Learned Model Compression for Efficient and Privacy-Preserving Federated Learning

Yiming Chen , Lusine Abrahamyan , Hichem Sahli , and Nikos Deligiannis , *Member, IEEE*

***Abstract*—Federated learning performs collaborative training of deep learning models among multiple clients, safeguarding data privacy, security, and legal adherence by preserving training data locally. While the training data remains stored in the client side, instead of being shared with the server and other clients, recent work has shown that the data can still be reconstructed by local updates or gradients. Different defense techniques have been proposed to address this information leakage from the gradient or updates, including introducing noise to gradients, performing gradient compression (such as sparsification), and feature perturbation. However, these methods either impede model convergence, impose restrictions on model architecture incur, or entail substantial communication costs. Furthermore, balancing model performance, communication cost and privacy preservation remains a challenging trade-off. To tackle the information leakage during collaborative training, we introduce an adaptive autoencoder-based method for compressing and, thus, perturbing the model parameters. The client utilizes an autoencoder to acquire the representation of the local model parameters within few local iterations, and then shares the compressed model parameters with the server, rather than the true model parameters. The use of the autoencoder for lossy compression serves as an effective protection against information leakage from the updates. Additionally, the perturbation is intrinsically linked to the autoencoder's input, thereby achieving a perturbation with respect to the parameters of different layers. Moreover, our approach can reduce 4.1 $\times$ the communication rate compared to federated averaging. We empirically validate our method using two widely-used models within the context of federated learning, considering three datasets, and assess its performance against several well-established defense frameworks. The results indicate that our approach attains a model performance nearly identical to that of unmodified local updates, while effectively preventing information leakage and reducing communication costs in comparison to other methods, including noisy gradients, gradient sparsification, and PRECODE.**

***Index Terms*—Deep learning, federated learning, data privacy, gradient compression, autoencoder.**

I. INTRODUCTION

FEDERATED learning (FL) [1] is a promising framework to address the challenges of privacy and data security when training AI models for applications in, for example, mobile internet [2], healthcare [3], and internet of things [4], [5]. In the traditional centralized learning, collecting sensitive data on a central server raises concerns in terms of privacy and data ownership. Instead, FL enables collaborative model training across multiple decentralized clients while ensuring the privacy and security of individual data, as Fig. 1 depicted. For example, Google improved predictive text suggestions in their G-keyboard using FL, without uploading users' information [2].

However, recent studies have shown that private user data can still be leaked from the exchanged updates of model parameters in FL, despite being solely kept on individual client devices. This can happen through the property inference attack [6], [7], membership inference attack [8], and gradient inversion attack [9]–[11]. The property inference attack infers the dataset's attributes by analyzing the model parameters' updates. The membership inference attack speculates whether the sample exists in the training set and is involved in the training process. The gradient inversion attack directly reconstructs the input data using gradients or model parameter updates, resulting in a lot of sensitive information leaking; especially, it has been shown that a single input to a fully connected network can always be reconstructed analytically [10]. Given the ubiquity of fully-connected layers in various models, the issue of privacy leakage becomes particularly grave.

To overcome information leakage, especially by gradient inversion attacks, some defense methods have been proposed. Secure multi-party computation (SMC) methods [12]–[14] collaboratively aggregate their sensitive local updates into global updates without revealing those updates to the malicious server. However SMC inflicts a significant computational overhead, especially on complex systems [15]. Differential privacy (DP) methods [9], [16]–[20] add calibrated noise to the updates before being shared. The perturbed local updates ensure that these gradients do not reveal precise information about any individual training example, thereby providing privacy guarantees. However, DP faces an inherent tradeoff between model performance and information leakage and requires a number of participating clients to ensure model convergence. Gradient compression-based methods [21]–[23] remove information from the updates of the model parameters, obstructing the reconstruction of input. This approach, however, can only achieve acceptable defense with high compression rates, which might jeopardize the performance of the trained model. Other studies [24], [25] proposed perturbing the data representation to protect privacy. However, adversaries can potentially circumvent the gradient perturbation by discarding the perturbed layer. Alternatively, PRivacy EnhanCing mODulE (PRECODE) [25] inserted a variational bottleneck [26] into the model to perturb the representation by sampling. Nevertheless, this approach requires modifying the model architecture,

This research received funding from Research Foundation – Flanders (FWO) research project G093817N.

Y. Chen, H. Sahli, and N. Deligiannis are with Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussels, Belgium and also with imec, Kapeldreef 75, B-3001 Leuven, Belgium. (e-mail: {cyiming, hsahli, ndeligia}@etrovub.be).

L. Abrahamyan was with Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussels, Belgium and also with imec, Kapeldreef 75, B-3001 Leuven, Belgium. She is now with BeVi - Best View, Vienna, Austria (e-mail: lusine.abrahamyan@vub.be).

breaking the integrity of the network. Therefore, it is imperative to design an innovative defense framework that effectively conceals sensitive training data information within gradients from any layer rather than from the last few layers [24]. Such a framework should safeguard gradient inversion attacks without requiring modifications to the model architecture. Furthermore, it should not harm the final model performance or the training process, causing, for example, prolonged convergence times or increasing the communication costs.

Our approach is inspired by our prior work [27], [28], which utilized a lightweight autoencoder to compress gradients in distributed training of deep learning models. In this work, which targets federated learning, we use a lightweight autoencoder to learn compressed representations of local model parameters. The lossy compression due to the autoencoder reduces the communication cost associated with the model parameter exchange and acts as an adaptive perturbation. This perturbation is learned directly from the model parameters rather than manually adjusted as in the DP approach. Moreover, as all layers undergo compression in our approach, circumventing the perturbed layer to reconstruct the training data is effectively thwarted [24]. Besides, by sharing the compressed model parameters with the server, our method significantly reduces the communication rate ($4.1\times$ less with respect to vanilla FL) compared to other defense approaches. We empirically validate our proposed method using two classical models in FL. We train the models using three datasets, both partitioned in independent and identically distributed (IID) and non-independent and non-identically distributed (non-IID) manner. Our method's performance is compared against several established defense mechanisms, including DP, gradient sparsification, and PRECODE. The results indicate that our approach achieves a model performance nearly identical to unmodified local updates while effectively preventing the risk of information leakage and significantly reducing the communication rate compared to other methods such as noisy gradient, gradient sparsification, and PRECODE.

The remainder of the paper reads as follows: Section II discusses background and reviews related work, Section III elaborates on the proposed framework, Section IV presents our experiments, and Section V draws our conclusion.

II. BACKGROUND AND RELATED WORK

This section discusses background and related work in federated learning (Section II-II-A), gradient leakage (Section II-II-B), defence mechanisms (Section II-II-C) and compression in FL (Section II-II-D).

A. THE FEDERATED AVERAGE APPROACH

Assume a collection of decentralized clients participating in the FL process [1], indexed by $k = 1, 2, \ldots, K$. Each client has a local dataset $\mathcal{D}_k$. The objective is to train a global deep learning model $f(\theta; \cdot)$, which consists of L layers, by optimizing its parameters θ. The layer's parameters are indexed by the corresponding layer's id $l = 1, \ldots, L$. The training process utilizes data from distributed clients and involves message exchanges across T communication rounds. *Federated Averaging* (FedAvg) [1] reduces the information transferred

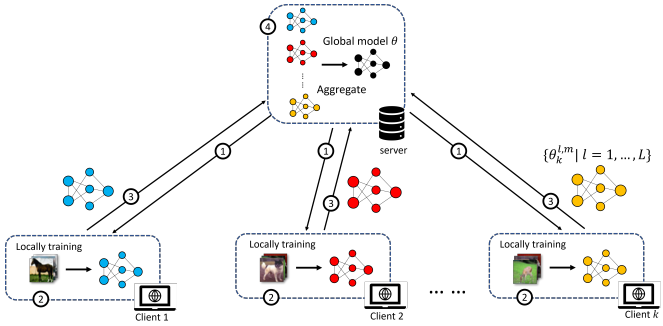


Fig. 1. Illustration of federated learning, where θ represents the global model maintained by the server, and $\{\theta_k^{l,m} | l = 1, \ldots, L\}$ denotes a set of model parameters indexed by the layer id l, which are locally updated on the client k over m iterations. FL involves the following steps: 1) The server shares the global model with participants. 2) Each client updates the global model using its private data for m iterations, as expressed in (2). 3) The clients transmit the updated model to the server. 4) The server aggregates all received models to form a global model for the subsequent communication round.

during training by performing multiple stochastic gradient descent (SGD) updates locally at the client before sending the aggregated model update to the server. This optimization problem can be expressed as follows:

$$\underset{\theta}{\operatorname{argmin}} \sum_{k=1}^{K} \frac{|\mathcal{D}_k|}{|\mathcal{D}|} \cdot \mathcal{L}(f(\theta; \cdot); \mathcal{D}_k), \tag{1}$$

where $|\cdot|$ denotes the cardinality of a set, and $\mathcal{L}$ is the loss function. To facilitate communication, at each training round, a subset C of $c\%$ of the total number of clients participates in collaborative learning. The server shares the global model with the selected clients in C, which in turn perform local updates of each layer l in each iteration $i = 1, \ldots, m$ by optimizing locally the loss function using their local dataset $\mathcal{D}_k$:

$$\theta_k^{l,i+1} = \theta_k^{l,i} - \eta \nabla_{\theta_k^{l,i}} \mathcal{L}(f(\{\theta_k^{l,i} | l = 1, \ldots, L\}; \cdot), \mathcal{D}_k^i), \tag{2}$$

with η and i denoting the learning rate and the iteration index, respectively, and $\mathcal{D}_k^i$ and $\{\theta_k^{l,i} | l = 1, \ldots, L\}$ being the mini-batch at iteration i and a set of model parameters indexed by layer's id l. Then, the clients send the set of updated parameters $\{\theta_k^{l,m} | i = 1, \ldots, L\}$ back to the server, which then updates the global model parameter of each layer l as follows:

$$\frac{1}{|C|} \sum_{k} \theta_k^{l,m}. \tag{3}$$

B. PRIVACY LEAKAGE IN FEDERATED LEARNING

A malicious server can still recover the local training data through the clients' local updates (or the gradient) [9]–[11], [29]–[31]. *Deep Leakage from Gradient* (DLG) [9] reconstructs the training data by minimizing the distance between the true gradient and the gradient backpropagated using the reconstructed data. This can be formulated as:

$$x', y' = \min_{(x', y')} d(\nabla \mathcal{L}(f(\theta; \cdot), x', y'), \nabla_\theta), \tag{4}$$

where x' and y' are the dummy input data and the dummy label, $\mathcal{L}$ is the model's loss function, and ∇_θ and $\nabla \mathcal{L}(f(\theta; \cdot), x', y')$ are the received gradient and the gradient obtained with the

dummy data and labels, respectively, and d denotes a distance metric (the Euclidean distance in DLG [9]), respectively. As shown in (4), a malicious server could optimize a dummy input x' and a dummy label y' for several iterations, resulting in the reconstruction of the training data without requiring any additional information. *Improved Deep Leakage from Gradients* (iDLG) [29] derives the true label from the sign of gradients in the last layer of the model, when the batch size is one. This approach replaces the dummy label y' with the true label y in (4), further enhancing the reconstruction quality. The study in [30] reported that DLG is highly sensitive to the initialization of dummy data and showed that using data from the same class as the dummy input benefits the reconstruction attack. *Inverting Gradient Attack* (IGA) [10] showed that most gradients from a trained model are very close to zero for different inputs, posing a challenge for DLG to distinguish them when using the Euclidean distance in (4). Therefore, [10] introduced the cosine distance, to measure the direction rather than the magnitude between the true gradient and the dummy gradient, and a total variation regularizer on the dummy data x'. The authors of [10] also proved that the input to any fully connected layer can be recovered analytically, which implies that the Multi-Layered Perceptron (MLP) is more vulnerable to attacks compared with other models, such as Convolutional Neural Networks (CNNs).

The aforementioned methods are not effective for large models with large batch sizes. The study in [11] introduced a solution, namely, *GradInversion*, which incorporated a group consistency regularization term into the optimization to address this limitation. They utilized multi-seed optimization and image registration techniques to improve the reconstruction quality. Some recent studies have also attacked other architectures. For instance, the authors of [31] successfully implemented the inversion of input data by leveraging gradients on Vision Transformer (ViT) models.

C. DEFENSE AGAINST PRIVACY LEAKAGE

Several defense mechanisms against information leakage in FL have been proposed. *Cryptographic protocols*, such as secure multi-party computation (SMC) [12]–[14], distribute the computation across multiple parties while preserving the privacy of each party's data. For example, [14] presented an algorithm that ensures the confidentiality of both the global model and the local updates. Cryptographic methods ensure that no individual party can access the data of other parties. However, executing complex cryptographic operations across multiple parties requires significant computational resources and results in a considerable computational overhead. *Differential privacy* (DP) methods [16] add random noise into the gradient, offering rigorous privacy protection guarantees, establishing a tradeoff between model performance and information leakage [18]. The study in [19] introduced a new DP perturbation mechanism with a time-varying noise amplitude, aiming to enhance the privacy protection of FL while simultaneously maintaining the ability to adjust the learning performance. However, DP-based methods require a substantial number of participants in the training to achieve model convergence and achieve a good trade-off between model performance and information leakage [18]. *Gradient compression* can also prevent leakage information. Gradient quantization [23] approximates gradients using a lower number of bits than full precision. The reduction in precision can lead to a loss of information, potentially impacting both the convergence of the model and the possibility of information leakage. The study of [9] revealed that using a half-precision float proposed in [32] failed to protect the training data adequately. On the other hand, utilizing a low-bit representation like INT8 successfully prevented data leakage but resulted in a significant drop in the model's performance. Gradient sparsification [21], [22] aims to reduce the number of non-zero elements in gradient updates before transmitting. By identifying and removing these small gradient values, sparsification effectively reduces the overhead of communication. However, the authors of [9] showed that only a high sparsification rate can achieve desirable defensive performance, which in turn can hinder the training convergence.

Some works have explored alternative defense approaches. Soteria [15] introduced perturbations to the data representation obtained from fully connected layers to severely degrade the reconstructed data's quality while maintaining the FL performance. However, [24] showed that a malicious server could bypass the perturbation by simply dropping the perturbed layer, resulting in a perfect reconstruction of the training data. The study of [33] introduced a transformation policy for data preprocessing, which made it infeasible for the adversary to extract sensitive information from gradients. However, the automatic transformation search imposes a significant computational overhead. The study in [25] introduced a data representation perturbation method, coined *PRivacy EnhanCing mODulE* (PRECODE), which inserts a variational autoencoder (VAE) [34] into the model architecture between the feature extractor and the fully-connected layers. The VAE bottleneck projected the data representation into a sampling space and produced a similar but distinct representation. PRECODE aims to prevent information leakage from the training data by creating a gap between the projected and true representation. However, it modifies the model architecture to accommodate the insertion of a VAE bottleneck, resulting in additional computational complexity. Moreover, [24] showed that the training data can be perfectly reconstructed from MLP models irrespective of PRECODE. Furthermore, our experiments show that the effectiveness of this approach is influenced by the assumption of non independently and identically distributed (non-IID) data. The robustness and performance of PRECODE are compromised in FL in scenarios where the data does not satisfy the IID assumption. The work discussed in [35] used a combination of compressed sensing and local DP to prevent information leakage. However, it still contends with the limitations associated with differential privacy and data compression strategies.

D. EFFICIENT COMMUNICATION IN FEDERATED LEARNING

FL involves a substantial communication cost across numerous clients and servers, resulting in bandwidth requirements and delays. Various compression techniques such as quantization

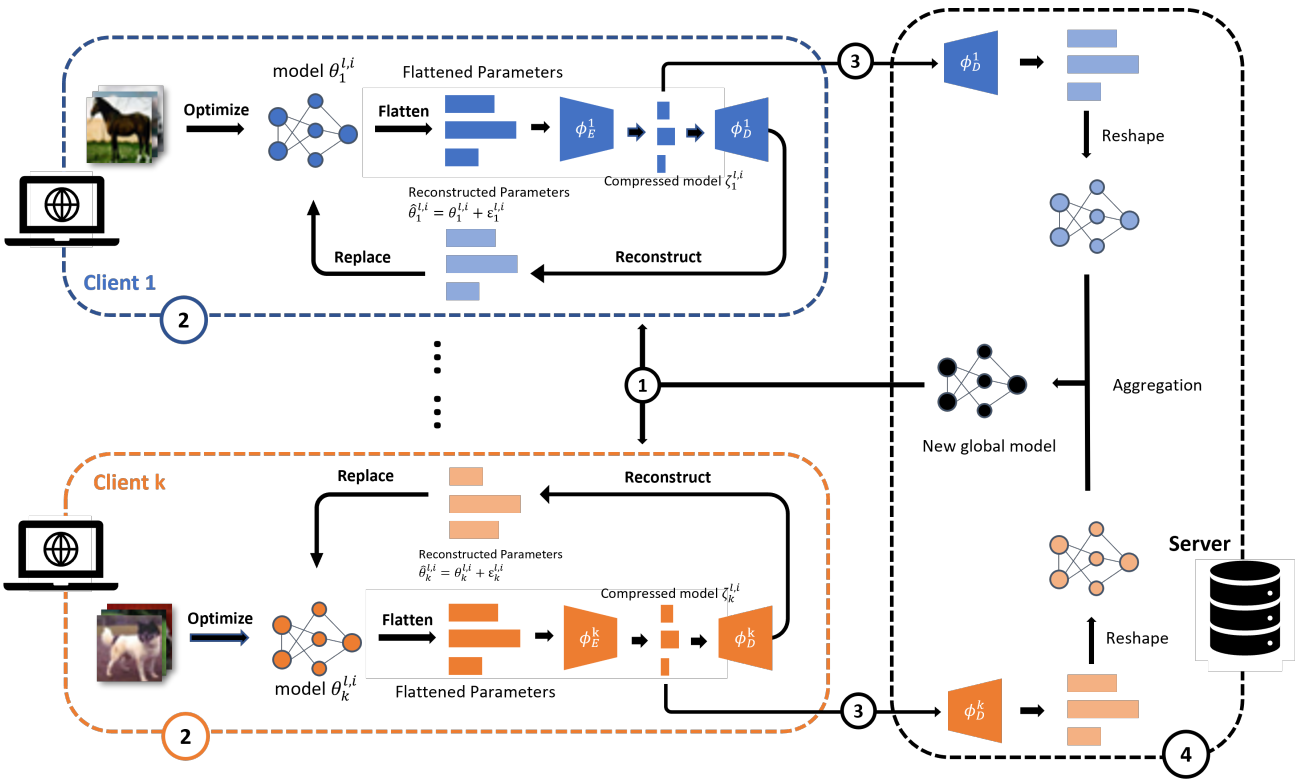


Fig. 2. Illustration of the proposed federated learning approach. It involves the following steps: 1) The server shares the global model with participants. 2) Each client updates the global model using its private data for m iterations and uses the pre-trained autoencoder to perturb (compress) the model parameters in every iteration, as expressed in eq. (11). 3) The clients share the compressed model with the server per eqs. (5) and (6). 4) The server reconstructs the model and aggregates all received models to form a global model for the subsequent communication round.

and sparsification are commonly applied on top of FedAvg [1] to reduce the information rate further. The method in [36] proposed quantizing the clients' updates before uploading to the parameter server. Alternatively, the study in [37] proposed moving momentum and error accumulation from clients to the central aggregator using a count sketch randomized data structure. Therefore, the gradients can be randomly projected several times to lower-dimensional spaces, such that high-magnitude elements can later be approximately recovered.

In our earlier research on distributed training, denoted as Learned Gradient Compression (LGC) [27], [28], we introduced a machine-learning method for compressing gradients before transmission. LGC involved training an autoencoder as the compression model using a representative set of gradients, preserving crucial information while reducing update sizes. Our work demonstrated the autoencoder's ability to compress gradients without compromising model performance, even when training deep convolutional neural networks (CNNs) like ResNet-101 [38] on large-scale datasets such as ImageNet [39]. Moreover, our work showed that an autoencoder could achieve rapid convergence with a minimal number of training iterations (around 200) in the distributed training approach. LGC focuses on enhancing communication efficiency through gradient compression in distributed training scenarios. In contrast, our emphasis in this work lies in privacy preservation achieved through lossy compression. This involves perturbing local model parameters within an autoencoder framework explicitly designed for FL.

III. PROPOSED METHOD

This section elaborates on the proposed privacy-preserving FL framework, describing the learned parameter compression method (Section III-III-A), the architecture of the autoencoder we consider (Section III-III-B), and the way we apply learned parameter compression into FL (Section III-III-C). Figure 2 depicts the overview of our method.

A. LEARNED PARAMETER COMPRESSION

In the classical FL system, discussed in Section II-II-A, consider a set of model parameters, denoted as $\{\theta_k^{l,i} | l = 1, \ldots, L\}$, each representing the parameters of the l^{th} layer of the model in the i^{th} iteration at client k. In our approach, these parameters are initially transformed into 1D vectors, that is:

$$\Theta_k^{l,i} = \texttt{flatten}(\theta_k^{l,i}), \tag{5}$$

where $\texttt{flatten}(\cdot)$ denotes the flattening function and $\Theta_k^{l,i}$ is the resulting parameter vector. At client k an autoencoder, comprising an encoder ϕ_E^k and a decoder ϕ_D^k, is used to compress the model parameters per local iteration. Specifically, the encoder ϕ_E^k compresses the flattened model parameter $\Theta_k^{l,i}$ into a lower-dimensional representation,

$$\zeta_k^{l,i} = \phi_E^k(\Theta_k^{l,i}) = \phi_E^k(\texttt{flatten}(\theta_k^{l,i})). \tag{6}$$

Depending on the training stage (see Section III-III-C), the compressed representation is either sent to the server, where it is decoded and aggregated with the decoded parameters from

the other clients, or decoded locally at client k, leading to a perturbed version that is used in the subsequent local iteration. Concretely, in the latter case, the decoder at client k performs lossy reconstruction of the flattened model parameters:

$$\hat{\Theta}_k^{l,i} = \phi_D^k(\zeta_k^{l,i}) = \Theta_k^{l,i} + \epsilon_k^{l,i}, \tag{7}$$

where $\epsilon_k^{l,i}$ denotes the reconstruction error with respect to the input $\Theta_k^{l,i}$. The reconstructed parameters are then reshaped back and replace the original, uncompressed, parameters:

$$\hat{\theta}_k^{l,i} = \texttt{reshape}(\hat{\Theta}_k^{l,i}), \tag{8a}$$

$$\theta_k^{l,i} \leftarrow \hat{\theta}_k^{l,i}. \tag{8b}$$

The compression error $\epsilon_k^{l,i}$ by the autoencoder acts as a perturbation on the model parameters applied per local iteration, serving as a method to obscure the real update information and hinder an attacker's ability to reconstruct the training data. As we will show in Section IV, this compression error does not hinder training convergence and model performance. Moreover, as the clients exchange low-dimensional (compressed) representation of the model parameters the communication cost across the clients and the server is reduced.

B. AUTOENCODER ARCHITECTURE

We propose using an one-dimensional (1D) convolutional kernel for the construction of the autoencoder model, operating on flattened parameter tensors. Following this approach, enables compressing the parameters of various deep learning models irrespective of the size of the model. Furthermore, using an 1D convolutional kernel offers a reduction in the autoencoder parameters by approximately 60% compared to a 2D convolutional kernel [27], thereby mitigating the risk of overfitting and reducing encoding and decoding latency, complexity and memory footprint. In this work, the encoder of the autoencoder compresses the parameter vectors through a series of five 1D convolutional kernels, each followed by a leaky-ReLU [40] activation function (see Table I). The decoder (whose architecture is given in Table II) consists of five 1D deconvolutional kernels, each also followed by a leaky-ReLU activation function, and concludes with a 1D convolutional kernel of size one.

C. PROPOSED FEDERATED LEARNING PROCESS

Consider a decentralized environment comprising K clients, indexed by k, the objective is to train a deep learning model (with layers $l = 1, \ldots, L$) using FL, specifically FedAvg as detailed in Section II-II-A. In each communication round $t = 1, \ldots, T$, the server selects a subset C clients to participate the training. The client $k \in C$ trains the global model disseminated by the server across m local iterations.

In the first communication rounds, each client experiences rapid changes in the model parameters during its initial local iterations [41]. We use the terms *warming-up rounds* and *warming-up iterations* to refer to the initial communication rounds and the local iterations therein, respectively. Applying compression to the parameters updated during the warming-up iterations may detrimentally affect model performance. Therefore, in line with alternative decentralized training methods [27], [41], [42], we do not apply compression during the warming up iterations. Instead, during the warming-up iterations each client trains (in the first communication round that the client participates) or fine-tunes (in subsequent communication rounds that the client participates) its client-specific autoencoder to be used for parameter compression in the subsequent iterations. In what follows, we describe how we train and use the autoencoder in the different stages, namely, initialization, warming-up communication rounds, and regular communication rounds, of the FedAveg process.

TABLE I
ARCHITECTURE OF THE CONVOLUTIONAL LAYERS OF THE ENCODER.

Layer	Filters	Kernel Size	Stride
conv1	64	3	2
conv2	128	3	2
conv3	256	3	2
conv4	64	3	2
conv5	4	1	1

TABLE II
ARCHITECTURE OF THE CONVOLUTIONAL LAYERS OF THE DECODER.

Layer	Filters	Kernel Size	Stride
deconv1	4	3	2
deconv2	32	3	2
deconv3	64	3	2
deconv4	128	3	2
deconv5	32	3	1
conv	1	1	1

1) INITIALIZATION: Before initiating the FL process, the server initializes a global model, and each client k initializes an autoencoder per the architecture outlined in Section III-III-B. This initial version of the autoencoder is stored locally at the client.

2) WARMING-UP COMMUNICATION ROUNDS: During the warming-up communication rounds, $t = 1, \ldots, T_w - 1$, and for a number of warming-up iterations, $i = 1, \ldots, m_w - 1$, each participating client loads and fin-tunes the local autoencoder[1]. This fine-tuning process utilizes each layer of model parameters $\theta_k^{l,i}$ at every iteration and is described as follows:

$$\phi_E^k = \phi_E^k - \lambda \nabla_{\phi_E^k} L_{\text{rec}}(\phi_E^k, \phi_D^k, \theta_k^{l,i}), \tag{9a}$$

$$\phi_D^k = \phi_D^k - \lambda \nabla_{\phi_D^k} L_{\text{rec}}(\phi_E^k, \phi_D^k, \theta_k^{l,i}), \tag{9b}$$

where λ represents the learning step of the autoencoder training. The loss function for the autoencoder training is the mean squared error (MSE) between the actual and predicted parameters:

$$L_{rec}(\phi_E, \phi_D, \theta_k^{l,i}) = ||\hat{\Theta}_k^{l,i} - \Theta_k^{l,i}||_2^2. \tag{10}$$

As will be shown in Section IV, $m_w = 30$ warming-up iterations per client can lead to a well-trained autoencoder. After these iteration, the autoencoder is stored locally for usage in future rounds. Algorithm 1 depicts the proposed FL process during the warming-up iterations.

[1]We use T_w and m_w to denote the number of warming-up communication rounds and iterations, respectively.

Algorithm 1 Warming-up Iteration Training by the Client

Require: Pre-saved autoencoders ϕ_E^k, ϕ_D^k for each client k; received model parameters $\theta_k^{l,0}$ indexed by layer l from the server; learning step of the model η; model's loss function $\mathcal{L}(\cdot)$; learning step of autoencoder λ; loss function of autoencoder L_{rec}; the mini-batch $\mathcal{D}_k^i$ at iteration i, respectively.

for each client k **do**
  Load pre-saved autoencoders ϕ_E^k, ϕ_D^k
  for i in warming-up iterations **do**
    $\texttt{loss} = \mathcal{L}(f(\{\theta_k^{l,i-1} | l = 1, \ldots, L\}; \cdot); \mathcal{D}_k^{i-1})$
    for $l = 1$ to L **do**
      $\theta_k^{l,i} \leftarrow \theta_k^{l,i-1} - \eta \nabla_{\theta_k^{l,i-1}} \texttt{loss}$
      $\phi_E^k \leftarrow \phi_E^k - \lambda \nabla_{\phi_E^k} L_{\text{rec}}(\phi_E^k, \phi_D^k, \theta_k^{l,i})$
      $\phi_D^k \leftarrow \phi_D^k - \lambda \nabla_{\phi_D^k} L_{\text{rec}}(\phi_E^k, \phi_D^k, \theta_k^{l,i})$
    end for
  end for
  Store the fine-tuned autoencoder ϕ_E^k, ϕ_D^k locally
end for

Ensure: Fine-tuned autoencoders ϕ_E^k, ϕ_D^k

In the rest of the local iterations during the warming-up communication rounds, $i = m_w, \ldots, m$, the local model updates become stable and are effectively learned by the autoencoder. Thus, the autoencoder is capable of perturbing the model parameters without compromising model performance. After each training iteration, the original model parameter $\theta_k^{l,i}$ is replaced with the reconstructed parameter $\hat{\theta}_k^{l,i}$, obtained from the autoencoder as detailed in (5), (6), (7), and (8). Concretely, the client estimates the next iteration's model parameters using the reconstructed model parameters from the previous iteration $\hat{\theta}_k^{l,i}$, rather than the original parameters $\theta_k^{l,i}$, that is:

$$\hat{\theta}_k^{l,i} = \texttt{reshape}(\phi_D^k(\phi_E^k(\texttt{flatten}(\theta_k^{l,i})))); \quad (11a)$$

$$\theta_k^{l,i+1} = \hat{\theta}_k^{l,i} - \eta \nabla_{\hat{\theta}_k^{l,i}} \mathcal{L}(f(\{\hat{\theta}_k^{l,i} | l = 1, \ldots, L\}; \cdot), \mathcal{D}_k^i), \quad (11b)$$

where η and $\mathcal{D}_k^i$ denote the learning rate of the model and the mini-batch at iteration i, respectively. Algorithm 2 illustrates the training process during the regular iterations in the warming-up communication rounds.

At the end of each warming-up communication round, that is, after completing m local training iterations, the clients share the perturbed parameters $\hat{\theta}_k^{l,m}$ with the server. The server then updates the parameters per layer per (3) and shares the updated model with the clients. Note that during the warming-up communication rounds the model parameters change rapidly thereby sending the compressed representations (instead of the perturbed version) does not lead to good model performance.

3) REGULAR COMMUNICATION ROUNDS: At the conclusion of the warming-up rounds, specifically at $t = T_w - 1$, the clients share the trained decoders ϕ_D^k of the local autoencoder with the server. After participating in training for T_w rounds, the change in the model parameters at the clients becomes stable, and the autoencoders have been effectively trained. Therefore, in the remaining *regular communication rounds,* $t = T_w, \ldots, T$, the clients only need to transmit the compressed model parameters, allowing the server to reconstruct the model on the server side meanwhile reducing the communication costs.

Algorithm 2 Regular Iteration Training by the Client

Require: Pre-saved autoencoders ϕ_E^k, ϕ_D^k for each client k; model parameters $\theta_k^{l,0}$ indexed by layer l; learning step of the model η; learning step of autoencoder λ; loss function of the model $\mathcal{L}(\cdot)$; the mini-batch $\mathcal{D}_k^i$ at iteration i, respectively.

for each client k **do**
  Load pre-saved autoencoders ϕ_E^k, ϕ_D^k
  for i in regular iterations **do**
    for $l = 1$ to L **do**
      $\theta_k^{l,i} \leftarrow \texttt{reshape}(\phi_D^k(\phi_E^k(\texttt{flatten}(\theta_k^{l,i}))))$
    end for
    $\texttt{loss} = \mathcal{L}(f(\{\theta_k^{l,i} | l = 1, \ldots, L\}; \cdot), \mathcal{D}_k^{i-1})$
    for $l = 1$ to L **do**
      $\theta_k^{l,i} \leftarrow \theta_k^{l,i-1} - \eta \nabla_{\theta_k^{l,i-1}} \texttt{loss}$
    end for
  end for
end for

Ensure: Locally Trained model

Algorithm 3 Global Model Aggregation by the Server

Require: Pre-received decoders ϕ_D^k and received compressed model parameters $\zeta_k^{l,i}$ indexed by layer l from each client k; C is the total set of participants, where $k \in C$, the set of new model parameters θ_{new}, respectively.

for each layer l **do**
  $\theta^l = 0$
  for each client k **do**
    $\theta^l = \theta^l + \texttt{reshape}(\phi_D^k(\zeta_k^{l,i}))$
  end for
  $\theta^l = \theta^l / |C|$
  $\theta_{\texttt{new}}$.append(θ^l)
end for

Ensure: a set of new global model parameters $\theta_{\texttt{new}}$ indexed by layer's id

Specifically, during the local iterations $i = 1, \ldots, m$ of a regular communication round, the original model parameters $\theta_k^{l,i}$ are replaced and by the locally reconstructed (perturbed) parameters $\hat{\theta}_k^{l,i}$, aligning with the process described in Section III-C III-C2 (see Algorithm 2). After the completion of m local training iterations, the clients share their compressed model parameters $\zeta_k^{l,m}$ with the server. Once the server receives the compressed model parameters $\zeta_k^{l,m}$, for the participating clients $k = 1, \ldots, K$, it reconstructs them utilizing the corresponding decoders ϕ_D^k [see (10) and (8)]. The server then updates the global model parameters of layer l to:

$$\frac{1}{|C|} \sum_{k \in C} \texttt{reshape}(\phi_D^k(\zeta_k^{l,i})) \quad (12)$$

Algorithm 3 elaborates on the process of reconstructing and aggregating the model parameters by the server.

IV. EXPERIMENTS

In this section, we evaluate our approach in terms of model performance, information leakage, and communication rate.

A. EXPERIMENTAL SETUP

The experiments were conducted on a single machine equipped with two GeForce RTX 3090 Ti GPUs and 128 GB of RAM. Random seeds were set to 1234 for reproducibility. We follow the experimental setup in [9], [15], [25], [29], [43]. Specifically, We evaluate our framework regarding model performance and privacy with two classic architectures, LeNet [29] and Multilayer Perceptron (MLP) [44], across 100 clients through $T = 500$ communication rounds. The LeNet architecture is widely utilized for evaluating the information leakage of convolutional neural network (CNN) models. The MLP model consists of three layers, with a hidden dimension of 256, widely used in assessing model convergence. Since the input to any fully connected layer can be recovered analytically [10], we use the MLP model to show the upper bound of privacy leakage. We follow the work of [25] and compare our method with three established methods, noisy gradient (NG) [16], gradient sparsification (GS) [9], and PRECODE [25]. We did not include Soteria [15] in the comparisons, as it induces high computational complexity when searching for the perturbation position of gradients. We also excluded the defenses proposed in [33], [43], [45] because they modify the training protocol. Regarding NG, we incorporate two levels of Gaussian noise into the gradients, generated by considering zero means and standard deviations of 10^{-1} and 10^{-2}. These levels are respectively denoted as NG-10^{-1} and NG-10^{-2}. Regarding GS, we retain the top 10% and 30% of significant gradients, while assigning zero to the remaining gradients; we refer to these configurations as GS-90% and GS-70%, respectively. Concerning PRECODE, we adopt the same configurations as described in [25] and append a variational bottleneck after the feature extractor. Concerning our methodology, as detailed in Section III, we set the warming-up rounds T_w to 35 for each client, and the fine-tuning process for the client's autoencoder entails using updates twice over a span of 30 iterations ($m_w = 30$).

We conduct experiments on the FashionMNIST [46], CIFAR-10 [47], and CIFAR-100 [47] datasets. Fashion-MNIST contains a collection of 70,000 grayscale clothing images, divided into a training set of 60,000 samples and a test set of 10,000 samples. Each image has a resolution of 28 $\times$ 28 pixels and belongs to one of 10 classes. CIFAR-10 consists of a collection of 60,000 color images, divided into a training set with 50,000 images and a test set with 10,000 images. Each image has a resolution of 32$\times$32 pixels and is labeled with one out of ten different classes. CIFAR-100 has 60,000 color images, with a resolution of 32$\times$32 pixels, divided into a training set with 50,000 images and a test set with 10,000 images. CIFAR-100 has 100 classes with 20 superclasses. We evaluate the model performance on both IID and non-IID data partition settings, where the partitions are the same as in [1].

To evaluate privacy leakage, we attempt reconstructing the training data by using two well-established gradient inversion attacks, DLG [9] and IGA [10]. Our defense evaluation primarily focuses on establishing a lower bound for assessing information leakage. We set a small batch size to make the reconstruction process easier and more susceptible to potential attacks. By narrowing our focus to small batch sizes, we can effectively evaluate the vulnerabilities and effectiveness of our defense mechanisms in scenarios where the risk of information leakage is higher. We have not considered attacks such as GradInversion [11], which requires the additional statistics of data that contrary to the basic concept of FL. We conducted random sampling to obtain a total of 128 images from the training set and attacked them using the publicly available PyTorch implementations of DLG and IGA taken from [10]. The attack configuration aligns with the settings described in prior work [9], [10], [15], [25], [29]. Specifically, randomly initializing the dummy input using samples from a Gaussian distribution, utilizing the Adam optimizer [48] with a learning rate of 0.01. DLG uses the Euclidean distance to constrain the true gradients and dummy gradients, while IGA utilizes the cosine similarity distance. The total variation regularization of IGA is set to 10^{-6}. The dummy data was optimized for 7,000 iterations with three times restarts. We assumed that the attacker knows the label, which can be inferred from the gradients in the final layer [29]. To quantitatively assess the quality of image reconstructions, similar to [9], [10], [25], [30], we calculated the mean squared error (MSE), peak signal-to-noise ratio (PSNR), structural similarity (SSIM) [49], and attack success rate (ASR) between the original and reconstructed images. The dummy images were defined as successful reconstructions when the SSIM value was equal to or exceeded the value of 0.6 [25], [30]. A high MSE, low PSNR, low SSIM, and low ASR of the reconstructed images indicate effective privacy protection.

B. MODEL PERFORMANCE EVALUATION

Tables III, IV, and V report the top-1 accuracy achieved by the MLP and LeNet with different defense methods on the Fashion-MNIST, CIFAR-10, and CIFAR-100 dataset, respectively. Bold numbers denote the best performance and underlined numbers indicate the second-best performance. B and E refer to the batch size and local number of epochs. To demonstrate the impact of the different defenses on model performance, we utilize two different batch sizes: a small batch size of 8 and a large batch size of 64. Our framework shows competitive performance across various models and partitions. The accuracy achieved by our approach is nearly close to the baseline (FedAvg). Consistent with the observations reported in [16], noisy gradient with a high noise level ($\sigma = 10^{-1}$) leads to a considerable decrease in accuracy; for example, in the case of the MLP model trained on the non-IID partition of Fashion-MNIST with a batch size of 64 and 8, the accuracy drops by 18.73% and 15.39%, respectively. In contrast, our proposed method exhibits only a marginal decrease of 1.14% and 0.51% in accuracy. When applying high sparsification (90%), we observe a significant degradation in the performance of LeNet on CIFAR-10 and CIFAR-100. Specifically, when using a batch size of 64, the accuracy of LeNet drops by 6.27% and 6.34% on CIFAR-10 for the IID and non-IID partitions,

TABLE III
TOP-1 ACCURACY IN FEDERATED LEARNING OF THE MLP AND LENET ON BOTH THE IID AND NON-IID FASHION-MNIST DATASET.

	$B=64, E=10$				$B=8, E=1$			
	LeNet		MLP		LeNet		MLP	
	IID	non-IID	IID	non-IID	IID	non-IID	IID	non-IID
FedAvg	89.02	79.78	89.89	85.06	87.85	80.53	89.70	85.13
DP (10^{-1})	86.68 (-2.34)	76.20 (-3.58)	84.79 (-5.10)	66.33 (-18.73)	86.97 (-0.88)	79.18 (-1.35)	84.70 (-5.0)	69.74 (-15.39)
NG (10^{-2})	89.00 (-0.02)	**79.67 (-0.11)**	89.53 (-0.36)	85.12 (+0.06)	**88.14 (+0.29)**	**80.62 (+0.09)**	89.43 (-0.27)	**85.29 (+0.16)**
GS (90%)	87.28 (-1.74)	76.85 (-2.93)	89.29 (-0.60)	84.43 (-0.63)	86.28 (-1.57)	77.45 (-3.08)	88.93 (-0.77)	84.33 (-0.80)
GS (70%)	88.41 (-0.61)	79.44 (-0.34)	**89.85 (-0.04)**	**85.15 (+0.09)**	87.65 (-0.20)	80.02 (-0.51)	**89.44 (-0.26)**	84.89 (-0.24)
PRECODE	**89.17 (+0.15)**	73.24 (-6.54)	89.33 (-0.56)	82.25 (-2.81)	85.17 (-2.68)	10.91 (-69.62)	88.84 (-0.86)	80.27 (-4.86)
Our work	88.24 (-0.78)	77.64 (-2.14)	89.76 (-0.13)	83.92 (-1.14)	87.05 (-0.80)	77.83 (-2.70)	89.22 (-0.48)	84.62 (-0.51)

TABLE IV
TOP-1 ACCURACY IN FEDERATED LEARNING OF THE MLP AND LENET ON BOTH THE IID AND NON-IID CIFAR-10 DATASET.

	$B=64, E=10$			$B=8, E=1$		
	LeNet		MLP	LeNet		MLP
	IID	non-IID	IID	IID	non-IID	IID
FedAvg	60.75	46.73	52.88	56.99	46.18	50.59
NG (10^{-1})	54.48 (-6.27)	37.27 (-9.46)	38.73 (-14.15)	51.09 (-5.90)	41.22 (-4.96)	24.62 (-25.59)
NG (10^{-2})	**60.66 (-0.09)**	**48.20 (+1.47)**	52.06 (-0.82)	**56.21 (-0.78)**	**47.16 (+0.98)**	49.87 (-0.72)
GS (90%)	54.48 (-6.27)	40.39 (-6.34)	52.41 (-0.47)	50.14 (-6.85)	40.85 (-5.33)	47.81 (-2.78)
GS (70%)	59.94 (-0.84)	47.62 (+0.89)	**53.07 (+0.19)**	54.00 (-2.99)	45.97 (-0.21)	**49.96 (-0.63)**
PRECODE	56.07 (-4.68)	39.26 (-7.47)	50.22 (-2.66)	52.16 (-4.83)	10.87 (-35.31)	48.52 (-2.07)
Our work	60.43 (-0.32)	45.23 (-1.50)	52.87 (-0.01)	54.82 (-2.17)	41.65 (-4.53)	46.77 (-3.82)

TABLE V
TOP-1 ACCURACY IN FEDERATED LEARNING OF LENET ON IID CIFAR-100.

	LeNet	
	$B=64, E=10$	$B=8, E=1$
FedAvg	27.63	24.71
NG (10^{-1})	22.57 (-0.51)	17.41 (-7.30)
NG (10^{-2})	**27.87 (+0.24)**	25.27 (+0.56)
GS (90%)	22.44 (-5.19)	20.36 (-4.35)
GS (70%)	23.58 (-4.05)	24.97 (+0.26)
PRECODE	25.25 (-2.38)	**25.28 (+0.57)**
Our work	27.02 (-0.61)	23.69 (-1.02)

TABLE VI
COMPRESSION RATIO (CR) OF VARIOUS DEFENSES

	FedAvg	NG (10^{-1})	NG (10^{-2})	GS (90%)	GS (70%)	Our work
CR	1.00×	1.00×	1.00×	3.33×	1.11×	**4.10** ×

TABLE VII
COMPRESSION RATIO (CR) OF VARIOUS MODELS ON DIFFERENT DATASETS USING PRECODE

	LeNet-FashionMNIST	LeNet-CIFAR10	LeNet-CIFAR100	MLP-FashionMNIST	MLP-CIFAR10
CR	0.0288×	0.0260×	0.1258×	0.5771×	0.8125×

respectively. In contrast, our framework experiences only a minimal loss of 0.32% and 1.50% in accuracy for the same setting. The results reveal that applying gradient sparsification of (70%) and noisy gradient with a low noise level ($\sigma = 10^{-2}$) leads to the best model training performance most of the times. However, as we will see in Section IV-IV-C and Section IV-IV-D, these configurations do not effectively prevent the reconstruction of training data from local updates. Furthermore, PRECODE exhibits limited accuracy robustness, particularly when dealing with non-IID data and smaller batch sizes. When training LeNet on the Fashion-MNIST and CIFAR-10 datasets with a batch size of 8, PRECODE achieves accuracy levels of only 10.91% and 10.87%, respectively, underlying its inability to maintain satisfactory performance under these conditions.

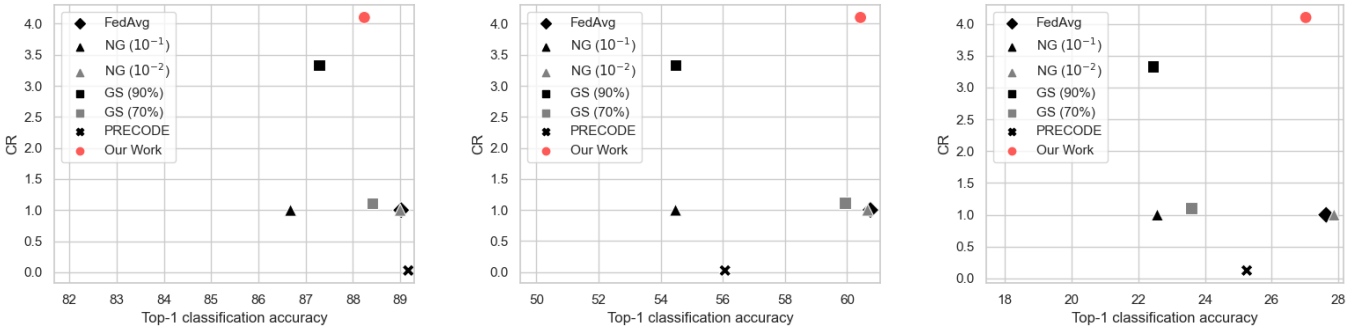
Table VI and VII report the compression ratio (CR) of all defenses as an ancillary evaluation. CR is defined as

$$\text{CR} = \texttt{size}(\theta^{\text{original}})/\texttt{size}(\theta^{\text{compressed}}), \tag{13}$$

where θ^{original} and $\theta^{\text{compressed}}$ are the uncompressed model and the compressed model by each defense approaches, respectively; the $\texttt{size}(\cdot)$ function computes the size of the parameters. The number of parameters of the decoder ϕ_D of our autoencoder is 72,352, resulting to a size of $289,408$ bytes, which is only transmitted once. The compression ratio results in Table VI and VII indicate that our framework can achieve a $4.1\times$ compression ratio compared to the FedAvg method while at the same time maintaining privacy (as we will discuss in Section IV-IV-C and Section IV-IV-D). On the contrary, PRECODE incurs a communication rate that is 34.8 times greater than FedAvg when employed to LeNet on the Fashion-MNIST dataset. Fig. 3 depicts the top-1 accuracy of LeNet trained with a batch size of 64 plotted against the CR for various datasets (IID splits). Our work achieves the best trade-off between communication rate reduction and model accuracy compared to all other methods.

C. INFORMATION LEAKAGE FROM UNTRAINED MODELS

When machine learning models are trained from scratch in FL, the magnitude of the gradient is generally large, making it easier to infer the training data [10]. Therefore, following the settings in [9], [10], [15], [25], we first evaluate the privacy-preserving capability of our approach for untrained, randomly initialized models. For a given test data sample, we perform a



(a) Fashion-MNIST (b) CIFAR-10 (c) CIFAR-100

Fig. 3. The top-1 accuracy of LeNet, trained with a batch size of 64, is plotted against the compression ratio (CR). The CR is calculated as per (13). The results are presented for (a) Fashion-MNIST, (b) CIFAR-10, and (c) CIFAR-100, where the data is split IID in the participating clients.

TABLE VIII
EVALUATION OF THE DEFENSE EFFECTIVENESS OF UNTRAINED MODELS ON THE FASHION-MNIST DATASET.

Defense	Untrained model on the Fashion-MNIST dataset							
	LeNet (DLG/IGA)				MLP (DLG/IGA)			
	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓
FedAvg	0.00/0.00	60.46/36.08	0.99/0.95	100/100	0.00/0.00	102.65/83.82	1.00/1.00	100/100
NG (10^{-1})	18.14/0.46	-6.11/9.67	0.05/0.27	0/1	0.34/0.29	10.79/11.38	0.32/0.29	0/0
NG (10^{-2})	0.03/0.02	22.51/24.66	0.81/0.84	98/100	0.00/0.01	30.55/28.76	0.92/0.92	100/100
GS (90%)	13.12/0.68	-4.76/7.68	0.03/0.12	0/0	0.08/0.05	17.23/19.42	0.60/0.64	49/63
GS (70%)	1.74/0.23	4.29/12.53	0.26/0.43	0/1	0.00/0.00	33.55/35.58	0.97/0.97	100/100
PRECODE	2.18/**1.01**	2.66/**6.00**	**0.01/0.04**	0/0	**0.91/1.47**	**6.53/4.36**	**0.06/0.00**	0/0
Our work	**32.67**/0.74	**-7.91**/7.34	0.027/0.072	0/0	0.48/0.40	9.45/9.97	**0.06**/0.03	0/0

TABLE IX
EVALUATION OF THE DEFENSE EFFECTIVENESS OF UNTRAINED MODELS ON THE CIFAR-10 DATASET.

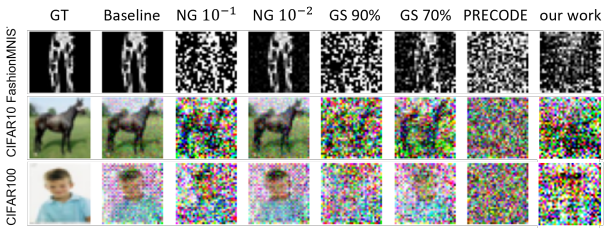
Defense	Untrained model on the CIFAR-10 dataset							
	LeNet (DLG/IGA)				MLP (DLG/IGA)			
	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓
FedAvg	0.38/0.29	18.95/19.01	0.62/0.60	58/56	0.00/0.01	52.89/49.34	1.00/0.99	100/100
NG (10^{-1})	4.98/1.47	5.31/10.63	0.10/0.18	0/0	0.48/0.48	15.30/15.32	0.34/0.33	0.02/0
NG (10^{-2})	0.73/0.50	17.13/17.35	0.56/0.53	54/48	0.01/0.01	33.86/33.67	0.96/0.95	100/100
GS (90%)	5.27/**1.93**	5.03/**9.22**	0.04/0.08	0/0	0.12/0.07	21.97/24.22	0.72/0.76	91/93
GS (70%)	2.90/1.05	7.63/11.92	0.13/0.21	0/0	0.00/0.00	37.07/37.43	0.98/0.98	100/100
PRECODE	2.34/1.90	8.43/9.31	**0.01**/0.04	0/0	**0.66/3.10**	**14.22/7.14**	0.19/**0.05**	0/0
Our work	**6.50**/1.71	**4.05**/9.86	0.02/**0.03**	0/0	0.60/0.66	14.89/14.34	**0.14**/0.10	0/0

TABLE X
EVALUATION OF THE DEFENSE EFFECTIVENESS OF UNTRAINED MODELS ON THE CIFAR-100 DATASET.

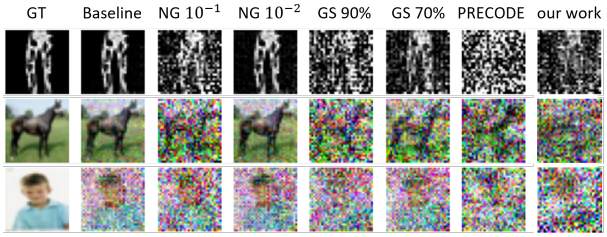
Untrained model	Untrained LeNet on CIFAR-100							
	DLG				IGA			
	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓
FedAvg	0.4684	15.156	0.399	3	0.5639	14.2303	0.3044	1
NG (10^{-1})	3.7914	5.7627	0.0702	0	1.271	10.5457	0.1178	0
NG (10^{-2})	0.5094	14.7163	0.381	0	0.6138	13.8747	0.2783	0
GS (90%)	3.0733	6.7929	0.0432	0	1.3862	10.2143	0.0624	0
GS (70%)	1.739	9.2713	0.1294	0	0.9316	11.9544	0.154	0
PRECODE	2.1969	8.203	0.0066	0	**1.6899**	**9.3411**	**0.0331**	0
Our work	**5.2635**	**4.3801**	0.0153	0	1.3550	10.3967	0.0442	0

single training iteration of a randomly initialized model (LeNet or MLP) using this data sample and obtain the updated model parameters. Then, we use the updated model parameters to train the autoencoder using 100 SGD iterations. Subsequently, we employ the trained autoencoder to compress the model parameters and then attempt to reconstruct the training data using the decompressed model parameters. It is clear that the autoencoder is overfitting in this case, resulting in a lower bound for privacy preservation.

Tables VIII, IX, and X report information leakage results when attacking LeNet and MLP using DLG and IGA on the Fashion-MNIST, CIFAR-10, and CIFAR-100 datasets, respectively. The first number represents the metric achieved using DLG, and the second number corresponds to the metric obtained using IGA. The bold numbers denote the best performance and the underlined numbers indicate the second-best performance. The results show that without privacy protection (FedAvg), both attacks can effectively reconstruct the data. When using high sparsification (90%) with gradient sparsification or high noise level within noisy gradient (NG-10^{-1}), the data can be effectively protected. However, as discussed in Section IV-IV-B, these methods significantly impact the model's performance. Conversely, the data can be attacked when using low noise level (NG-10^{-2}) with noisy gradient. In the case of MLP, neither high sparsification (90%) in gradient sparsification nor noisy gradient with a significant noise level (NG-10^{-1}) effectively prevent data leakage. PRECODE is acknowledged as one of the most



(a) Attack on untrained LeNet by DLG



(b) Attack on untrained LeNet by IGA

Fig. 4. Reconstruction results on the Fashion MNIST, CIFAR10, and CIFAR100 datasets for the initialized LeNet model and different defense mechanisms.

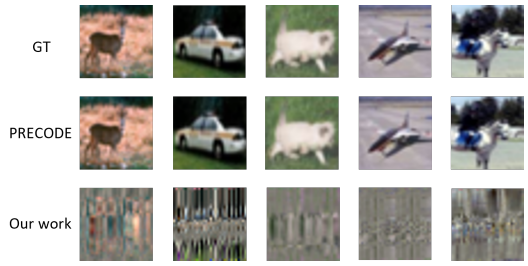


Fig. 5. Reconstruction results on the CIFAR-10 by using Bayesian Framework for Gradient Leakage [24].

effective defense approaches against information leakage. Nevertheless, as discussed in Section IV-IV-B, PRECODE induces modifications to the model architecture and leads to non-convergence for non-IID data when employed with small batch sizes, particularly when training CNN models. Moreover, as elaborated in Section II, in the event that the malicious server manages to access the model's architecture and the model is fully connected, it can lead to an impeccable reconstruction of the training data by circumventing PRECODE using attack proposed in [24]. The training data can be analytically reversed by multiplying the gradient of weights and biases in the first layers.

Fig. 4 depicts visual examples of reconstructed data samples. noisy gradient and gradient sparsification techniques are ineffective in concealing information from the gradients. However, our framework successfully removes nearly all useful information, resulting in the attacker being able to reconstruct only a highly blurred dummy image. Fig. 5 depicts the reconstruction results using the attack in [24], which successfully circumvents the PRECODE defense mechanism in the MLP model. Contrarily, our work demonstrates effective countermeasures against such forms of attacks.

D. INFORMATION LEAKAGE FROM TRAINED MODELS

We also investigate the privacy leakage prevention of the various defenses in the middle of the training stage, which is more close to the real FL environment. The models are attacked after five communication rounds of training. We randomly select a client and randomly obtain 128 local training data samples. These data samples are then grouped into batches of size 8, and used to obtain the updated model parameters. The model parameters are compressed and decompressed by the autoencoder and used to reconstruct the data batch using the corresponding attack.

TABLE XI
EVALUATION OF THE DEFENSE EFFECTIVENESS ON TRAINED MLP MODELS FOR THE FASHION-MNIST AND CIFAR DATASETS BY USING IGA

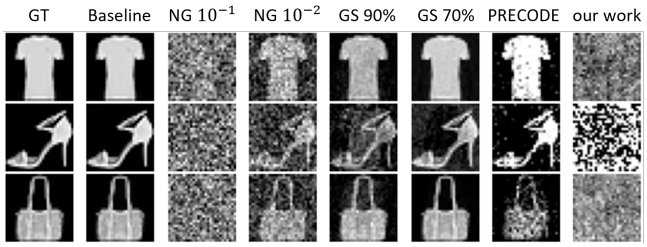
Defense	Fashion-MNIST				CIFAR			
	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓	MSE ↑	PSNR ↓	SSIM ↓	ASR ↓
FedAvg	0.00	35.97	0.97	100	1.97	14.58	0.37	28.75
NG (10^{-1})	1.69	3.94	0.03	**0**	**4.10**	**6.28**	0.03	**0**
NG (10^{-2})	0.33	10.90	0.29	0	3.39	10.18	0.22	6.35
GS (90%)	0.66	14.36	0.45	42.5	2.05	11.28	0.23	5
GS (70%)	0.91	9.28	0.29	18.75	2.16	13.70	0.37	27.5
PRECODE	1.58	4.52	0.04	1.25	1.84	10.40	0.05	2.5
Our work	**1.69**	**3.77**	**0.00**	**0**	1.52	10.82	**0.00**	**0**

Table XI reports the results of information leakage when attacking the locally trained MLP model using IGA on the Fashion-MNIST and CIFAR-10, respectively. The results obtained with noisy gradient and gradient sparsification are consistent with those reported in Section IV-IV-C, while our approach is the most effective in protecting information. Fig. 6 depicts visual examples of data reconstructed from attacking the MLP model. It is evident that PRECODE does not effectively protect against information leakage. This could be due to that after training, the variational bottleneck has learned a much smaller sampling space, thereby limiting its ability to provide effective perturbations on the representation. In contrast, after undergoing actual local training, our method does not exhibit information leakage.

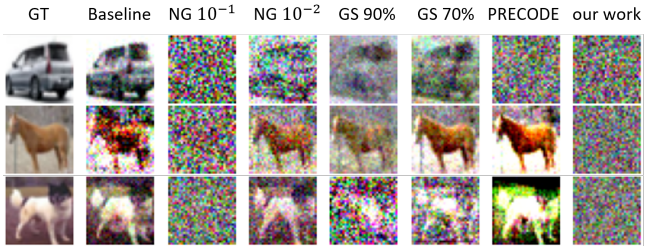
Fig. 7 illustrates the top-1 accuracy of MLP, which is trained using a batch size of 64, plotted against the Structural Similarity Index Measure (SSIM) of the reconstructed training data by using IGA for various IID datasets. Notably, our work secures the highest level of privacy protection while simultaneously preserving the model's performance.

V. CONCLUSION

We proposed an autoencoder-based method to safeguard neural network models against the risk of information leakage during the exchange of model updates in federated learning. The autoencoder's lossy compression mechanism applies a perturbation to the updates. We assessed our method through experiments on two classical neural network architectures and three datasets, considering scenarios with both and non-iid data partitions. The evaluation encompassed two well-known gradient inversion attacks. When contrasted with other defense strategies, including differential privacy, data compression, and PRECODE, our approach stands out due to its ability to achieve a good privacy-performance trade-off, avoid prolonged convergence associated with these methods, and evade perturbed

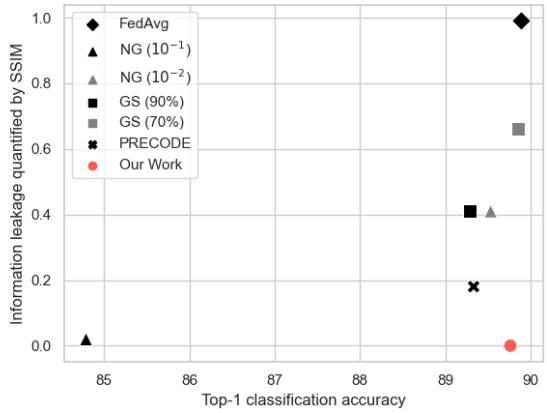


(a) Fashion-MNIST data reconstructed by IGA

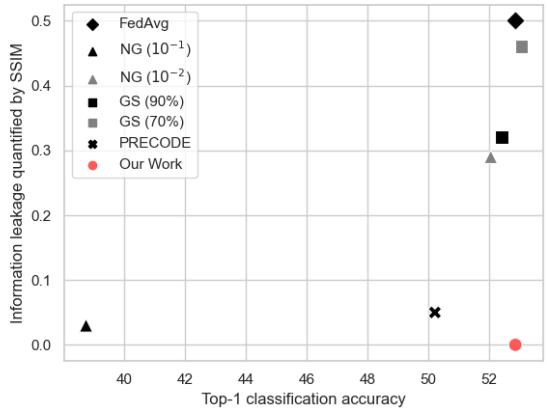


(b) CIFAR-10 data reconstructed by IGA

Fig. 6. Reconstructed training data including, (a) Fashion-MNIST and (b) CIFAR-10, from the gradient of trained MLP



(a) Fashion-MNIST



(b) CIFAR-10

Fig. 7. The top-1 accuracy of MLP, trained with a batch size of 64, is plotted versus the Structural Similarity Index Measure (SSIM) of the reconstructed training data across various datasets (IID splits), which includes (a) Fashion-MNIST and (b) CIFAR-10 datasets.

layers as opposed to PRECODE. Our method achieves accuracy levels comparable to those obtained during unprotected training and less communication cost with a compression ratio of at least four times less rate compared to federated averaging, while simultaneously preventing information leakage.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics*, 2016.

[2] A. AbhishekV, S. Binny, R. JohanT, N. Raj, and V. Thomas, "Federated learning: Collaborative machine learning without centralized training data," *international journal of engineering technology and management sciences*, 2022.

[3] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *Journal of Healthcare Informatics Research*, vol. 5, pp. 1–19, 2021.

[4] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[5] X. Liu, J. Yu, Y. Liu, Y. Gao, T. Mahmoodi, S. Lambotharan, and D. H.-K. Tsang, "Distributed intelligence in wireless networks," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 1001–1039, 2023.

[6] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning: Revisited and enhanced," in *International Conference on Applications and Techniques in Information Security*, 2017.

[7] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[8] A. Sablayrolles, M. Douze, C. Schmid, Y. Ollivier, and H. Jégou, "White-box vs black-box: Bayes optimal strategies for membership inference," in *International Conference on Machine Learning*, 2019.

[9] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Neural Information Processing Systems*, 2019.

[10] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients - how easy is it to break privacy in federated learning?" *ArXiv*, vol. abs/2003.14053, 2020.

[11] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See through gradients: Image batch recovery via gradinversion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 337–16 346.

[12] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, pp. 143–202, 2000.

[13] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively secure multi-party computation," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 639–648.

[14] J. Zhao, H. Zhu, F. Wang, R. Lu, Z. Liu, and H. Li, "Pvd-fl: A privacy-preserving and verifiable decentralized federated learning framework," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2059–2073, 2022.

[15] J. Sun, A. Li, B. Wang, H. Yang, H. Li, and Y. Chen, "Soteria: Provable defense against privacy leakage in federated learning from representation perspective," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9307–9315, 2020.

[16] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *International Conference on Learning Representations*, 2017.

[17] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *ArXiv*, vol. abs/1712.07557, 2017.

[18] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farhad, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:207847853

[19] X. Yuan, W. Ni, M. Ding, K. Wei, J. Li, and H. V. Poor, "Amplitude-varying perturbation for balancing privacy and utility

in federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1884–1897, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257405067

[20] Z. Liang, P. Yang, C. Zhang, and X. Lyu, "Secure and efficient hierarchical decentralized learning for internet of vehicles," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 1417–1429, 2023.

[21] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Conference on Empirical Methods in Natural Language Processing*, 2017.

[22] Y. Tsuzuku, H. Imachi, and T. Akiba, "Variance-based gradient compression for efficient distributed deep learning," *ArXiv*, vol. abs/1802.06058, 2018.

[23] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *ArXiv*, vol. abs/1610.05492, 2016.

[24] M. Balunovi'c, D. I. Dimitrov, R. Staab, and M. T. Vechev, "Bayesian framework for gradient leakage," *ArXiv*, vol. abs/2111.04706, 2021.

[25] D. Scheliga, P. Mäder, and M. Seeland, "Precode - a generic model extension to prevent deep gradient leakage," *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 3605–3614, 2021.

[26] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:216078090

[27] L. Abrahamyan, Y. Chen, G. Bekoulis, and N. Deligiannis, "Learned gradient compression for distributed deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7330–7344, 2021.

[28] ——, "Autoencoder-based gradient compression for distributed training," in *2021 29th European Signal Processing Conference (EUSIPCO)*. IEEE, 2021, pp. 2179–2183.

[29] B. Zhao, K. R. Mopuri, and H. Bilen, "idlg: Improved deep leakage from gradients," *arXiv preprint arXiv:2001.02610*, 2020.

[30] W. Wei, L. Liu, M. Loper, K.-H. Chow, M. E. Gursoy, S. Truex, and Y. Wu, "A framework for evaluating client privacy leakages in federated learning," in *European Symposium on Research in Computer Security*. Springer, 2020.

[31] A. Hatamizadeh, H. Yin, H. R. Roth, W. Li, J. Kautz, D. Xu, and P. Molchanov, "Gradvit: Gradient inversion of vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 021–10 030.

[32] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "A transprecision floating-point platform for ultra-low power computing," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1051–1056, 2017.

[33] W. Gao, X. Zhang, S. Guo, T. Zhang, T. Xiang, H. Qiu, Y. Wen, and Y. Liu, "Automatic transformation search against deep leakage from gradients," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–18, 2023.

[34] A. A. Alemi, I. S. Fischer, J. V. Dillon, and K. P. Murphy, "Deep variational information bottleneck," *ArXiv*, vol. abs/1612.00410, 2016.

[35] Y. Miao, W. Zheng, X. Li, H. Li, K. R. Choo, and R. H. Deng, "Secure model-contrastive federated learning with improved compressive sensing," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3430–3444, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259062442

[36] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2021–2031.

[37] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. E. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," *ArXiv*, vol. abs/2007.07682, 2020.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[40] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *ArXiv*, vol. abs/1505.00853, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:14083350

[41] Y. Lin, S. Han, H. Mao, Y. Wang, and W. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/pdf?id=SkhQHMW0W

[42] J. Ma, T. Zhou, G. Long, J. Jiang, and C. Zhang, "Structured federated learning through clustered additive modeling," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=2XT3UpOv48

[43] W. Wei, L. Liu, Y. Wu, G. Su, and A. Iyengar, "Gradient-leakage resilient federated learning," *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pp. 797–807, 2021.

[44] Q. Li, B. He, and D. X. Song, "Model-contrastive federated learning," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10 708–10 717, 2021.

[45] H. Lee, J. Kim, S. Ahn, R. Hussain, S. Cho, and J. Son, "Digestive neural networks: A novel defense strategy against inference attacks in federated learning," *Comput. Secur.*, vol. 109, p. 102378, 2021.

[46] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[47] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," in *Advances in Neural Information Processing Systems*, 2009.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[49] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.