relied on an adversary with access to a large pool of sybil devices and with the capability to select which clients participate in a given round of training.

3 PRELIMINARIES

Notation. Unless stated otherwise, capital letters denote sets and machine learning models. The cardinality of a set X is expressed as $|X|$, and the intersection ($\cap$), union ($\cup$), and proper subset ($\subset$) operators are used. Machine learning models are expressed as W, with additional subscripts/superscripts providing more information on the specific model type. Given two machine learning models, W_1 and W_2, where $W_2 \subset W_1$, $W_1 \setminus W_2$ refers to the submodel of W_1, where all model parameters of W_2 are excluded. ∇ is used to define a gradient update for a model W with respect to a loss function f. Bolded lowercase letters (e.g. $\mathbf{x}$) denote vectors. $\lfloor v \rfloor$ denotes the floor function applied to a scalar v. We use $\cdot$ to denote the multiplication operation, either between two scalar operands or between a matrix and a vector.

3.1 Federated Learning

FL allows a set of n clients to jointly train a global machine learning model on decentralized data [37]. The FL protocol guarantees that a given client's private dataset never leaves that client. Instead, each client sends their gradient update to a central server. The standard FL protocol is organized into rounds. Before the first round, the server randomly initializes weights for the global model, W_0. Then, for each round, t, the following steps take place. First, the server samples a subset of $k \leq n$ clients, and distributes the global model, W_t, to each of those k clients. Next, each client minimizes an objective function with respect to the global model W_t, based on its local dataset. Finally, each client sends its gradient update to the central server. Afterwards, the server updates the global model based on the average gradient across all selected clients. This process can be formalized as follows. For notational convenience, the round index, t, is dropped from Equation (1). G_i denotes client i's gradient update, X_i denotes the set of training samples held by client i, and Y_i denotes the labels corresponding to X_i. f represents the objective function that client i is trying to minimize. $\overline{G}$ denotes the average gradient update across all k clients.

$$G_i = \nabla_W f(X_i, Y_i), \quad \overline{G} = \frac{1}{k}\sum_{i=1}^{k} G_i \tag{1}$$

Finally, the server computes W_{t+1}, the global model for the next round, according to a learning rate parameter η. W_{t+1} is computed in the following manner:

$$W_{t+1} = W_t - \eta\overline{G} \tag{2}$$

Alternatively, clients can take a gradient descent step locally and upload the resulting model to the server. The server can then compute the new global model, W_{t+1}, by averaging each of the client models. When clients train on batches consisting of their entire dataset for 1 local epoch, which we assume to be the case, this aggregation technique is equivalent to that of Equation (2). Concretely, let W_t^i denote client i's local model after training. The server can compute W_{t+1} by averaging the client models as follows:

$$W_{t+1} = \frac{\sum_{i=1}^{k} W_t^i}{k} \tag{3}$$

Secure Aggregation. Bonawitz et al. [10] proposed secure aggregation as a way to increase the privacy guarantees of the FL protocol. With secure aggregation, clients do not send their individual updates to the server. Instead, the clients and server collectively run a Secure Multiparty Computation (Secure MPC) protocol. The server is different from clients, in that it holds no input, but can communicate with clients through secure authenticated channels. This protocol ensures that the server learns the aggregate update across a set of k clients without learning any individual client's update. The scheme proposed by Bonawitz et al. can be used by the clients and server to jointly compute W_{t+1} in Equations (2) or (3).

3.2 Single-Input Gradient Leakage

Prior works have shown that the input to a densely connected neural network layer can be reconstructed through the gradients of the model parameters for that layer. While the leakage is applicable to arbitrary activation functions, we will focus on the ReLU activation function, since it is commonly used in practice [7]. In particular, consider the case when the first layer of a neural network is densely connected. In that case, the original input can be reconstructed in the following manner.

Let n_x denote the size of the input layer and n_y denote the size of the first dense layer. Let $\mathbf{x} \in \mathbb{R}^{n_x}$ denote the flattened input, and $\mathbf{y} \in \mathbb{R}^{n_y}$ denote the ReLU activation of $\mathbf{x}$, computed as

$$\mathbf{y} = \max(W \cdot \mathbf{x} + b, 0)$$

for $W \in \mathbb{R}^{n_y \times n_x}$ and $b \in \mathbb{R}^{n_y}$. Let the loss of the network be represented by $\mathcal{L}$. Now, consider a particular node, k, in the first dense layer, and let $y_k \in \mathbf{y}$ represent its activated output. Assume that $y_k > 0$. Let $W_k \in \mathbb{R}^{1 \times n_x}$ represent the weight vector between the input layer and node k, and let b_k represent the bias term associated with k. Then, the gradients of $\mathcal{L}$ with respect to b_k and W_k can be computed as follows:

$$\frac{\partial \mathcal{L}}{\partial b_k} = \frac{\partial \mathcal{L}}{\partial y_k} \cdot \frac{\partial y_k}{\partial b_k} = \frac{\partial \mathcal{L}}{\partial y_k}$$

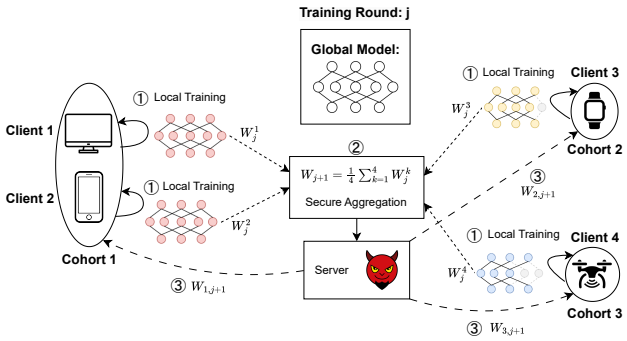

Figure 1: Model-Heterogeneous Federated Learning System

since $\frac{\partial y_k}{\partial b_k} = 1$ for $y_k = W_k \cdot \mathbf{x} + b_k$.

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial y_k} \cdot \frac{\partial y_k}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial b_k} \cdot \mathbf{x} \tag{4}$$

since $\frac{\partial \mathcal{L}}{\partial y_k} = \frac{\partial \mathcal{L}}{\partial b_k}$ and $\frac{\partial y_k}{\partial W_k} = \mathbf{x}$.

From equation (4), it follows that

$$\mathbf{x} = \left(\frac{\partial \mathcal{L}}{\partial b_k}\right)^{-1} \cdot \frac{\partial \mathcal{L}}{\partial W_k}$$

Hence, the input $\mathbf{x}$ can be reconstructed using gradient information from the first densely connected layer.

4 MODEL

System Model. Our attacks consider a system model with two main entities: n client devices and a server. The clients and server jointly participate in a FL protocol with secure aggregation in place. Clients have differing computational power from each other, and clients with similar computational power are grouped together into cohorts. There are $m \leq n$ different cohorts, where cohort 1 consists of clients with the most computational power, and each cohort $i+1$ has less computational power than cohort i ($1 \leq i < m$). Figure 1 illustrates the interaction between the parties in our system. Training proceeds in synchronous rounds, similar to the standard FL scheme [37]. A given round is considered complete after the server receives the aggregated output for each of the submodels. During each round, the server first distributes subsets of the global model to clients, based on the cohort they belong to.

Then, as Figure 1 shows: ① Clients update the model locally with respect to their training data. ② The clients and server run a secure aggregation protocol to compute the average model update across clients, which the server sets as the global model for the next round. ③ The server distributes subsets of the new global model to all cohorts.

Threat Model and Assumptions. We consider the server to be the adversary and clients to be trusted. The server's goal is to compromise model and data confidentiality of individual clients or cohorts. In order to achieve this goal, it will suffice for the server to learn a specific cohort's aggregated gradient update or to extract an individual client's gradient update. This is because after obtaining

an individual gradient update, the server can perform a gradient inversion attack to reconstruct data samples [9, 23, 62]. The server is malicious, meaning it can deviate from the FL protocol. Specifically, the server may assign arbitrary model parameters (weights and biases) to any number of cohorts during the distribution phase (step ③ in Figure 1). Note, however, that the server does not have control over the model architectures. Therefore, the server is not able to change the size of the submodels (i.e. the number of nodes in hidden layers) issued to clients in a given cohort. Our threat model is investigated in many prior works, including [9, 21, 23, 69]. Furthermore, a malicious adversary is a more realistic threat model than a honest-but-curious adversary in most real-world settings. For example, OpenMined and PyTorch recently released four new libraries for FL [12]. These libraries allow malicious participants to alter the parameters and even make minimal changes to the architecture of the shared model.

5 THE ATTACKS

5.1 Convergence Rate Attack

Attack Scenario. Our first attack applies to static partial-training based approaches, where clients train on the same segment of the global neural network every round [20, 22, 61]. In this setting, clients have different computational capabilities, and clients with more computational resources generally train on a larger subset of nodes than clients with less computational resources. A number of works use the static partial training approach [13, 18, 26, 33, 34]. In [13, 18] clients are organized into cohorts (or groups), based on their computational resources. In fact, each of these works can be viewed as organizing clients into cohorts, since a cohort may, in the extreme case, consist of a single client. We will focus on the setting where submodels issued to the different cohorts are proper subsets of one another. Without loss of generality, however, this attack can be applied to other static partial-training schemes.

Suppose there are n clients, split into $m \leq n$ different cohorts. The model-heterogeneous FL process can then be described as follows. (1) The server initially holds global model W_j. In the distribution phase for round j, the server issues submodel $W_{i,j}$ to each cohort i, $1 \leq i \leq m$, where $W_{i,j}$ is a subset of W_j and each $W_{i+1,j} \subset W_{i,j}$. (2) Each client k computes its local gradient G_k, and updates its local model accordingly. (3) The clients and server participate in a secure aggregation protocol. This protocol takes each trained client model, W_j^k as input, and outputs W_{j+1}, the global model for the next round. Finally, steps (1) - (3) repeat until the final round of training.

Attack Details. To formulate our attack, we introduce the following notation. Let $n_1, n_2, \ldots, n_m$ denote the number of clients in cohorts $1, 2, \ldots, m$, such that $n_1 + n_2 + \cdots + n_m = n$. Let K represent the set of all clients and M be a client-to-cohort mapping from $\{1, \ldots, n\} \rightarrow \{1, \ldots, m\}$. Further, let $W_{i,j}^k$ represent the submodel held by client k after local training that matches the architecture of $W_{i,j}$, and $G_{i,j}^k$ denote client k's gradient update to $W_{i,j}$. If client k belongs to cohort $l > i$, $W_{i,j}^k = G_{i,j}^k = 0$, and $G_{i,j}^k$ will be ignored in the aggregation process. This ensures that clients do not update larger models than their computation or storage capabilities allow.

The average gradients for each submodel will be represented by $\overline{G}_{i,j}$. Additionally, $\widehat{W}_{i,j} := W_{i,j} \setminus W_{i+1,j}$, and $\widehat{G}_{i,j}^k := G_{i,j}^k \setminus G_{i+1,j}^k$. In the equations that follow, subscripts and superscripts may be dropped when they are clear from the context. The global model for the next iteration, W_{j+1}, can be computed as follows:

$$W_{j+1} = W_{m,j+1} \cup (\widehat{W}_{m-1,j+1}) \cup \cdots \cup (\widehat{W}_{1,j+1})$$
$$W_{i,j+1} = W_{i,j} - \eta \overline{G}_{i,j}$$

At a high level, the Convergence Rate Attack exploits the heterogeneity of client devices across cohorts. Some cohorts have clients with much more computational power than other cohorts. Therefore, cohorts with higher-end devices will reach convergence quicker than cohorts with lower-end devices [2, 38, 55]. Specifically, we assume that cohorts $1, 2, \ldots, m$ converge prior to the distribution phase for rounds $r_1, r_2, \ldots, r_m$, where $0 < r_1 \leq r_2 \leq \cdots \leq r_m \leq t$, and t is the total number of training rounds in the FL protocol. Using this assumption, we show that we are able to extract the aggregated output from individual cohorts.

Let p denote the cohort which is targeted for convergence. At the start of the training process, the server will hold model W_0. Additionally, the server will hold a malicious model W_{mal}, with the same architecture as cohort p's submodel, W_p. The model parameters of W_{mal} will be chosen such that the server is able to easily reconstruct data samples from clients in cohort p, given this cohort's gradient update to W_{mal}. This can be done, for example, using the trap weights technique of Boenisch et al. [9]. Let G_{mal} denote any gradient updates to W_{mal}. Also, let $\widehat{W}_{mal}$ and $\widehat{G}_{mal}$ denote the submodels of W_{mal} and G_{mal}, respectively, where the model parameters updated by cohort $p+1$ are excluded. The server will initially distribute $W_{i,0}$ to each cohort i. Then, for each subsequent round, the server may assign arbitrary model parameters. For rounds $1 \leq j \leq r_p - 1$, the server will distribute subsets of the true global model to all clients. Then, for round r_p, the server will distribute W_{mal} to cohort p instead of W_{p,r_p}. All other cohorts, $i \neq p$, will receive W_{i,r_p}.

After local training, clients will hold the following submodels:

$$W_{i,r_p}^k = \begin{cases} i = p, & W_{mal} - \eta G_{mal}^k \\ i \neq p, & W_{i,r_p} - \eta G_{i,r_p}^k \end{cases}$$

Next, the server will obtain $\widehat{W}_{p,r_p+1}$ from the secure aggregation protocol. Since clients in cohorts $\geq p+1$ do not contribute to $\widehat{W}_{p,r_p+1}$, their updates will be ignored. Hence, $\widehat{W}_{p,r_p+1}$ can be expressed as:

$$\widehat{W}_{p,r_p+1} = \frac{\sum_{k=1}^{n} \widehat{W}_{p,r_p}^k}{n_1 + n_2 + \cdots + n_p} \tag{5}$$

The numerator in Equation (5) can be split into two parts, based on whether $M(k) < p$ or $M(k) = p$. Let $K_1 := \{k \in K \mid M(k) < p\}$ and $K_2 := \{k \in K \mid M(k) = p\}$. Equation (5) can be expressed as:

$$\frac{\sum_{k \in K_1} (\widehat{W}_{p,r_p} - \eta \widehat{G}_{p,r_p}^k) + \sum_{k \in K_2} (\widehat{W}_{mal} - \eta \widehat{G}_{mal}^k)}{n_1 + n_2 + \cdots + n_p}$$

However, since all cohorts $i < p$ have already converged, the gradient updates of the clients in these cohorts will be ≈ 0. Therefore,

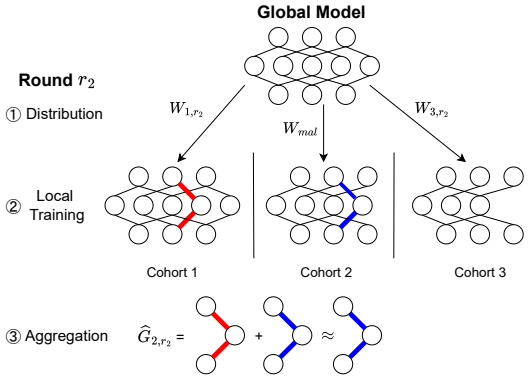

Figure 2: Convergence Rate Attack

the server will observe

$$\begin{aligned}\widehat{W}_{p,r_p+1} &= \frac{\sum_{k\in K_1}(\widehat{W}_{p,r_p}) + \sum_{k\in K_2}(\widehat{W}_{mal} - \eta\widehat{G}^k_{mal})}{n_1 + n_2 + \cdots + n_p} \\ &= \frac{\sum_{k\in K_1}(\widehat{W}_{p,r_p}) + n_p\widehat{W}_{mal} - \eta(\sum_{k\in K_2}\widehat{G}^k_{mal})}{n_1 + n_2 + \cdots + n_p}\end{aligned} \tag{6}$$

Notice, however, that the server already knows $\widehat{W}_{p,r_p}$ for each client in cohort $i < p$, since $\widehat{W}_{p,r_p} \subset W_{i,r_p}$. The server also knows the number of clients in each cohort, n_i, for $1 \le i \le m$, as well as $\widehat{W}_{mal}$ and η. Finally, the server will obtain $\widehat{W}_{p,r_p+1}$ from the secure aggregation protocol. Therefore, the server can solve Equation (6) for $\sum_{k\in K_2}\widehat{G}^k_{mal}$. This represents cohort p's aggregated gradient update to $\widehat{W}_{mal}$, and can be used as the input to gradient inversion attacks, like [9, 23, 62]. Therefore, the server will be able to compromise the confidentiality of cohort p.

Concrete Example. Consider the scenario shown in Figure 2, where there are 3 cohorts in total and cohort 2 is targeted. The server will initially hold global model W_0, and before the first round, the server will distribute $W_{i,0}$ to all cohorts i, $1 \le i \le 3$. Then, for all rounds $1 \le j \le r_2 - 1$, the server will distribute the true global model to all clients. For round r_2, the server will distribute inconsistent models to cohorts. Cohort 2 will be issued a malicious model, W_{mal}, while all other cohorts will be issued W_{i,r_2}. However, by round r_2, cohort 1 will have converged. Therefore, during the local training phase, clients in cohort 1 will compute gradients of approximately 0. Hence, when the server obtains $\widehat{W}_{2,r_2+1}$, it will be able to directly observe cohort 2's contributions to $\widehat{W}_{mal}$. Using cleverly selected model parameters, the server can invert cohort 2's gradient update to reconstruct private data samples. Finally, this attack can be repeated until the server learns the aggregated gradient updates of all other cohorts.

Comparison to Model-Homogeneous FL. The Convergence Rate Attack exploits the fact that different submodels are issued to different cohorts. Hence, it may not work as well in the model-homogeneous FL setting. For a similar attack to work in this setting, the server has to wait for all clients to converge, and then issue malicious model parameters to a target client. In the model-heterogeneous FL setting, however, only the target cohort and all cohorts with greater computational resources need to converge. This is because cohorts with less computational resources than the target cohort will not send updates to the target cohort's submodel.

5.2 Rolling Model Attack

Attack Scenario. The Rolling Model Attack is present in the setting where submodels issued to different cohorts overlap. Unlike the static partial-training approaches described in Section 5.1, these submodels do not stay fixed. Instead, they are modified, either deterministically or randomly, each round. For example, the Federated Dropout scheme randomly drops out a fixed percentage of neurons each training round [11]. Other schemes deterministically rotate which nodes are selected for a particular cohort's submodel. In Helios [60], a fixed percentage of the neurons with the highest changing values are kept, while a fraction of the other neurons are dropped. In FedRolex [4], nodes are selected deterministically based on the round index. For the remainder of this section, we will focus on the deterministic rotating scheme proposed by FedRolex. Without loss of generality, however, this attack can be applied to other deterministic and random submodel rotating schemes. Throughout the discussion of this attack, we also assume that all cohorts train on data from the same distribution.

Attack Details. We use the following notation to formulate this attack. Suppose there are n clients split into $m \le n$ cohorts. Consider a neural network, W_j, with a number of hidden layers. Take one such hidden layer, indexed by h, and let θ_h represent the model parameters (weights and biases) between layers $h-1$ and h.

Each cohort updates a set of nodes that varies across rounds. Let n_h denote the total number of nodes in layer h and let β_i represent the proportion of nodes in layer h that cohort i keeps. Additionally, let $S^{i,j}_h$ represent the node indices in layer h that cohort i updates during round j. The first and last node indeces are 0 and $n_h - 1$, respectively. Finally, let $\overline{j} = j \bmod n_h$. $S^{i,j}_h$ can be defined as follows:

$$S^{i,j}_h = \begin{cases} \{\overline{j}, \overline{j}+1, \ldots, \overline{j} + \lfloor \beta_i n_h \rfloor - 1\} & \text{if } \overline{j} + \lfloor \beta_i n_h \rfloor \le n_h \\ \{\overline{j}, \overline{j}+1, \ldots, n_h - 1\} \cup \{0, \ldots, \overline{j} + \lfloor \beta_i n_h \rfloor - 1 - n_h\}, & \\ \text{otherwise} & \end{cases}$$

Let $\theta^{1,j}_h, \theta^{2,j}_h, \ldots, \theta^{m,j}_h$ denote different subsets of θ_h for round j, where parameters $\theta^{i,j}_h$ are issued to cohort i during round j. Each $\theta^{i,j}_h$ consists of the parameters between nodes in layer $h-1$ and nodes indexed by $S^{i,j}_h$. Note that $|\theta^{1,j}_h| > |\theta^{2,j}_h| > \cdots > |\theta^{m,j}_h|$.

The Rolling Model Attack exploits the leakage caused by cohorts changing which submodel they update across rounds. If cohorts statically updated the same set of nodes, this vulnerability would not be present. However, since cohorts rotate which nodes they update, the number of cohorts that contribute to a given node varies across rounds. In particular, one cohort may contribute to a given node one round, but then not update that node during a subsequent round. A malicious adversary can distribute the same model parameter values to all cohorts for both rounds, thereby obtaining the aggregated update of the cohort that dropped out. Specifically, the attack can be formulated as follows. Let $n_1, n_2, \ldots, n_m$ denote the number of clients in cohorts $1, 2, \ldots, m$, such that $n_1 + n_2 + \cdots + n_m = n$. The server holds global model W_0, and

initially distributes subsets of W_0 to cohorts. Consider a particular node, X, in hidden layer h, and let θ_X denote the set of all model parameters between layer $h-1$ and X. Let $G_X^{i,j}$ denote cohort i's aggregate gradient update to θ_X for round j. Due to the way that submodels are selected, there must be a round, j, such that $X \in S_h^{i,j}$ for all $i \in \{1, 2, \ldots, m-1\}$ and $X \in S_h^{i,j+1}$ for all $i \in \{1, 2, \ldots, m\}$. In particular, for $j = 0$, X is indexed by $\lfloor \beta_m n_h \rfloor$. Suppose the server chooses to distribute subsets of W_0 for both rounds j and $j+1$. Let W_X denote the subset of W_0 consisting of all model parameters between layer $h-1$ and node X. Then, the following system of equations can be constructed:

$$G_X^j = \frac{1}{\sum_{p=1}^{m-1} n_p} \sum_{i=1}^{m-1} G_X^{i,j}, \quad \theta_X^j = W_X - \eta G_X^j$$

$$G_X^{j+1} = \frac{1}{n} \sum_{i=1}^{m} G_X^{i,j+1}, \quad \theta_X^{j+1} = W_X - \eta G_X^{j+1}$$

Note that $G_X^{i,j} = G_X^{i,j+1}$ for all i, where $1 \le i \le m-1$, since clients in those cohorts train on the same model with the same local dataset for both rounds. The server can then compute $\theta_X^{j+1} - \theta_X^j$. This difference can be expressed as:

$$\begin{aligned}
&\theta_X^{j+1} - \theta_X^j \\
&\quad = (W_X - \eta G_X^{j+1}) - (W_X - \eta G_X^j) \\
&\quad = \eta (G_X^j - G_X^{j+1}) \\
&\quad = \eta \left(\frac{1}{n - n_m} \sum_{i=1}^{m-1} G_X^{i,j} - \frac{1}{n} \sum_{i=1}^{m} G_X^{i,j+1} \right) \\
&\quad = \eta \left(\frac{n}{n(n-n_m)} \sum_{i=1}^{m-1} G_X^{i,j} - \frac{n - n_m}{n(n-n_m)} \sum_{i=1}^{m} G_X^{i,j+1} \right) \\
&\quad = \frac{\eta}{n(n-n_m)} \left[n \sum_{i=1}^{m-1} G_X^{i,j} - (n - n_m) \left(\sum_{i=1}^{m-1} G_X^{i,j+1} + G_X^{m,j+1} \right) \right] \\
&\quad = \frac{\eta}{n(n-n_m)} \left[n_m \left(\sum_{i=1}^{m-1} G_X^{i,j+1} \right) + n_m (G_X^{m,j+1}) - n (G_X^{m,j+1}) \right] \\
&\quad = \frac{\eta}{n(n-n_m)} \left[n_m \left(\sum_{i=1}^{m} G_X^{i,j+1} \right) - n (G_X^{m,j+1}) \right] \\
&\quad = \frac{\eta}{(n-n_m)} (n_m (G_X^{j+1}) - G_X^{m,j+1})
\end{aligned}$$

Observe that the server knows the values of θ_X^{j+1}, θ_X^j, η, n, n_m, and G_X^{j+1}. Therefore, the server can solve for $G_X^{m,j+1}$ to extract cohort m's gradient update to node X for round $j+1$.

Concrete Example. Consider the scenario shown in Figure 3. For round j, only cohorts A and B update the parameter represented by θ_3. Then, during round $j+1$, cohorts A, B, and C all update θ_3. Cohorts A and B have the same gradient update to θ_3 for rounds j and $j+1$, because the server distributes the same model parameters for both of these rounds. Therefore, the server can observe cohort C's update to θ_3 for round j+1 in the clear, by computing $\theta_{3,j+1} - \theta_{3,j}$.

Expanding the Attack Scope. The attack details in this section explain how the server can extract the gradient update of cohort m to a target node X in hidden layer h. However, notice that clients only know the architecture of the model they are issued – not the values of the model parameters. Therefore, the server can deviate from the distribution protocol, and assign arbitrary model parameters to the target cohort. This capability allows the server to extract the entire gradient update for cohort m.

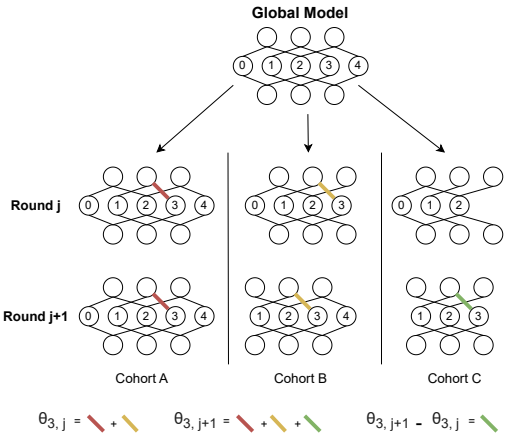

Figure 3: Rolling Model Attack

For example, consider the scenario when $\beta_{m-1} \ge 2\beta_m$ and $\lfloor \beta_{m-1} n_h \rfloor \le n_h$. For round 0, the server will honestly distribute model parameters. Then, for round 1, the server may distribute the submodel associated with node indeces $\{\lfloor \beta_m n_h \rfloor, \ldots, 2\lfloor \beta_m n_h \rfloor - 1\}$ to cohort m and honest model parameters to all other cohorts. Since $2\lfloor \beta_m n_h \rfloor \le \lfloor \beta_{m-1} n_h \rfloor$, all cohorts $i \le m-1$ will update nodes $\{\lfloor \beta_m n_h \rfloor, \ldots, 2\lfloor \beta_m n_h \rfloor - 1\}$ for both rounds 0 and 1. Therefore, the only change to the aggregate values of the model parameters associated with these nodes will be cohort m's contribution. Thus, the server will be able to extract cohort m's entire plaintext gradient update for round 1.

Reducing the Attack Footprint. In the Rolling Model Attack, the server must distribute the same model parameters to non-target cohorts across rounds in order to recover the target cohort's gradient update. However, with this approach, clients can easily detect that the server is acting maliciously, since they can observe that model parameters do not change across rounds. Therefore, we investigate the effect of adding noise to the malicious model parameters.

We add noise through the following procedure. For the first round, the server initializes the model parameters in all layers to a fixed value v. Each subsequent round, the server observes the average value of the global model parameters for each layer in the network. Let avg_L denote this average value for layer L. The server also maintains a constant value, p, which affects the range of noise to be sampled. The model parameters for layer L are then initialized according to a uniform distribution, $\mathcal{U}$, of the following form:

$$\mathcal{U}(v - avg_L \cdot p, v + avg_L \cdot p) \tag{7}$$

Note that in the Rolling Model Attack, the server is able to compromise the confidentiality of private training samples as early as the second round of training. Therefore, the server may only need to use the noisy initialization technique once.

6 EVALUATION

In this section, we evaluate the effectiveness of the Convergence Rate Attack and the Rolling Model Attack.

Implementation. Our implementation was written in Python 3.9.18, and our code was interpreted using the Python runtime. We added roughly 500 lines of code to the implementation of HeteroFL [18]. We made use of the PyTorch library [46] for model initialization, distribution, and local training. We also used Matplotlib [27] to help us visualize the original and reconstructed images. For model distribution, we used an index array to track the model parameter indices for each layer that should be assigned to a particular cohort. Then, we assigned the corresponding entries of the layer based on the index array. In the local training phase, clients took a single gradient descent step with respect to the samples in their datasets. We used the Adam optimizer. After local training, we simulated a secure aggregation protocol. The inputs to the aggregation protocol consisted of individual client models. These models were reshaped to fit the global model architecture, and any parameters not trained on were initialized with zeros. The aggregation protocol outputted the average of the input client models. The server treated the aggregation process as a black box. Our implementation is available at https://github.com/vt-asaplab/model-hetero-fl-attacks.

6.1 Experimental Setup

Hardware Setup. We tested our attacks on a x86_64 virtual machine running CentOS Stream 9, with 8 cores, 32 GB of RAM, and 256 GB of disk space. To accelerate the training of the machine learning models, we used a P40 Tesla server grade GPU. We utilized NVIDIA's CUDA Toolkit for GPU support. We spawned separate threads for each client and the server.

System Setup. We placed clients into one of three cohorts: A, B, or C. Cohort A received all model parameters during the distribution phase. The other cohorts were issued models with scaled-down hidden layers. Cohort B only received half of the model parameters for each hidden layer in the network, and cohort C only received a quarter of the parameters for each hidden layer. For the Convergence Rate Attack, we ran our experiments with 5 active users, with 1, 1, and 3 users in cohorts A, B, and C respectively. Our choice for the number of active users is similar to HeteroFL [18], where 10 active users were sampled per round. We placed more users in cohort C to simulate a large number of resource-limited devices training alongside fewer resource-rich devices. We targeted cohort B for this attack. By targeting this cohort, we only needed to wait for cohorts A and B to converge before running our attack. Therefore, we were able to show that this attack is more effective in the model-heterogeneous FL setting than in the model-homogeneous FL setting. For the Rolling Model Attack, we ran our experiments with 3 active users, with 1 client each in cohorts A, B, and C. The plaintext gradient update observable by the adversary would still be just as accurate with a larger number of users, as is explained by the methodology of Section 5.2. We targeted the client in cohort C.

Data Distribution. We ran our experiments on the MNIST [32] and CIFAR-10 [29] datasets. For the Convergence Rate Attack, we ran our experiments in the non-IID setting, to simulate a practical heterogeneous training environment [22, 64]. We trained in the IID setting for the Rolling Model Attack, since this attack did not rely on any convergence guarantees from the training process. In the IID setting, each client held training data samples corresponding to each of the labels of the dataset. However, in the non-IID setting, each client was limited to holding data corresponding to two distinct labels. The number of data samples per class was balanced. This particular non-IID setup was also investigated in [4, 18].

Table 1: Global model architecture used in the experiments on the MNIST dataset. The model size was scaled down based on the computational resources of the cohort.

FCNN Architecture - MNIST
Linear(in_features=784, out_features=5000, bias=True)
ReLU()
Linear(in_features=5000, out_features=10, bias=True)

Table 2: Global model architecture used in the experiments on the CIFAR-10 dataset. The model size was scaled down based on the computational resources of the cohort.

FCNN Architecture - CIFAR-10
Linear(in_features=3072, out_features=15000, bias=True)
ReLU()
Linear(in_features=15000, out_features=10, bias=True)

Neural Network Model. We evaluated our attacks on a fully connected neural network (FCNN) [9, 19]. The architecture for this model is given in Tables 1 and 2 for the MNIST and CIFAR-10 datasets, respectively. The model consisted of an input layer, a hidden layer scaled based on the client's computational capacity, and an output layer. The ReLU activation function was applied to the output of the hidden layer. As we chose to train on the MNIST and CIFAR-10 datasets, the output layer consisted of 10 neurons, representing each of the ten classes that the images could potentially belong to. We used the cross-entropy loss function.

Attack Parameters. For both attacks on the MNIST dataset, the size of the hidden layer for cohorts A, B, and C was 5000, 2500, and 1250, respectively. For the Convergence Rate Attack, we targeted the part of cohort B's submodel that did not overlap with cohort C's submodel (see Section 5.1 for details). For the Rolling Model Attack, we targeted the client in cohort C. Thus, for both attacks, the server was able to observe the plaintext gradient update to 1250 nodes. This attack surface is comparable to the trap weights architecture of Boenisch et al. [9]. On the CIFAR-10 dataset, the hidden layer consisted of 15,000 nodes. The larger number of hidden nodes was chosen to adjust for the larger input size of 3072. For both attacks, clients trained on small local dataset sizes of up to 20 samples for 1 local epoch with a batch size equal to the local dataset size. The Convergence Rate Attack was conducted over 10 global epochs. By running this attack for 10 global epochs, we observed that clients in both cohorts A and B converged. The Rolling Model Attack was run over 2 global epochs, since this was the minimum number of epochs necessary for the server to successfully observe the plaintext updates of the client in cohort C.

In order to simulate different cohorts having different computational power, clients in cohorts A, B, and C were assigned different

learning rates. For both attacks, clients in cohorts A, B, and C had fixed learning rates of 0.1, 0.05, and 0.01, respectively.

Evaluation Metrics. For both attacks, we first obtained individual client gradients through the methodology described in Section 5. Then, we performed an analytical gradient inversion attack, using leakage from the first dense layer of the network (See Section 3.2 for more details). We evaluated the accuracy of our image reconstructions by using the Pearson correlation coefficient and the peak-signal-to-noise ratio (PSNR) value between the reconstructed image and the original image.

The Pearson coefficient is invariant against both scaling and adding values by a constant, and can be used to show that there is a linear relationship between target and reconstructed samples. It ranges from -1, indicating a perfectly negative correlation, to +1, indicating a perfectly positive correlation. PSNR is a metric to evaluate the quality between an original and derived image. It is measured in decibels (dB). The higher the PSNR value, the better the quality of the derived image is relative to the original image. Prior gradient inversion attacks in the standard FL setting use these metrics [19, 58, 62]. We considered a reconstruction to be fully revealing of the original input if the Pearson coefficient between the two images was ≥ 0.98. This threshold accounts for floating-point calculation errors and is in line with prior works [19].

For our experiments, these metrics were calculated in the following manner. First, from each of the rows in our weight gradient, we were able to extract a single partial reconstruction of the target image. We took the best partial reconstruction among all the images in the target client's local dataset. Then, we repeated this procedure with 30 distinct seed values. The plots in Section 6.2 display the maximum and average Pearson coefficients, PSNR values, and number of completely leaked target images for these 30 trials.

6.2 Experimental Results

Rolling Model Attack. Figures 4 and 5 show the maximum and average Pearson coefficients obtained from the 30 trials for the MNIST and CIFAR-10 datasets, respectively. The best reconstruction has a nearly perfect correlation with the original image across varying local dataset sizes. Furthermore, the Pearson coefficient for the averaged 30 trials also consistently remains above 0.78, indicating a very close match. The PSNR values shown in Figure 6 are also quite high. The maximum PSNR value remains above 60 dB, while the average PSNR value stays around 40 dB for both datasets. This indicates that the quality of the reconstructed images is very good relative to the original images in the client's local dataset. Figure 7 shows that the target images and reconstructions are close to a perfect match.

Figure 8 shows that in the best case, all private training images can be fully recovered, and in the average case, about half of all private training images can be fully recovered for small local dataset sizes of up to 10 images. As the local dataset size increases beyond 10 images, the percentage of fully revealed target images relative to the size of the client's local dataset tends to decrease. However, it is important to note that reconstructions may leak a substantial amount of information about the target image, despite being classified as not fully revealing the original sample.

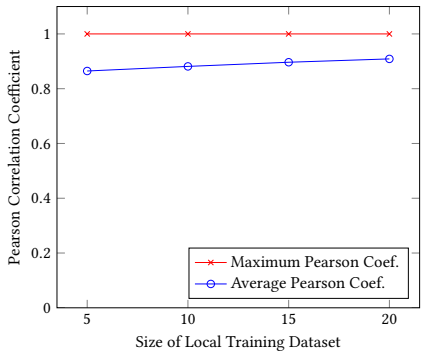

Figure 4: Maximum and average Pearson correlation coefficients, using 30 distinct seeds, for the best Rolling Model Attack partial reconstructions on the MNIST dataset. Clients trained for 1 local epoch using a batch size equal to the local dataset size.

Convergence Rate Attack. Figure 9 shows the maximum and average Pearson coefficients for the Convergence Rate Attack across 30 trials. These coefficients are between 0.87 and 1.0 in the best case and fall roughly between 0.6 and 0.8 in the average case. Figure 10 shows the maximum and average PSNR values across varying local dataset sizes. This figure shows that the quality of the reconstructions tends to decrease slightly as a client's local dataset size increases. Interestingly, the PSNR values for the CIFAR-10 dataset appear to be higher than the PSNR values for the MNIST dataset. In the best case for CIFAR-10, the best PSNR values stay above 70 dB, while on average these values remain higher than 20 dB. For the MNIST dataset, the PSNR values exceed 15 dB in the best case and remain above 12 dB in the average case. Figure 11 shows the original images and best reconstructions for the images in the local dataset of the target client in cohort B. While some of the reconstructed images are a little blurry, one can easily tell that they all represent the number 2. Furthermore, most of the features of the images are preserved in the reconstructions (e.g. the loops in the first two reconstructions).

Rolling Model Attack Noise Addition. With the standard Rolling Model attack, outlined in Section 5.2, the server distributes the same global model parameters to each non-target cohort. However, this technique is easily detectable by clients in these cohorts, since they must simply observe that the global model is the same across rounds. To address this problem, we investigated the effect of adding noise to the distributed models. The initialization of the noised model parameters follows the uniform distribution of Equation (7). Figures 12, 13, and 14 show how the Pearson coefficient, PSNR value, and number of completely leaked images, respectively, vary as a function of the noise percentage, p. As p increases, it becomes harder for clients to detect that the model parameters have been maliciously initialized. We use a local training size of 10 for Figures 12, 13, and 14.

Figure 12 shows how the Pearson coefficients vary as the noise percentage increases. As one might expect, when a larger percentage of noise is added, the Pearson coefficient tends to decrease. However, even when the noise percentage is relatively substantial,

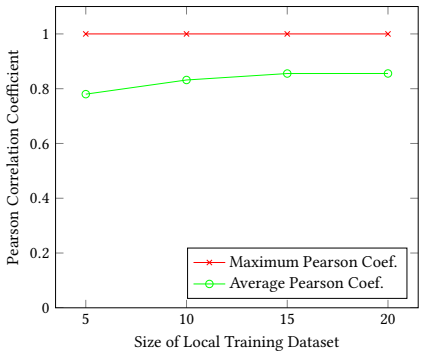

Figure 5: Maximum and average Pearson correlation coefficients, using 30 distinct seeds, for the best Rolling Model Attack partial reconstructions on the CIFAR-10 dataset. Clients trained for 1 local epoch using a batch size equal to the local dataset size.

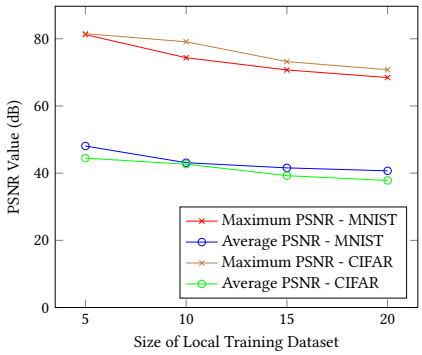

Figure 6: Maximum and average PSNR values using 30 distinct seeds, for the best Rolling Model Attack partial reconstructions. Clients trained for 1 local epoch using a batch size equal to the local dataset size.

at 30%, the Pearson coefficient still remains above 0.95 in the best case and dips just below 0.8 in the average case. Figure 13 shows how the PSNR value is affected by the percentage of noise added. This value remains above 20 and 15 dB, respectively, for the best and average cases. Finally, Figure 14 shows the number of fully recoverable images with varying levels of noise added. The same Pearson coefficient threshold value of 0.98 is used for this figure,

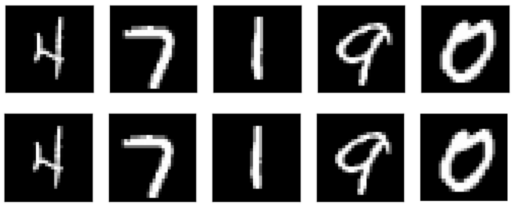

Figure 7: Original images (top row) and their respective reconstructions (bottom row) from the Rolling Model Attack.

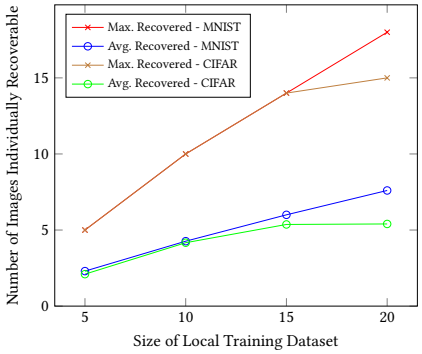

Figure 8: Maximum and average number of images fully recovered, using 30 distinct seeds, for the best Rolling Model Attack partial reconstructions. Clients trained for 1 local epoch using a batch size equal to the local dataset size. An image is classified as fully recovered if the Pearson coefficient between the reconstructed and original image is ≥ 0.98.

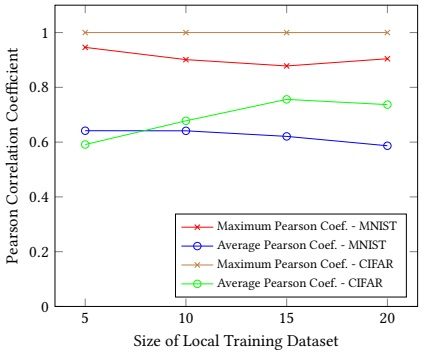

Figure 9: Maximum and average Pearson correlation coefficients, using 30 distinct seeds, for the best Convergence Rate Attack partial reconstructions. Clients trained for 1 local epoch using a batch size equal to the local dataset size.

and the maximum number of fully recoverable images is 10. As Figure 14 shows, the number of images that are completely revealed tends to decrease as the percentage of noise is increased. With a noise percentage of 25 percent or higher, none of the reconstructions have a Pearson coefficient of 0.98 or higher with respect to the target images. However, note that Figure 12 indicates that much of the information contained in the target images is still leaked.

7 DISCUSSION

7.1 Root Cause of Vulnerabilities

We believe that the root cause of the vulnerabilities exploitable by our proposed attacks is the power imbalance between clients and the server. This sentiment is in line with the work of Boenisch et al. [8]. Specifically, clients must trust the server to distribute the true global parameters each round. Moreover, clients may not have an easy way to detect that the server is acting maliciously. This power

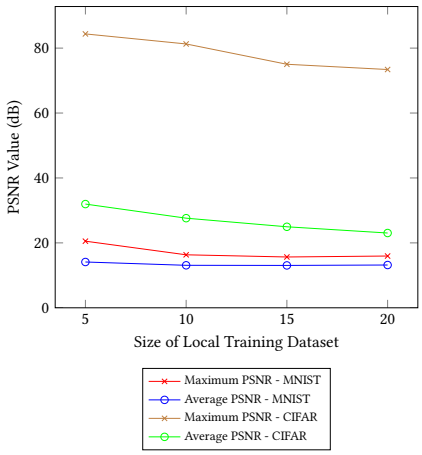

Figure 10: Maximum and average PSNR values, using 30 distinct seeds, for the best Convergence Rate Attack partial reconstructions. Clients trained for 1 local epoch using a batch size equal to the local dataset size.

Figure 11: Original images (top row) and their reconstructions (bottom row) from the Convergence Rate Attack.

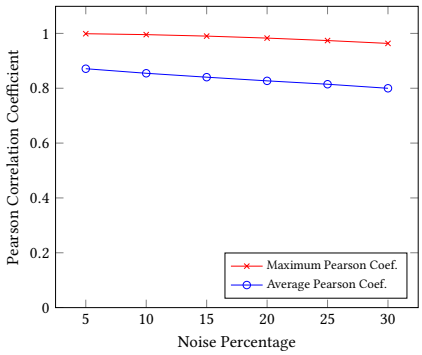

Figure 12: Maximum and average Pearson correlation coefficients, using 30 distinct seeds, for the best Rolling Model Attack partial reconstructions on the MNIST dataset. Clients trained for 1 local epoch using a batch size equal to the local dataset size. The percentage of noise added is shown on the x-axis.

imbalance allows the server to break the assumptions underlying cryptographic primitives like secure aggregation.

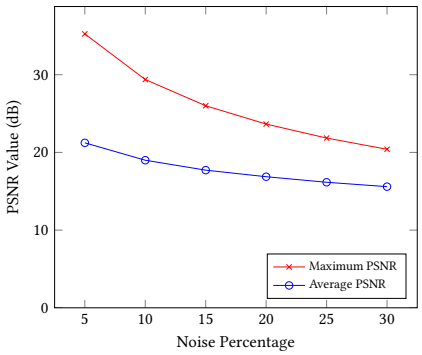

Figure 13: Maximum and average PSNR values, using 30 distinct seeds, for the best Rolling Model Attack partial reconstructions on the MNIST dataset. Clients trained for 1 local epoch using a batch size equal to the local dataset size. The percentage of noise added is shown on the x-axis.

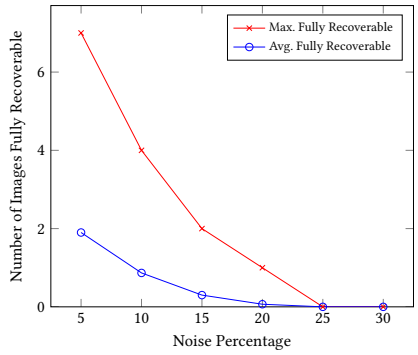

Figure 14: Maximum and average number of images fully recovered, using 30 distinct seeds, for the best Rolling Model Attack partial reconstructions on the MNIST dataset. An image is classified as fully recovered if the Pearson coefficient between the reconstructed and original image is ≥ 0.98. Clients have local dataset sizes of 10, and clients trained for 1 local epoch with a batch size of 10. The percentage of noise added is shown on the x-axis.

Model-Heterogeneous FL Setting. As we have shown, the model-heterogeneous FL setting only exacerbates security issues present in model-homogeneous FL. Specifically, the computational heterogeneity between clients and the method of submodel distribution introduce additional vulnerabilities that are not exploitable in the model-homogeneous FL setting.

For the Convergence Rate Attack in the model-homogeneous FL setting, the server would need to wait for all clients to converge before manipulating the global model for a target user. However, in the model-heterogeneous FL setting, the server simply needs to wait for the target cohort and all cohorts with more computational resources to converge before the attack can be carried out. Since the rate of convergence for different cohorts may be vastly different [2, 38, 55], the Convergence Rate Attack can be implemented potentially very early in the training process.

Similarly, the Rolling Model Attack does not work in the model-homogeneous FL setting. In this setting, each client updates the entire global model. Therefore, there is no way for the server to form a set of equations where an individual client's gradient update can be isolated. Conversely, with the rolling model scheme presented in Section 5.2, the server is able to observe the aggregated value of model parameters with and without the target client's update.

7.2 Practicality and Impact of Attacks

Convergence Rate Attack Practicality. First, training in the FL protocol continues until the global model converges [37, 44]. Furthermore, Zhou et al. [68] prove that model-heterogeneous FL schemes converge to a stationary point, for both IID and non-IID data, given a general smooth cost function. Finally, [38, 55] demonstrate that clients with greater computational capabilities will likely make more training progress than slower clients. These works demonstrate that our assumptions for the Convergence Rate Attack are valid and applicable to real-world settings.

The practicality of the Convergence Rate Attack increases significantly when the computational discrepancy between clients is large. When there is a lot of computational heterogeneity between clients, the global epochs at which different cohorts converge will vary more substantially than when clients all have similar computational resources. Therefore, the server will be able to conduct the Convergence Rate Attack earlier in the training process and over more rounds. Hence, the Convergence Rate Attack is likely to be most effective in the setting where various IoT devices participate in the FL training process. IoT devices are resource-limited in terms of both compute power and on-device memory [28, 40, 64]. Therefore, when IoT devices train alongside mobile phones and resource-heavy servers, the computational resources of client devices will vary significantly.

IoT devices are already being used for various FL applications, and their use is only expected to increase in the future [28, 64]. For example, IoT devices have applications to healthcare. Today, smart devices can be used to measure a patient's heart rate, blood pressure, glucose levels, etc. Using this data, IoT devices can participate in FL protocols designed to train machine learning (ML) models for patient monitoring and treatment. Another FL application for IoT devices is with regard to the smart city. Specifically, IoT devices collect data related to transportation, public safety, smart agriculture, etc. This data can be used to help government officials make better decisions with regard to infrastructure, disease prevention and mitigation, etc. Smart homes also rely on IoT devices like smart cameras, smart doorbells, and smart bulbs. These devices collect a large amount of raw data, which can be used to train ML models. Additionally, virtual assistants, like Siri and Alexa, have become commonplace in many households. It may be valuable for IoT devices in smart homes to collaboratively train ML models through a FL protocol [64]. FL is responsible for enabling a multitude of important IoT applications. Furthermore, due to the resource constraints of IoT devices, it is quite plausible that these devices will participate in model-heterogeneous FL schemes. Therefore, the Convergence Rate Attack has broad implications to privacy.

Rolling Model Attack Practicality. The Rolling Model Attack relies on fewer assumptions than the Convergence Rate Attack and can also be conducted earlier. In fact, the Rolling Model Attack can be used to reconstruct private data samples of a target client as early as the second global iteration. This means that clients must be able to detect that the server maliciously initialized model parameters and abort from the FL protocol before the second round of training. This is quite difficult, since clients have no way of knowing the model updates of other clients. Furthermore, the server can add noise to the distributed models, making it harder for clients to detect malicious behavior. As was shown in Section 6.2, even a relatively high noise percentage does not impact the server's ability to reconstruct data samples from a target client.

7.3 Potential Countermeasures

Countermeasures against the Convergence Rate Attack and Rolling Model Attack involve reducing the power imbalance between clients and the server. This can be achieved through decentralization of the FL process, adding more hardware support, differential privacy, and encryption of model parameters. With each of these techniques, clients can limit the amount of trust they must place in the server, thus reducing the power of the server.

Decentralization. First, decentralized schemes can be adopted to reduce the dependence of clients on the central server. For example, Shayan et al. propose a blockchain-based FL protocol named Biscotti [52]. Biscotti uses the Proof-of-Federation protocol, which is similar to Proof-of-Stake. However, the reliability of a client with the Proof-of-Federation protocol is determined by the quality of their model updates and their contribution to the consensus process. To protect the privacy of individual client model updates, Biscotti uses masking and secure aggregation. Wang et al. propose BPFL, a privacy-preserving FL scheme that uses the blockchain [56]. In BPFL, several different entities are involved, including FL nodes, who download the global model and perform local model updates, and model aggregation nodes, who aggregate local model updates. BPFL also makes use of homomorphic encryption for privacy-preserving model aggregation. However, the expensive cryptographic protocols used in blockchain-based algorithms may impose a large overhead. Furthermore, since clients may be resource-limited in the model-heterogeneous FL setting, it will be difficult for clients to perform intensive storage and computational tasks necessary for most blockchain schemes. Finally, since clients in the model-heterogeneous FL setting may have non-IID data, it is difficult to determine the quality of the model update for a given client. Hence developing an efficient incentive mechanism is a major challenge.

Secure Hardware Support. Another way that the power imbalance between clients and the server can be reduced is to make use of additional hardware support. Specifically, Trusted Execution Environments (TEEs) can be utilized to hide gradient updates from the adversary. A TEE provides a strongly isolated secure compartment for executing code, such that even privileged code cannot gain access to the secure compartment. Hence, TEEs can provide strong confidentiality and integrity guarantees. Mo et al. [39] propose a privacy-preserving FL scheme using TEEs. To address the memory constraints associated with TEEs, Mo et al. proposed a greedy, layer-wise training and aggregation scheme. In

their scheme, clients use TEEs for local training and the server utilizes a TEE for aggregation. However, in the model-heterogeneous FL setting, it may be difficult to use TEEs, since adding specialized hardware to resource-constrained devices may be infeasible in practice. Furthermore, TEEs impose additional computational, memory, and energy consumption overhead [39]. Therefore, the use of TEEs would most likely worsen system heterogeneity issues present in model-heterogeneous FL schemes.

Differential Privacy (DP). Another technique to prevent the server from breaking the confidentiality of client updates is DP. While centralized DP is incompatible with our threat model, since it relies on a trusted server, Local DP (LDP) may be an effective way to prevent the server from observing the gradients of target clients. With LDP, clients perturb their local gradient updates before sending these updates to the aggregator [67]. If clients add a sufficient amount of noise to their local updates, the central server will be unable to observe individual client gradients in the clear. However, with any Local DP scheme, there is a tradeoff between privacy and utility. When clients add a large amount of noise to their local model updates, they can expect better privacy guarantees, but the overall utility of the global model may be degraded. For example, in complex machine learning models, the range of the weight values at different layers may vary substantially. As many existing LDP schemes assume that the range of these weights are fixed, the noise is not adaptively modified for each layer. Hence, the utility of the shared global model may be greatly degraded [54].

Another concern with Local DP is large resource consumption, including communication and energy overhead. Specifically, the degradation to global model utility may necessitate a larger number of communication rounds between the clients and the server in order for the global model to converge. Hence, the wall-clock time necessary for model convergence may increase significantly [51]. In order to address the communication overhead, Wei et al. [57] propose the communication rounds discounting (CRD) algorithm, which allows the server to adjust the number of communication rounds during training. This algorithm aims to achieve a good trade-off between privacy protection and global model convergence.

Distributed DP (DDP) is another solution for addressing the gradient leakage of individual clients. In DDP schemes, clients add a small amount of noise to their individual updates. The amount of noise added by clients is not sufficient to protect their local model update from leaking sensitive information. However, when combined with secure aggregation, the server is unable to extract information from individual client updates [3]. While DDP schemes provide better global model utility than LDP [15], the majority of clients must be honest. If this assumption is violated, then the server may be able to break the privacy guarantees of DDP schemes [8]. Furthermore, the server will still be able to distribute inconsistent models to users in order to force the final aggregated value to be a function of only a single target client's private dataset [45].

Secure Computation. Finally, clients can choose to participate in FL schemes where the output from secure aggregation is encrypted. Due to this encryption, the server will never have access to the aggregate update value, and will therefore be unable to compromise the privacy of individual clients. A number of works [25, 43, 65] propose such schemes. These schemes make use of homomorphic encryption, which allows arithmetic operations to be performed directly on ciphertexts. Hence, a central server is able to aggregate the encrypted client updates without ever needing to decrypt these updates. Zhang et al. [65] also incorporate verification into their FL scheme. The verification step ensures that clients are able to detect when the server falsifies the aggregated model. However, homomorphic encryption schemes impose a large performance and communication overhead. The authors of [63] show that FL schemes with homomorphic encryption have significantly longer training times and substantially more data transfer than FL schemes without homomorphic encryption. Since clients in the model-heterogeneous FL setting may have limited computational resources, homomorphic encryption schemes may not be suitable to this setting.

7.4 Future Work

Better Mitigation Strategies. One direction of future work would be to find more effective countermeasures against the Convergence Rate and Rolling Model Attacks. While Section 7.3 discusses several countermeasures that can be employed against these attacks, each of the countermeasures has some drawbacks. Therefore, finding low-overhead mitigations that do not degrade utility is an important area for future work.

Increasing Attack Effectiveness. Another area of future work would be to find ways to improve the effectiveness of our proposed attacks. For example, our proposed attacks may be combined with schemes that break secure aggregation in the model-homogeneous FL setting, like [8, 45], in order to be more effective. One challenge is to identify other ways that the computational and storage heterogeneity of client devices can be exploited by the aggregation server. Another aspect of improving our attacks would be to reduce their detectability. As such, an important area of future work would be to investigate what noisy parameter initialization strategies can be employed by the server to reduce the probability that clients detect the server's malicious behavior.

8 CONCLUSION

In this work, we propose two attacks that exploit characteristics of the model-heterogeneous FL setting. By targeting the distribution pattern of model-heterogeneous FL schemes and the heterogeneity of client devices, we show that a malicious server can obtain the plaintext gradient from a target cohort. We empirically demonstrate the effectiveness of our attacks in allowing an adversary to reconstruct data samples from a victim client. Finally, we discuss potential countermeasures to our proposed attacks. We hope that our work encourages researchers to improve the confidentiality guarantees of existing model-heterogeneous FL schemes.

ACKNOWLEDGMENTS

We thank our shepherd and anonymous reviewers for their valuable feedback to improve the quality of this work. This work is based upon work supported by the Commonwealth Cyber Initiative (CCI) – an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

REFERENCES

[1] 2022. What is GDPR, the EU's new Data Protection Law? https://gdpr.eu/what-is-gdpr/

[2] Ahmed M Abdelmoniem, Chen-Yu Ho, Pantelis Papageorgiou, and Marco Canini. 2023. A comprehensive empirical study of heterogeneity in federated learning. *IEEE Internet of Things Journal* (2023).

[3] Naman Agarwal, Peter Kairouz, and Ziyu Liu. 2021. The skellam mechanism for differentially private federated learning. *Advances in Neural Information Processing Systems* 34 (2021), 5052–5064.

[4] Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. 2022. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Advances in Neural Information Processing Systems* 35 (2022), 29677–29690.

[5] Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. 2020. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access* 8 (2020), 140699–140725.

[6] Syreen Banabilah, Moayad Aloqaily, Eitaa Alsayed, Nida Malik, and Yaser Jararweh. 2022. Federated learning review: Fundamentals, enabling technologies, and future applications. *Information processing & management* 59, 6 (2022), 103061.

[7] Chaity Banerjee, Tathagata Mukherjee, and Eduardo Pasiliao Jr. 2019. An empirical study on generalizations of the ReLU activation function. In *Proceedings of the 2019 ACM Southeast Conference*. 164–167.

[8] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. 2023. Reconstructing Individual Data Points in Federated Learning Hardened with Differential Privacy and Secure Aggregation. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 241–257.

[9] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. 2023. When the curious abandon honesty: Federated learning is not private. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 175–199.

[10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.

[11] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. 2018. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210* (2018).

[12] Patrick Cason. 2020. Announcing 4 new libraries for Federated Learning on web and mobile devices. https://blog.openmined.org/announcing-new-libraries-for-fl-on-web-and-mobile/

[13] Yun-Hin Chan, Rui Zhou, Running Zhao, Zhihan Jiang, and Edith C-H Ngai. 2023. Internal Cross-layer Gradients for Extending Homogeneity to Heterogeneity in Federated Learning. *arXiv preprint arXiv:2308.11464* (2023).

[14] Hong-You Chen and Wei-Lun Chao. 2020. Fedbe: Making bayesian model ensemble applicable to federated learning. *arXiv preprint arXiv:2009.01974* (2020).

[15] Wei-Ning Chen, Christopher A Choquette Choo, Peter Kairouz, and Ananda Theertha Suresh. 2022. The fundamental price of secure aggregation in differentially private federated learning. In *International Conference on Machine Learning*. PMLR, 3056–3089.

[16] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 15–24.

[17] Yae Jee Cho, Andre Manoel, Gauri Joshi, Robert Sim, and Dimitrios Dimitriadis. 2022. Heterogeneous ensemble knowledge transfer for training large models in federated learning. *arXiv preprint arXiv:2204.12703* (2022).

[18] Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264* (2020).

[19] David Enthoven and Zaid Al-Ars. 2022. Fidel: Reconstructing private training samples from weight updates in federated learning. In *2022 9th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 1–8.

[20] Boyu Fan, Siyang Jiang, Xiang Su, and Pan Hui. 2023. Model-Heterogeneous Federated Learning for Internet of Things: Enabling Technologies and Future Directions. *arXiv preprint arXiv:2312.12091* (2023).

[21] Liam Fowl, Jonas Geiping, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. 2021. Robbing the fed: Directly obtaining private data in federated learning with modified models. *arXiv preprint arXiv:2110.13057* (2021).

[22] Dashan Gao, Xin Yao, and Qiang Yang. 2022. A survey on heterogeneous federated learning. *arXiv preprint arXiv:2210.04505* (2022).

[23] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems* 33 (2020), 16937–16947.

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

[25] Neveen Mohammad Hijazi, Moayad Aloqaily, Mohsen Guizani, Bassem Ouni, and Fakhri Karray. 2023. Secure federated learning with fully homomorphic encryption for iot communications. *IEEE Internet of Things Journal* (2023).

[26] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. 2021. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems* 34 (2021), 12876–12889.

[27] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[28] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M Hadi Amini. 2021. A survey on federated learning for resource-constrained IoT devices. *IEEE Internet of Things Journal* 9, 1 (2021), 1–24.

[29] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n. d.]. The CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html

[30] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 19–35.

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

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

[33] Ang Li, Jingwei Sun, Pengcheng Li, Yu Pu, Hai Li, and Yiran Chen. 2021. Hermes: an efficient federated learning framework for heterogeneous mobile clients. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 420–437.

[34] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. 2020. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371* (2020).

[35] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine* 37, 3 (2020), 50–60.

[36] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. 2020. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 2351–2363.

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

[38] Aritra Mitra, Rayana Jaafar, George J Pappas, and Hamed Hassani. 2021. Linear convergence in federated learning: Tackling client heterogeneity and sparse gradients. *Advances in Neural Information Processing Systems* 34 (2021), 14606–14619.

[39] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*. 94–108.

[40] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. 2021. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 23, 3 (2021), 1622–1658.

[41] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 3581–3607.

[42] Takayuki Nishio and Ryo Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE international conference on communications (ICC)*. IEEE, 1–7.

[43] Jaehyoung Park and Hyuk Lim. 2022. Privacy-preserving federated learning using homomorphic encryption. *Applied Sciences* 12, 2 (2022), 734.

[44] Francesco Pase, Marco Giordani, and Michele Zorzi. 2021. On the convergence time of federated learning over wireless networks under imperfect CSI. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 1–7.

[45] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. 2022. Eluding secure aggregation in federated learning via model inconsistency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2429–2443.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[47] Kilian Pfeiffer, Martin Rapp, Ramin Khalili, and Jörg Henkel. 2023. Federated learning for computationally constrained heterogeneous devices: A survey. *Comput. Surveys* 55, 14s (2023), 1–27.

[48] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2017. Privacy-preserving deep learning: Revisited and enhanced. In *Applications and Techniques in Information Security: 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6–7, 2017, Proceedings*. Springer, 100–110.

[49] KM Jawadur Rahman, Faisal Ahmed, Nazma Akhter, Mohammad Hasan, Ruhul Amin, Kazi Ehsan Aziz, AKM Muzahidul Islam, Md Saddam Hossain Mukta, and AKM Najmul Islam. 2021. Challenges, applications and design aspects of federated learning: A survey. *IEEE Access* 9 (2021), 124682–124700.

[50] Felix Sattler, Tim Korjakow, Roman Rischke, and Wojciech Samek. 2021. Fedaux: Leveraging unlabeled auxiliary data in federated learning. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[51] Mohamed Seif, Ravi Tandon, and Ming Li. 2020. Wireless federated learning with local differential privacy. In *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2604–2609.

[52] Muhammad Shayan, Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2020. Biscotti: A blockchain system for private and secure federated learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2020), 1513–1525.

[53] Ningxin Su and Baochun Li. 2022. How asynchronous can federated learning be?. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–11.

[54] Lichao Sun, Jianwei Qian, and Xun Chen. 2020. LDP-FL: Practical private aggregation in federated learning with local differential privacy. *arXiv preprint arXiv:2007.15789* (2020).

[55] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. 2020. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems* 33 (2020), 7611–7623.

[56] Naiyu Wang, Wenti Yang, Zhitao Guan, Xiaojiang Du, and Mohsen Guizani. 2021. Bpfl: A blockchain based privacy-preserving federated learning scheme. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[57] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Hang Su, Bo Zhang, and H Vincent Poor. 2021. User-level privacy-preserving federated learning: Analysis and performance optimization. *IEEE Transactions on Mobile Computing* 21, 9 (2021), 3388–3401.

[58] Yuxin Wen, Jonas Geiping, Liam Fowl, Micah Goldblum, and Tom Goldstein. 2022. Fishing for user data in large-batch federated learning via gradient magnification. *arXiv preprint arXiv:2202.00580* (2022).

[59] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).

[60] Zirui Xu, Fuxun Yu, Jinjun Xiong, and Xiang Chen. 2021. Helios: Heterogeneity-aware federated learning with dynamically balanced collaboration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 997–1002.

[61] Mang Ye, Xiuwen Fang, Bo Du, Pong C Yuen, and Dacheng Tao. 2023. Heterogeneous federated learning: State-of-the-art and research challenges. *Comput. Surveys* 56, 3 (2023), 1–44.

[62] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. 2021. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16337–16346.

[63] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. 2020. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*. 493–506.

[64] Tuo Zhang, Lei Gao, Chaoyang He, Mi Zhang, Bhaskar Krishnamachari, and A Salman Avestimehr. 2022. Federated learning for the internet of things: Applications, challenges, and opportunities. *IEEE Internet of Things Magazine* 5, 1 (2022), 24–29.

[65] Xianglong Zhang, Anmin Fu, Huaqun Wang, Chunyi Zhou, and Zhenzhu Chen. 2020. A privacy-preserving and verifiable federated learning scheme. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[66] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2020. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610* (2020).

[67] Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Dusit Niyato, and Kwok-Yan Lam. 2020. Local differential privacy-based federated learning for internet of things. *IEEE Internet of Things Journal* 8, 11 (2020), 8836–8853.

[68] Hanhan Zhou, Tian Lan, Guru Prasadh Venkataramani, and Wenbo Ding. 2024. Every parameter matters: Ensuring the convergence of federated learning with dynamic heterogeneous models reduction. *Advances in Neural Information Processing Systems* 36 (2024).

[69] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. *Advances in neural information processing systems* 32 (2019).