Model Inversion Attacks Against Collaborative Inference

Zecheng He
zechengh@princeton.edu
Princeton University

Tianwei Zhang
tianwei.zhang@ntu.edu.sg
Nanyang Technological University

Ruby B. Lee
rblee@princeton.edu
Princeton University

ABSTRACT

The prevalence of deep learning has drawn attention to the privacy protection of sensitive data. Various privacy threats have been presented, where an adversary can steal model owners' private data. Meanwhile, countermeasures have also been introduced to achieve privacy-preserving deep learning. However, most studies only focused on data privacy during training, and ignored privacy during inference.

In this paper, we devise a new set of attacks to compromise the *inference* data privacy in *collaborative* deep learning systems. Specifically, when a deep neural network and the corresponding inference task are split and distributed to different participants, one malicious participant can accurately recover an arbitrary input fed into this system, even if he has no access to other participants' data or computations, or to prediction APIs to query this system. We evaluate our attacks under different settings, models and datasets, to show their effectiveness and generalization. We also study the characteristics of deep learning models that make them susceptible to such inference privacy threats. This provides insights and guidelines to develop more privacy-preserving collaborative systems and algorithms.

CCS CONCEPTS

• Security and privacy → Systems security; **Distributed systems security**; **• Computing methodologies → Artificial intelligence**.

KEYWORDS

Deep Neural Network, Model Inversion Attack, Distributed Computation

ACM Reference Format:
Zecheng He, Tianwei Zhang, and Ruby B. Lee. 2019. Model Inversion Attacks Against Collaborative Inference. In *2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3359789.3359824

1 INTRODUCTION

Deep learning technology has developed rapidly, especially Deep Neural Networks (DNNs). Deep learning models outperform traditional machine learning approaches on various artificial intelligence tasks, e.g., image recognition [19], natural language processing [31], speech recognition [16], anomaly detection [20] etc. Such high and reliable performance is attributed to the models' complex structures (i.e., a large number of hidden layers and parameters), time and resource consuming training process, and a significant amount of data.

To accelerate the learning and prediction processes, as well as reduce overheads, collaborative deep learning systems have been designed. Typically there are two collaborative modes. The first is *collaborative training* [6, 7]. The training task is distributed to multiple participants. Each participant trains an individual model over his private dataset, and periodically exchanges model updates. The final model is aggregated from each individual model. Collaborative training can significantly improve the training speed.

The second mode is *collaborative inference* [8, 17, 25, 27, 43]. The basic idea is to split a deep neural network into multiple parts, with each part allocated to a different participant. An input sequentially goes through each part of the neural network on these participants to generate the final output. Collaborative inference has gained popularity in the edge-cloud scenario. As edge devices, e.g. IoT devices and smartphones, have limited computation and storage capacities, it is difficult for a single device to host an entire model, and conduct the inference within reasonable latencies. Instead, the neural network can be divided into two parts. The first few layers of the network are stored in the local edge device, while the rest are offloaded to a remote cloud server. Given an input, the edge device calculates the output

Code available: https://github.com/zechenghe/Inverse_Collaborative_Inference

ACSAC '19, December 9–13, 2019, San Juan, PR, USA

ACM ISBN 978-1-4503-7628-0/19/12…$15.00
https://doi.org/10.1145/3359789.3359824

of the first layers, sends it to the cloud, and retrieves the final results. Collaboration between the edge device and cloud server achieves higher inference speed and lower power consumption than running the task solely on the local or remote platform.

Privacy has become a big security concern in deep learning. Past work presented a variety of privacy threats against training data, e.g., *property inference attacks* [3, 11], *membership inference attacks* [18, 29, 30, 39, 41, 49], *model inversion attacks* [9, 10]. These threats are also important in the context of collaborative training. Since multiple participants are involved in one task, but not all of them are necessarily trusted, it is essential to prevent malicious participants from stealing sensitive data. It has been shown that the privacy of training data is better protected if each participant in the distributed system uses his own dataset and never shares it with other participants [15, 40]. However, it is still possible for an adversary participant to infer the sensitive information and properties of other participants' training data indirectly via model updates. Distributed model inversion attacks [22] and property inference attacks [32] were designed and implemented.

In contrast, inference data privacy is less studied, either in single-party or multi-party machine learning systems. It is much more challenging to recover inference samples than training samples due to the following reasons: (1) the model parameters do not depend on the inference input. Thus the inference process reveals less useful information about inference samples to the adversary; (2) Training samples usually follow certain distributions, enabling the adversary to recover statistical information about such distributions. Inference samples do not have this assumption, making it hard to recover an individual sample. In single-party systems, Wei et al. [48] made an attempt towards inference data recovery in DNN accelerators via power side channels. Their attacks required the adversary to be able to hack into the victim device and install trojans, and the input image needs to be very simple, e.g. binary. These assumptions make the attack less practical. To the best of our knowledge, there is no work exploring the privacy issues in multi-party inference systems.

This paper presents the *first* investigation of inference data privacy in a collaborative ML system. Two key questions are considered in this study. The first one is: *if one intermediate participant is compromised and controlled by an adversary, can he recover an arbitrary input sample?* To answer this question, we design a set of novel attack techniques for different settings. (1) In a white-box setting where the adversary knows the target model on other participants, we propose regularized Maximum Likelihood Estimation (rMLE), to recover the input from the model parameters and intermediate output. (2) In a black-box setting, the model parameters on other participants are inaccessible to the adversary. We introduce the Inverse-Network technique to identify the mapping from the intermediate output to input. (3) We further consider the query-free black-box setting, in which the adversary cannot query the inference system. We design an approach to reconstruct an alternative version of the target model and then recover the input via rMLE. Our attack results indicate that it is feasible for a compromised participant to accurately recover the input data, even if he has no knowledge of the target model, training data, or capability of querying the system.

The second question is: *Of the target system and model, which characteristics make the inference process more vulnerable to such privacy threats? Which can reduce privacy leakage?* To fully understand the impact of the model features and system designs on the attacks, we conduct empirical evaluations on different model partitioning strategies and adversary's capabilities. Through quantitative comparisons, we identify the critical features and conditions that determine the success of the model inversion attacks. We hope our findings can guide machine learning researchers and practitioners to design more secure collaborative inference systems.

The key contributions of this paper are:

- The first systematic study of inference data privacy in collaborative machine learning systems.
- The *regularized Maximum Likelihood Estimation* technique to recover inference data under the white-box setting;
- The *Inverse-Network* technique to recover inference data under the black-box setting;
- The *query-free shadow model reconstruction* technique to recover inference data under the query-free setting;
- Quantitative discussion about the impact of system and model features on the attacks, and investigation of defense strategies.

The rest of the paper is organized as follows: Section 2 gives the background of DNN and collaborative inference. Section 3 presents the threat models in our consideration and experimental configurations. Sections 4, 5 and 6 describe attacks under white-box, black-box and query-free settings, including attack approaches, implementations and evaluation results. Section 7 compares different attack factors, and discusses possible mitigation solutions. We give related work in Section 8 and conclude in Section 9.

2 BACKGROUND

2.1 Deep Neural Networks

A DNN is a parameterized function $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ that maps an input tensor $x \in \mathcal{X}$ to an output tensor $y \in \mathcal{Y}$. Various neural network architectures have been proposed, e.g., multilayer

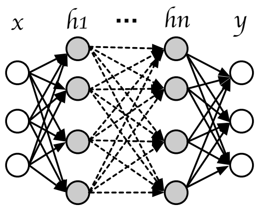


Figure 1: DNN architecture

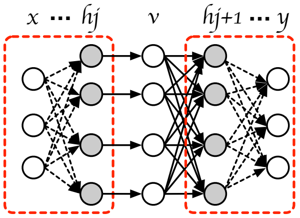


Figure 2: Collaborative inference

perceptrons [36], convolutional neural networks [28] and recurrent neural networks [38].

Figure 1 shows the structure of a DNN. It usually consists of an input layer, an output layer and a sequence of hidden layers between the input and output. Each layer is a collection of units called *neurons*, which are connected to other neurons in the previous layer and the next layer. Each connection between the neurons can transmit a signal to another neuron in the next layer. In this way, a neural network transforms the inputs through hidden layers to the outputs, by applying operations (e.g., linear function or element-wise nonlinear activation function) in each layer.

Model training. The training process of a neural network is to find the optimal parameters θ that can accurately reflect the relationship between $\mathcal{X}$ and $\mathcal{Y}$. To achieve this, a training dataset $D^{train} = \{x_i^{train}, y_i^{train}\}_{i=1}^{N}$ with N samples is needed, where $x_i^{train} \in \mathcal{X}$ is the feature data and $y_i^{train} \in \mathcal{Y}$ is the corresponding ground-truth label. Then a loss function L is applied to measure the distance between the ground-truth output y_i^{train} and the predicted output $f_\theta(x_i^{train})$. The goal of training a neural network is to minimize this loss function (Eq. 1). Backward propagation [14] and stochastic gradient descent [35] are commonly used methods to approximately achieve this goal. The optimal parameters θ^* together with the network topology form the deep learning model.

$$\theta^* = \arg\min_{\theta}(\sum_{i=1}^{N} L(y_i^{train}, f_\theta(x_i^{train}))) \tag{1}$$

Model inference. After the model training is completed, given an input x, the corresponding output can be calculated as $y = f_{\theta^*}(x)$.

2.2 Collaborative Inference

In a collaborative inference system, a DNN is partitioned into n parts: $f_\theta = f_{\theta 1} \cdot f_{\theta 2} ... f_{\theta n}$. Each part $f_{\theta i}$ contains several layers, and is distributed to a participant $\mathbb{P}_i$. Given an input x, the first participant $\mathbb{P}_1$ generates $v_1 = f_{\theta 1}(x)$ and sends it to $\mathbb{P}_2$. Each participant $\mathbb{P}_i$ receives the intermediate value v_{i-1} from $\mathbb{P}_{i-1}$, calculates $v_i = f_{\theta i}(v_i)$, and passes it to $\mathbb{P}_{i+1}$. The final participant $\mathbb{P}_n$ generates the final output $y = f_{\theta n}(v_{n-1})$. Figure 2 shows an example of a collaborative inference system with two participants.

A use case. With the growing proliferation of Internet of Things (IoT), we need ways to deploy deep learning inference applications on commodity resource-constrained edge devices [50]. Running the entire application on the edge device has several challenges: the limited computation resources of the device can cause significant latency; the limited storage capacity makes it hard to store a large DNN model; the limited battery capacity causes a critical energy consumption constraint. An alternative is to offload the entire DNN model and inference computation to the cloud. The edge device sends the input data to the cloud and receives the output. While this can resolve the limitations of edge devices, it can incur significant communication costs when sending a large volume of input data. Besides, there can be privacy issues of the inference data and integrity issues of the model [21], if the cloud is not trusted.

An optimized strategy is to adopt collaborative inference between the edge device and the cloud [8, 17, 25, 27, 43]. The first few simple layers of the DNN model are deployed on the edge device, while the remaining complex layers are offloaded to the cloud. This approach can reduce communication costs, as the intermediate output can be designed to be much smaller than the raw input. Such low data transfer bandwidth can also achieve lower latency. Collaborative inference makes it feasible and efficient to deploy large-scale intelligent workloads on today's edge platforms.

Collaborative inference can also provide better privacy protection, as the cloud now only receives the intermediate values instead of the raw data [43]. The raw data can cause significant privacy issues, e.g. biosensor readings, medical diagonosis and examination data, and facial images. In this paper, however, we show that information leakage is still possible in collaborative inference. An untrusted cloud can easily and accurately recover the sensitive data from the intermediate values without accesses to the edge device. The existing ML privacy protection mechanisms include leverging data obfuscation, Trusted Execution Environment (TEE), homomorphic encryption and differential privacy. We discuss their feasibility and potential drawbacks in Section 7.4.

3 PRELIMINARIES

3.1 Threat Model

Without loss of generality, we consider a collaborative inference system between two participants, $\mathbb{P}_1$ and $\mathbb{P}_2$. The target model is split into two parts: $f_\theta = f_{\theta 2} \cdot f_{\theta 1}$. $\mathbb{P}_1$ performs the earlier layers $f_{\theta 1}$, while $\mathbb{P}_2$ performs the later layers $f_{\theta 2}$. We consider $\mathbb{P}_1$ is trusted: when an input is fed into $f_{\theta 1}$,

$\mathbb{P}_1$ correctly processes it and never leaks it to other parties. However, $\mathbb{P}_2$ is untrusted, attempting to steal the input. This assumption is reasonable in the edge-cloud scenario: the model owner configures and operates the local device ($\mathbb{P}_1$) to ensure the computation is trusted. But he does not have control over the cloud server ($\mathbb{P}_2$), which may be untrusted.

Our threat model can also be applied to systems with more than two participants. As most real systems are split into two parties, without loss of generality, for the rest of this paper, we use a two-participant system to describe and evaluate our attacks. Multi-participant attacks can be achieved in a similar way. $\mathbb{P}_2$ can be defined by all consecutive adversarial participants from an intermediate layer to the last layer. $\mathbb{P}_1$ can be defined as the initial layers to that intermediate layer.

Adversary's capabilities. We assume the untrusted participant $\mathbb{P}_2$ strictly follows the collaborative inference protocols: receiving $v = f_{\theta 1}(x)$ from $\mathbb{P}_1$ and generating $y = f_{\theta 2}(v)$. He cannot compromise the inference process conducted by $\mathbb{P}_1$, and he has no knowledge of the input x, nor any intermediate values inside $\mathbb{P}_1$, except v.

We consider the adversary with different capabilities, summarized in Table 1. These capabilities include knowledge of the target model $f_{\theta 1}$, knowledge of training data, and access to the target system for a query. Based on these, we consider three types of settings:

In the white-box setting (Section 4), $\mathbb{P}_2$ obtains knowledge of the DNN layers $f_{\theta 1}$ controlled by $\mathbb{P}_1$, including the network structure and parameters. Then the adversary can use the model parameters to recover input data, without the requirements of knowing training data or querying models.

In the black-box setting (Section 5), we relax the assumption about the knowledge of the target model. The adversary can only learn information about the model $f_{\theta 1}$ indirectly through querying the inference system. We demonstrate that the adversary can recover the sensitive input when he knows the values, or distribution of the original training dataset, or neither.

We further consider the query-free setting (Section 6), which is a special case of the black-box scenario without the capability of model query. This type of attacks needs the lowest requirements. We show that the adversary can recover the data even when he has no knowledge of the edge model and cannot query the model.

Table 1: Adversary's capability in our consideration (✓: the adversary needs this capability; –: this capability is not necessary.)

Setting	Target model		Training Data		Inference
	Parameters	Structure	Values	Distribution	Query
Section 4 White-box	✓	✓	–	–	–
Section 5 Black-box	–	–	✓	✓	✓
	–	–	–	✓	✓
	–	–	–	–	✓
Section 6 Query-free	–	✓	✓	✓	–
	–	–	✓	✓	–
	–	✓	–	✓	–
	–	–	–	✓	–

3.2 Experimental Configurations

In the rest of this paper, we evaluate our attacks on two standard DNN benchmark datasets: MNIST and CIFAR10.

The target models we try to find inverses are convolutional neural networks (CNN). Specifically, we adopt LeNet (2 convolutional layers and 3 fully connected layers) on the MNIST dataset, and a CNN with 6 convolutional layers and 2 fully connected layers on the CIFAR10 dataset. We split each model at different layers (mainly convolutional layers). Table 2 lists the detailed experimental configurations. These configurations are realistic in the case of edge-cloud scenarios, as the most heavy-computational layers (including all fully-connected layers) are offloaded to the cloud. We will explore the cases that the model is split at fully-connected layers in Section 7.1.

Table 2: Experiment Configurations

Dataset	MNIST	CIFAR10
Target Model	LeNet-5 (2 conv + 3 fc)	6 conv + 2 fc CNN
Split point	• 1st conv layer (conv1) • 2nd conv layer after activation (ReLU2)	• 1st conv layer (conv11) • 4th conv layer after activation (ReLU22) • 6th conv layer after activation (ReLU32)

We follow the standard MNIST and CIFAR split for training and testing samples [1]. We set the learning rate to 10^{-3} and choose ADAM as our optimizer. The target models and all attacks are implemented with Pytorch 1.0.1. We run our experiments on a server with 1 Nvidia 1080Ti GPU, 2 Intel Xeon E5-2667 CPUs, 32MB cache, 64GB memory and 2TB hard-disk.

3.3 Evaluation Metrics

To quantify the attack results, we adopt two metrics, Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) [47]. PSNR mathematically measures the pixel level recovery quality of the image. It is defined in Equation 2, where $MAXI$ is the maximum possible intensity of a signal (255.0 for images), and $I(i, j)$ refers to the intensity at position (i, j) of image I. SSIM measures the human perceptual similarity of two images. It considers luminance, contrast and structure of two images. SSIM is a single value between 0 and 1, where 0 represents least similar and 1.0 indicates most similar.

$$PSNR(I, I_0) = 10log(\frac{MAXI^2}{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(I(i,j) - I_0(i,j))^2}) \quad (2)$$

4 WHITE-BOX ATTACKS

We start from the white-box setting, where the adversary participant knows the parameters of the initial layers $f_{\theta 1}$ on the trusted participant. As deep learning frameworks and tools become prevalent and mature, many trained models are published online for free use, covering various prediction tasks. It is a common practice for model owners to directly download and deploy these models. In this case, the adversary participant has white-box access to the target model.

Formally, the model inversion problem is: how can the adversary recovers an input x_0, from the corresponding intermediate value $f_{\theta 1}(x_0)$, and the model parameters $\theta 1$? We propose regularized Maximum Likelihood Estimation (rMSE) to solve this problem.

4.1 Regularized Maximum Likelihood Estimation

We treat the model inversion as an optimization problem: given $f_{\theta 1}(x_0)$, our goal is to find a generated sample x, that satisfies two requirements: (1) the intermediate output of this sample, $f_{\theta 1}(x)$, is similar to $f_{\theta 1}(x_0)$; (2) x is a natural sample, following the same distribution as other inference samples.

For requirement (1), we use the Euclidean Distance (ED) to measure the similarity between $f_{\theta 1}(x)$ and $f_{\theta 1}(x_0)$ (Eq. 3(a)). Note that $f_{\theta 1}(x)$ can be interpreted as the mapping from the input space (unobservable to the adversary) to the feature space (observable to the adversary). Then this Euclidean Distance represents the *posteriori* information from the adversary's intermediate-level observation. Our goal is to find the optimal sample x that minimizes this distance.

For requirement (2), we adopt the *Total Variation* [37] to represent the *prior* information of an input sample. The total variation of a 2D image x is defined in Equation 3(b), where $x_{i,j}$ represents the pixel at position (i, j). The total variation encourages the generated image x to be piece-wise smooth, i.e. avoiding drastic variations inside regions but allowing large changes along the region boundaries, controlled by β.

$$ED(x, x_0) = ||f_{\theta 1}(x) - f_{\theta 1}(x_0)||_2^2 \quad (3a)$$

$$TV(x) = \sum_{i,j}(|x_{i+1,j} - x_{i,j}|^2 + |x_{i,j+1} - x_{i,j}|^2)^{\beta/2} \quad (3b)$$

$$x^* = argmin_x \;\; ED(x, x_0) + \lambda TV(x) \quad (3c)$$

The total objective function of the model inversion problem is a combination of feature space similarity and natural-input a priori, as shown in Eq. 3c. In this equation, λ is a hyperparameter to balance the effects of the two terms. If the feature space $f_{\theta 1}(x)$ is far from the input space, i.e. a lot of network layers are computed on the trusted participant $\mathbb{P}_1$, a large λ is required because less posterior information about the input can be recovered from the feature space and the adversary needs to rely on the prior information. In contrast, if only a small number of layers are deployed on $\mathbb{P}_1$, then the adversary only needs to select a small λ. We perform gradient descent (GD) to solve Eq. 3c and recover the image.

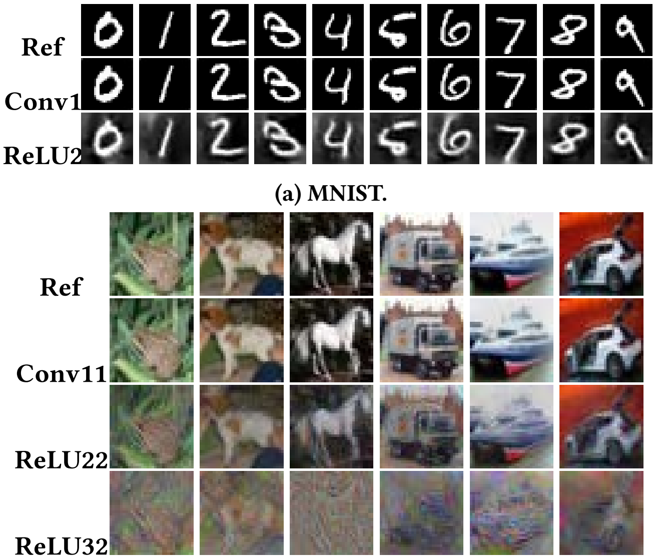
Algorithm 1 White-box model inversion attack

```
**Function** WhiteboxAttack($f_{\theta 1}$, $f_{\theta 1}(x_0)$, $T$, $\lambda$, $\epsilon$)
/* $f_{\theta 1}$: the target model */
/* $f_{\theta 1}(x_0)$: the intermediate output of sensitive input $x_0$ */
/* $T$: maximum number of iterations */
/* $\lambda$: tradeoff between prior and posteriori information
/* $\epsilon$: step size in GD */

$L(x)$=$||f_{\theta 1}(x) - f_{\theta 1}(x_0)||_2^2 + \lambda TV(x)$
$t = 0$
$x^{(0)}$= ConstantInit()
while ($t < T$) do
    $x^{(t+1)} = x^{(t)} - \epsilon * \frac{\partial L(x^{(t)})}{\partial x^{(t)}}$
    $t$+=1
end for
return $x^{(T)}$
```

Algorithm 1 shows the detailed white-box attack. The input image is initialized with constant gray, i.e. 0.5 for all RGB channels. We choose ADAM [26] to accelerate the optimization. We observe that ADAM converges more stably when performing model inversion from shallow layers. Therefore, we choose a large step size (10^{-2}) and a small iteration number (500) for shallow layers, and a small step size (10^{-3}) and a large iteration number (5000) for deep layers. We choose $\beta = 1.0$ and observe this is enough to generate good results.

4.2 Evaluation

Figure 3 shows the white-box attack results on MNIST and CIFAR10 datasets. For each dataset, the first row shows the target inference samples, and the remaining rows are the recovered images when the split point is at different layers. For MNIST, we observe that the adversary can accurately recover the images with high fidelity, when the split point is either in the first (conv1) or last (ReLU2) convolutional layer. For CIFAR10, when the split layer is in the first (conv11) or fourth (ReLU22) convolutional layer, the recovered images maintain high quality. When the neural network is split after the last convolutional layer (ReLU32), the recovered images are hardly recognizable.



(a) MNIST.

(b) CIFAR10.

Figure 3: Recovered inputs in white-box attacks

Table 3 shows the PSNR and SSIM metrics for each experiment. We observe that when the split point is in a deeper layer, the quality and similarity of recovered images become worse. For CIFAR10, there is a significant drop in SSIM from ReLU22 to ReLU32. These conclusions are consistent with the visual assessments in Figure 3. We set the threshold of SSIM as 0.3: a recovered image with an SSIM value below this threshold (shaded entries in Table 3, and other tables in the following sections) is regarded as being unrecognizable.

Table 3: PSNR (db) and SSIM for white-box attacks

	MNIST		CIFAR10		
	conv1	ReLU2	conv11	ReLU22	ReLU32
PSNR	39.69	15.10	37.59	19.47	13.38
SSIM	0.9969	0.5998	0.9960	0.6940	0.1625

5 BLACK-BOX ATTACKS

Next, we consider the black-box setting, where the adversary does not have knowledge of the structure or parameters of $f_{\theta 1}$. We assume that the adversary can query the black-box model: he can send an arbitrary input x to $\mathbb{P}_1$, and observe the corresponding output $f_{\theta 1}(x)$. This assumption applies to the case where the model owner releases prediction APIs to end users as an inference service. We further relax this assumption in Section 6.

Model inversion attacks under the black-box setting are more challenging, because without the knowledge of model parameters, the adversary cannot directly perform a gradient descent operation on $f_{\theta 1}$ to solve the optimization problem in Equation 3(c). One solution is to first recover the model structure and parameters by querying the model, and then recover the inference samples. The possibility of model reconstruction has been demonstrated in [33, 44, 46]. We prove that the model inversion attacks can be achieved based on the reconstructed model in Section 7.3.

We propose a more efficient approach, Inverse-Network, to directly identify the inversed mapping from output to input, without the need to obtain the model information. Our solution is easier to implement, and can recover inputs with higher fidelity. We describe this approach and evaluate it in this section. Quantitative comparisons between these two solutions are presented in Section 7.3.

5.1 Inverse-Network

Conceptually, the Inverse-Network is the approximated inverse function of $f_{\theta 1}$, trained with $v = f_{\theta 1}(x)$ as input, and x as output. We show the detailed description of Inverse-Network approach in Algorithm 2. The attack consists of three phases: ① generating a training set for the Inverse-Network; ② training the Inverse-Network; and ③ recovering the input sample by querying the Inverse-Network.

First, the adversary generates a bag of samples $X = (x_1, x_2, ..., x_m)$ to query the target system, and observes the corresponding intermediate outputs $V = (f_{\theta 1}(x_1), f_{\theta 1}(x_2), ..., f_{\theta 1}(x_m))$ (Lines 10-17 in Algorithm 2). We consider three cases for selecting X: (1) the adversary has access to the original dataset used for training f_θ, and adopts it as X; (2) the adversary does not have the original training set. Instead, he has a different set following the same distribution; (3) the adversary has neither the original dataset or its distribution. He has to randomly generate some samples. In our experiment, we generate pure noise sampled from the standard Gaussian distribution (zeros mean, unit variance) to form X.

Next, the adversary can directly train an Inverse-Network $f_{\theta 1}^{-1}$ using V as the training input and X as the training target (Lines 19-29). We initialize the Inverse-Network with Xavier initialization [13]. We leverage l_2 norm in the pixel space as the loss function (Equation 4), and stochastic gradient descent (SGD) to train the Inverse-Network. It is worth noting that the architecture of the Inverse-Network is not necessarily related to the target model $f_{\theta 1}$. In our experiment, we use an entirely different architecture.

$$f_{\theta 1}^{-1} = argmin_g \frac{1}{m} \sum_{i=1}^{m} ||g(f_{\theta 1}(x_i)) - x_i||^2 \qquad (4)$$

Algorithm 2 Black-box model inversion attack

```
Function BlackBoxAttack($f_{\theta 1}$, $f_{\theta 1}(x_0)$)
/* $f_{\theta 1}$: the target model */
/* $x_0$: the target sensitive input to recover */
/* $f_{\theta 1}(x_0)$: the intermediate layer output */
$X$=GenerateTrainingSet()
$g$=TrainInverseNet($X$, $f_{\theta 1}$)
$\hat{x_0}$ = Inverse($g$, $f_{\theta 1}(x_0)$)
return $\hat{x_0}$

Function GenerateTrainingSet()
if known training set then
    $X$=data.TrainingSet
else if Known training distribution then
    $X$= NewSet ~ data.TrainingSet
else
    $X$=GaussianNoise
return $X$

Function TrainInverseNet($X$, $f_{\theta 1}$)
/* $k$: BatchSize */
/* $\epsilon$: StepSize */
$g^{(0)}$=Init()
while ($t < T$) do
    Randomly sample $x_1, x_2 ... x_k$ from $X$
    $L(g^{(t)} = \frac{1}{k} \sum_{i=1}^{k} ||g(f_{\theta 1}(x_i)) - x_i||_2^2$
    $g^{(t+1)} = g^{(t)} - \epsilon * \frac{\partial L(g^{(t)})}{\partial g^{(t)}}$
    $t$+=1
end while
return $g^{(T)}$

Function Inverse($g$, $f_{\theta 1}(x_0)$)
$\hat{x_0}$=$g(f_{\theta 1}(x_0))$
return $\hat{x_0}$
```

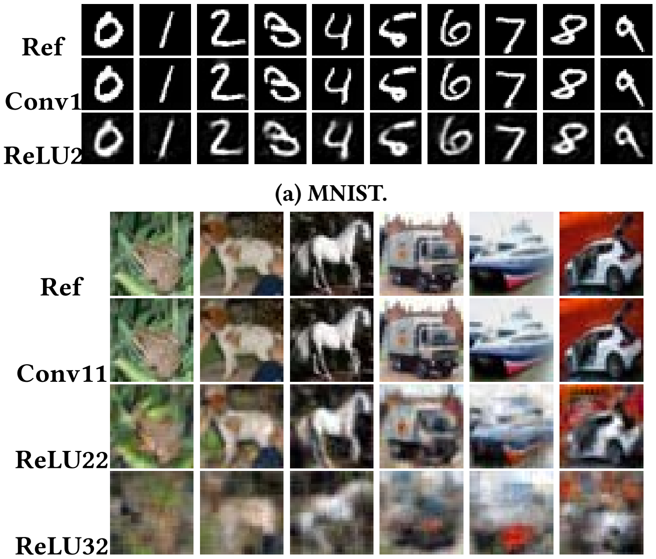
Once the Inverse-Network $f_{\theta 1}^{-1}$ is obtained, the adversary can recover any inference sample from the intermediate-level value: $x = f_{\theta 1}^{-1}(v)$. This approach is more efficient than rMLE: (1) for each target sample, the adversary only needs to pass through the Inverse-Network once, while in rMLE, an iterative process is required to solve the optimization problem; (2) calculating the inversed input is parameter-free, while rMLE requires tuning the parameters (λ, β in Eq. 3).

5.2 Evaluation

Figures 4, 5 and 6 show the recovered images of two datasets under three different circumstances. Table 4 shows the PSNR and SSIM metrics of these attacks. From these recovery results, we draw some conclusions.

First, the adversary can recover the input with black-box access for most cases. The quality of the recovered images in MNIST is very high when the split point is in conv1 or ReLU2. For CIFAR10, the recovered images still maintain high quality when the split point is in a shallow layer (conv11). They become relatively vague and lose certain details when the split point is in a deeper layer, e.g. layer ReLU32 for the CIFAR10 dataset.

Second, we observe that there is no significant difference between the cases where the adversary uses the same training set, or a different set with the same distribution to train the Inverse-Network. For MNIST, the attack with a different set are even slightly better than the ones with the same set. However, when the adversary does not know the training data distribution, and adopts randomly generated samples, the attack effects drop significantly. This is especially prominent in the case of the CIFAR10 split in the ReLU22 layer. We conclude that the knowledge of the training data distribution is important to recover samples from deep layers.



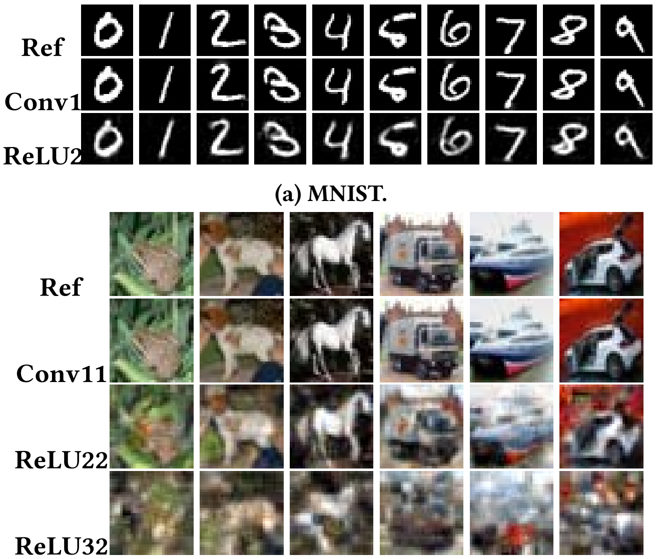
(a) MNIST.

(b) CIFAR10.

Figure 4: Recovered inputs in black-box attacks (same training data)

6 QUERY-FREE ATTACKS

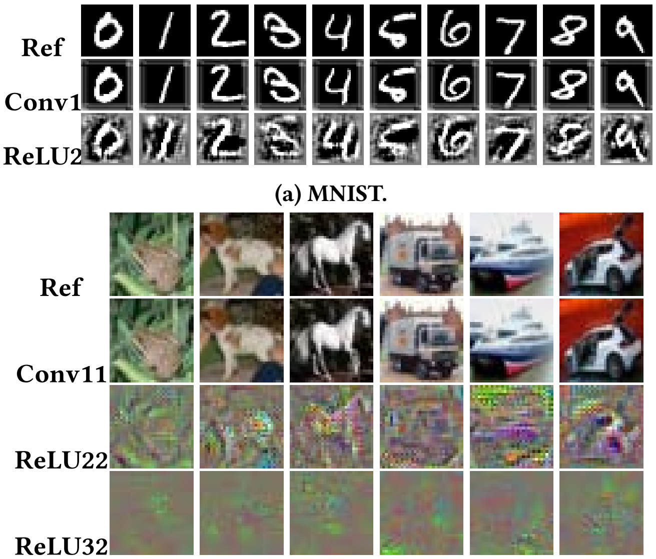
The Inverse-Network approach requires the adversary to be able to query the target model, to generate the data set for training f_θ^{-1}. In this section, we consider the query-free setting, where the adversary cannot query the system, and does not know the client-side model information. The basic idea is that the adversary first reconstructs a shadow model, which can imitate the target model's behavior, and then uses



(a) MNIST.

(b) CIFAR10.

Figure 5: Recovered inputs in black-box attacks (different training data, same distribution).



(a) MNIST.

(b) CIFAR10.

Figure 6: Recovered inputs in black-box attacks (different distributions).

rMLE over this shadow model to recover the sensitive input samples.

6.1 Shadow Model Reconstruction

The problem in our consideration is: how can the adversary reconstruct a shadow model of the former model layers, $f_{\theta 1}$, only with the knowledge of the latter layers $f_{\theta 2}$ and a dataset S drawn from the same distribution as the original training set? He cannot query the model with specified samples to get the intermediate values.

Table 4: PSNR (db) and SSIM for black-box attacks

	Dataset	MNIST		CIFAR10		
		conv1	ReLU2	conv11	ReLU22	ReLU32
PSNR	same set	39.64	20.35	49.88	19.81	15.42
	same dist	40.72	20.81	49.02	19.36	13.95
	rand set	14.76	7.72	48.59	12.79	12.37
SSIM	same set	0.9887	0.7334	0.9993	0.6939	0.3124
	same dist	0.9950	0.8046	0.9992	0.6802	0.2196
	rand set	0.7188	0.4310	0.9996	0.2930	0.0440

The key insight of our approach is that, if the shadow model is reconstructed as $f'_{\theta 1}$, it should be able to classify the input with high accuracy when combined with the later layers $f_{\theta 2}$:

$$y_i \sim f_{\theta 2}(f_{\theta 1}(x_i)) \sim f_{\theta 2}(f'_{\theta 1}(x_i)), \text{for } (x_i, y_i) \in S \qquad (5)$$

Then the task of model reconstruction can be translated into minimizing the classification error of the composition of the two models: $f_{\theta 2}(f'_{\theta 1}(x_i))$ versus y_i. Equation 6 shows the loss function for training the model, where m is the number of samples in S, *CrossEntropy* is the cross-entropy loss. Equivalently, this means the training process of $f'_{\theta 1}$ is supervised at the output layer of $f_{\theta 2}$. Once the model $f'_{\theta 1}$ is reconstructed, the adversary can perform model inversion attacks using the rMLE technique in Section 4.

$$f'_{\theta 1} = argmin_g \frac{1}{m} \sum_{i=1}^{m} CrossEntropy(f_{\theta 2}(g(x_i)), y_i) \qquad (6)$$

Algorithm 3 describes the query-free attacks. There are two phases: ① offline shadow model reconstruction (Lines 10-21) and ② online model inversion (Line 7). The shadow model reconstruction only needs to be performed once. Then all the input samples can be recovered using the same shadow model, by inferencing only once for each input .

In the shadow model reconstruction phase, a training set and an initial network are required (Lines 13-14). We consider four cases with different adversary's capabilities in two dimensions, i.e. training set and network structure: (1) the adversary's dataset S is the same as the original set used for training f_θ. He also knows the network structure of $f_{\theta 1}$; (2) the adversary has a different dataset S from the original training set, but follows the same distribution. This assumption is reasonable, because there exist various public datasets for different tasks. He knows the network structure of $f_{\theta 1}$; (3) the adversary has the same training set. But he does not know the structure of $f_{\theta 1}$. He has to use an alternative one

Algorithm 3 Query-free model inversion attack

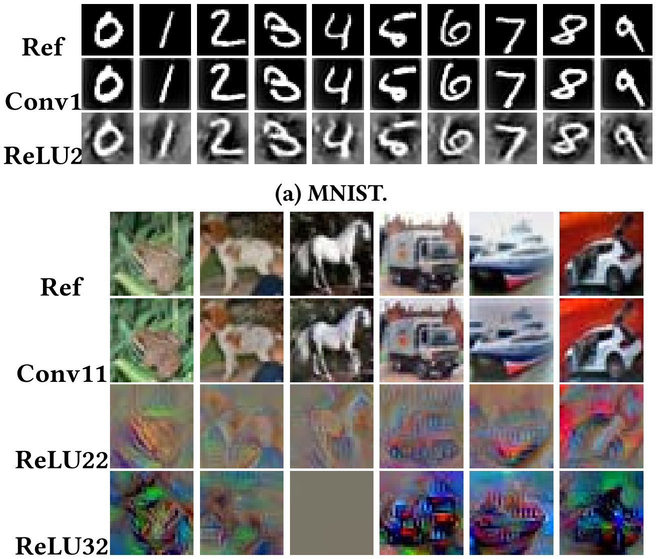
```
Function QueryFreeAttack($f_{\theta 1}$, $f_{\theta 2}$, $f_{\theta 1}(x_0)$)
/* $f_{\theta 1}$: the target model */
/* $f_{\theta 2}$: the known model */
/* $x_0$: the target sensitive input to recover */
/* $f_{\theta 1}(x_0)$ the intermediate layer output */
$\hat{f_{\theta 1}}$=ModelReconstruction($S$, $f_{\theta 2}$)
$\hat{x_0}$ = WhiteboxAttack($\hat{f_{\theta 1}}$, $f_{\theta 1}(x_0)$, $T$, $\lambda$, $\epsilon$)
return $\hat{x_0}$

Function ModelReconstruction($f_{\theta 2}$)
/* $k$: BatchSize */
/* $\epsilon$: StepSize */
$S$=GenerateTrainingSet()
$g_0$=InitArchitecture()
while ($t < T$) do
    Randomly sample $x_1, x_2 ... x_k$ and labels $y_1, y_2 ... y_k$ from $S$
    $L(g^t) = \frac{1}{k} \sum_{i=1}^{k} y_i (f_{\theta 2}(g^t(x_i))) + (1 - y_i)(1 - f_{\theta 2}(g^t(x_i))$
    $g^{(t+1)} = g^t - \epsilon * \frac{\partial L(g^t)}{\partial g^t}$
    $t$+=1
end while
return $g^{(T)}$
```

for the shadow model. We assume that both the target model and the shadow model are convolutional neural networks, but with different numbers of layers and filters, as well as filter sizes. Table 5 shows the network structure configurations used in our experiments; (4) the adversary does not know the training set nor the network structure.

After the training set and network structure are determined, the adversary can adopt SGD to optimize the loss function of the composition of the two models. We choose the cross-entropy loss because it performs well on image classification tasks. Other loss functions can be leveraged, if the adversary aims to find inverses of the DNN for different tasks. Once the shadow model is obtained, the adversary can use rMLE (Algorithm 1) to recover the inputs.

Table 5: Neural network configurations for query-free attacks.

Dataset	Layer	Target Model	Shadow Model
MNIST	conv1	One 5X5 conv layer	Two 3X3 conv layers
	ReLU2	Two 5X5 conv layers	Four 3X3 conv layers
CIFAR10	conv11	One 3X3 64 filters layer	One 3X3 16 filters layer + one 3X3 64 filters layer
	ReLU22	Two 3X3 64 filters layers + two 3X3 128 filters layers	One 5X5 filters layer + one 5X5 128 filters layer



(a) MNIST.

(b) CIFAR10.

Figure 7: Recovered input in query-free attacks (same training data, same network structure).

6.2 Evaluation

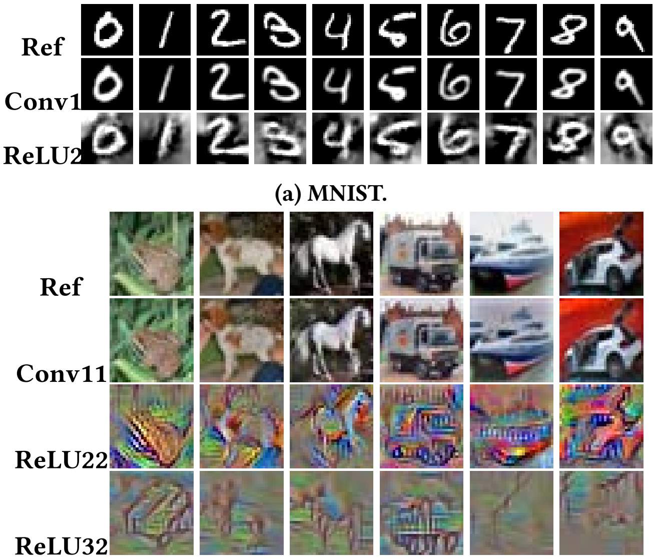
We illustrate the recovered images under the four adversary's capability settings in Figures 7, 8, 9 and 10 respectively. The corresponding quantitative results are listed in Table 6.

For MNIST, the adversary can still recover the input images from conv1 and RelU2 layers. The quality of the images is relatively lower than the ones in the white-box or black-box setting. For CIFAR10, attacks are successful only from the shallow conv11 layer with the knowledge of training set or network structure. These results indicate that query-free attacks are harder to achieve than white-box or black-box attacks. This is straightforward, as the adversary now has smaller capabilities. Besides, more layers on the trusted participant can also increase the difficulty of image recovery.

We also observe that a different training set with the same distribution has similar effects on model inversion attacks. So the adversary does not need to know the exact training set for attacks. This is also observed in the black-box setting (Section 5). However, if the adversary has no knowledge of the network structure, then an alternative network has worse performance. This emphasizes the importance of knowledge of network structure for a model inversion attack.

7 DISCUSSIONS

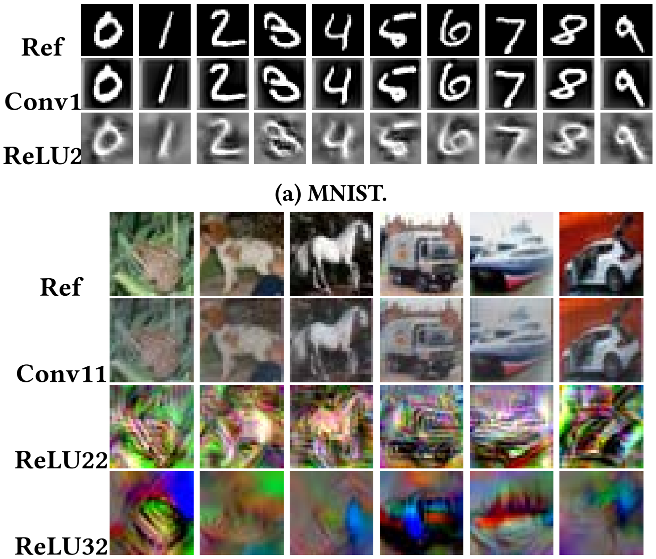
In this section, we review, summarize and compare the attack results under different settings. We explore the impacts of system features and adversary's capabilities on the model inversion attacks. We also discuss possible defense solutions.



(a) MNIST.

(b) CIFAR10.

Figure 8: Recovered input in query-free attacks (different training data, same network structure).



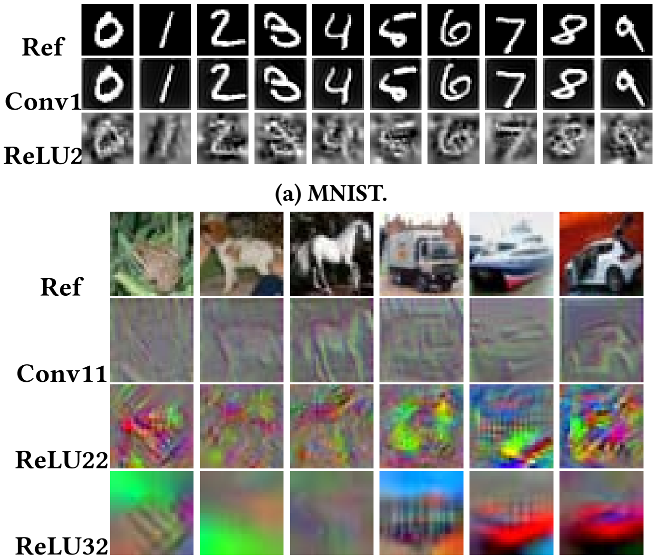
(a) MNIST.

(b) CIFAR10.

Figure 9: Recovered input in query-free attacks (same training data, different structures.)

7.1 Impact of System Configurations

From the results in previous sections, we observe that different split points can yield different attack effects. This raises an important question: *how to split the neural network in the collaborative system, to make the inference data more secure?*



(a) MNIST.

(b) CIFAR10.

Figure 10: Recovered input in query-free attacks (different training data, different structures.)

Table 6: PSNR (db) and SSIM for query-free attacks

	Dataset	MNIST		CIFAR10		
	Net Structure	**conv1**	**ReLU2**	**conv11**	**ReLU22**	**ReLU32**
PSNR	same set, same net	17.60	9.61	21.16	12.74	11.09
	diff sets, same net	21.53	9.27	21.45	11.51	11.98
	same sets, diff nets	12.59	8.05	17.55	12.46	10.68
	diff sets, diff nets	17.86	8.03	13.06	11.30	11.47
SSIM	same set, same net	0.7423	0.4981	0.9104	0.1752	0.0419
	diff sets, same net	0.9121	0.4652	0.9145	0.1723	0.0102
	same sets, diff nets	0.6430	0.3790	0.6344	0.2714	0.0247
	diff sets, diff nets	0.6952	0.3226	0.1553	0.0467	0.0793

We use the query-free attack over the LeNet model (MNIST dataset) as an example to explore this question. We select the split point at each layer, and perform model inversion attacks. Figures 11 and 12 show the recovered images, and PSNR/SSIM metrics respectively.

Generally, we observe that the quality of recovered images decreases when the split layer goes deeper. This is straightforward as the relationship between input and output becomes more complicated and harder to revert when there are more layers. Besides, we also observe that the image quality drops significantly, both qualitatively (Figure 11) and quantitatively (Figure 12), on the fully-connected layer (fc1), indicating that model inversion with fully-connected layers is much harder than for convolutional layers. The reason is that a convolutional layer only operates on local elements (the locality depends on the kernel size), while a fully-connected layer entirely mixes up the patterns from the previous layer. Besides, the number of output neurons in a fully-connected layer is typically much smaller than input neurons. So it is relatively harder to find the reversed relationship from the output of the fully-connected layer to the input.

Unfortunately, privacy is usually not considered when selecting the optimal split point in a collaborative system. In the case of an edge-cloud scenario, most layers (including all fully-connected layers) are commonly offloaded to the cloud, while the edge device only computes a small number of convolutional layers for feature extraction, due to power and resource constraints [25]. This gives a chance for an untrusted cloud provider to steal sensitive inference input.

Takeaway: When selecting the split point in a collaborative inference system, privacy should also be considered, in addition to latency and power constraints. We recommend placing at least one fully-connected layer on the trusted participant to hide the information of sensitive input samples.

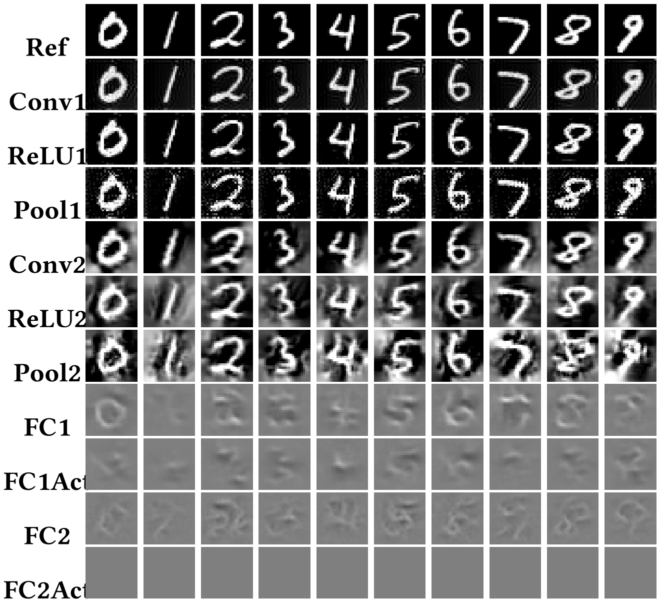


Figure 11: Recovered images in query-free attacks

7.2 Impact of Adversary's Capabilities

In addition to the selection of split point, the adversary's capability can also have an impact on the attack results. The question we consider is: *which capabilities are critical for model inversion attacks?*

Knowledge of target model. If the adversary can query the system, then it is not necessary for him to know the parameters or network structure on the trusted participant. Comparing Tables 3 and 4, we find that the effects of black-box attacks using our Inverse-Network technique are as good as the white-box attacks using rMLE technique. However, if the adversary does not have access to the model query APIs and model parameters (the query-free setting), then the knowledge of network structure plays a relatively important role in recovering inputs, as discussed in Section 6.2 and Table 6.

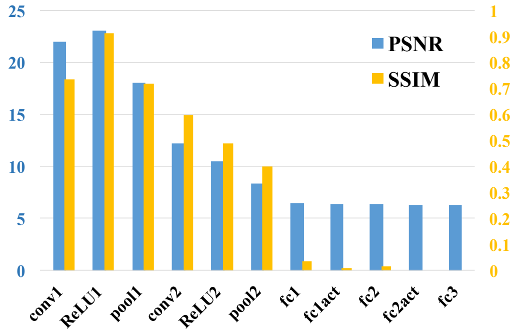


Figure 12: PSNR and SSIM in query-free attacks.

Knowledge of the training set. The adversary does not need to know the exact training set. Using a different set following the same distribution, the adversary can recover the input images with similar quality in the black-box setting (Table 4), or the query-free setting (Table 6). However, the knowledge of training data distribution is very critical: without such information, the adversary has to use randomly generated samples to reverse the network in the black-box attacks, whose performance drops significantly (Table 4). In the query-free case, the adversary cannot reconstruct the model without knowing the training data distribution.

Capability of model query. This is also a critical requirement for model inversion attacks. If the adversary is not able to query the model in a black-box setting, he has to reconstruct the model before recovering the input. It takes more effort to implement the attacks, and the performance is lower (comparing Tables 4 and 6).

Takeaway: We recommend the model owner trains the target model using a training set whose distribution is unknown to the adversary. Restricting the query APIs from untrusted participants can also make the attacks harder.

7.3 Comparisons of Attack Techniques

We propose three different attack techniques under different threat models. We summarize and compare these techniques.

Table 7: Applicability of techniques under different settings.

	rMLE	Inverse Network	Shadow Model
White-box	✓	✓	✓
Black-box	–	✓	✓
Query-free	–	–	✓

Applicability. Table 7 lists the effectiveness of each technique under different settings. The white-box scenario is the most basic setting: since the adversary knows all the details about the target model, other techniques without such an assumption can also be applied here, although some of them may not be efficient (e.g., reconstructing the shadow model). For the black-box setting, since the adversary does not know the model parameters, he cannot use the rMLE technique. He can either adopt the Inverse-Network approach, or reconstruct the shadow model and then use rMLE to recover the input. For the query-free scenario, since the adversary does not know the model parameters, and has no access to query the model, he can only use the shadow model reconstruction with rMLE to recover the image.

Table 8: Comparison of Inverse-Network and shadow model reconstruction in the black-box setting.

	Technique	MNIST		CIFAR10		
		conv1	ReLU2	conv11	ReLU22	ReLU32
PSNR	Inverse-network	39.64	20.35	49.87	19.81	15.41
	Model reconstruction	39.67	15.41	28.67	18.02	12.61
SSIM	Inverse-network	0.9887	0.7334	0.9993	0.6939	0.3124
	Model reconstruction	0.9968	0.6103	0.9766	0.6893	0.1145

Performance. We first compare the attack performance of the rMLE (Table 3) and Inverse-Network [1] (Table 4) approaches in the white-box setting, respectively. We observe that when the adversary knows either the training data or its distribution, the recovered images from Inverse-Network maintain higher quality than the ones from rMLE. Otherwise, rMLE performs better than Inverse-Network with randomly generated samples.

We then compare the performance of Inverse-Network and shadow model reconstruction solutions in the black-box setting. As introduced in Section 5, the adversary can query the model to get pairs of input and corresponding intermediate values, based on which he can reconstruct a shadow model. We implement this approach and show the quantitative comparisons with Inverse-Network in Table 8. We find that Inverse-Network has better results than shadow model reconstruction for most datasets and split points.

Takeaway: For the white-box setting, if the adversary has no knowledge of the training set or distribution, he can use rMLE for better performance. Otherwise, he can select Inverse-Network, as it has better results, and takes less effort to implement and perform. For the black-box setting, Inverse-Network is recommended over shadow model reconstruction. For the query-free setting, shadow model reconstruction is the only applicable method.

7.4 Defenses

Since current privacy-preserving algorithms and systems all focus on training data, we provide some possible defense strategies to mitigate the inference privacy attacks discussed in this paper.

Running fully-connected layers before sending out results. As shown in Section 7.1, a fully-connected layer can mix up inputs, and hide information about the inference samples. So a model owner can deploy at least one such layer on the first trusted participant. This makes it very difficult for the adversary to recover the input. Typically, the fully-connected layers follow convolutional layers in a DNN. This requires computing all convolutional layers on the client-side, which can be heavy for an edge device.

Make the client-side network deeper. As illustrated in Figures 11 and 12, both qualitative and quantitative measurements of the inversed images become worse as the network becomes deeper. Therefore, making the client-side network deeper can help mitigate the attacks. On the other hand, deeper networks increase the compution on the client-side. The client device or IoT device may not have sufficient computation, storage or battery resources for this, nor for the above mitigation strategy.

Trusted Execution on untrusted participants. The hardware support for a Trusted Execution Environment (TEE), e.g., Intel SGX, ARM TrustZone, is effective at secure remote computation and data confidentiality protection against privileged adversaries. In the case of collaborative DNN computation, the inference application can be deployed inside a TEE (or secure enclave) on the untrusted participants, and the intermediate values are encrypted against the adversary during transmission between participants. This can provide privacy protection for the inference data. However, this requires special architecture support on the cloud side, and careful crypto key management.

Differential privacy. We can use differential privacy to add noise and obfuscate sensitive information. Specifically, we can add noise to the inference input, and the intermediate value becomes $v = f_{\theta 1}(x + \epsilon)$. We can also add noise directly to the intermediate value before sending it to the untrusted participant: $v = f_{\theta 1}(x) + \epsilon$. In these two cases ϵ is the random noise that satisfies differential privacy. It is obvious that there exists a trade-off between usability and privacy: as a higher level of noise is added, the model accuracy may drop.

Homomorphic encryption. This allows the inference application on the untrusted participant to directly perform DNN computations on encrypted input, so the sensitive information will not be leaked. A drawback of homomorphic encryption is that it suffers from huge inefficiency and is not applicable for all DNN operations.

[1] Inverse-Network gives the same results for both white-box and black-box settings.

8 RELATED WORK

8.1 Machine Learning Privacy Attacks

Training data privacy attacks. There are different types of privacy attacks against the training data. The first type is *property inference attacks*, which tries to infer some properties of the training data from the model parameters. Attacks were demonstrated in traditional machine learning classifiers [3] and fully-connected neural networks [12].

A special case of property inference attacks is *membership inference attacks*, which infers whether one individual sample is included in the training set. This attack was first presented in [41]. The following work explored the feasibility of attacks with different adversary's capabilities [39], model features [30, 49], in Generative Adversarial Networks [18, 29], and collaborative training systems [32].

The second type are *model inversion attacks* [10]: given a machine learning model, and part of the training samples' features, the adversary can recover the rest of the features of the samples. Advanced model inversion attacks were designed to recover images from deep neural networks in single-party systems [9], and collaborative learning systems [22].

The third type are *model encoding attacks* [42]: the adversary with direct access to the training data can encode the sensitive data into the model for a receiver entity to retrieve.

Model privacy attacks. The adversary attempts to steal the model parameters [44], hyperparameters [46] or structures [23, 33], via prediction APIs, memory access side channels, etc.

Inference data privacy attacks. Closer to our study is the work [48], which adopted a power side channel to recover inference data. However, this attack required the adversary to compromise the victim device for side-channel information collection, and it could only recover simple images (single pixel). Our attack is applied to the collaborative systems, and can recover any arbitrary complex data without access or knowledge of the victim's computation and data.

8.2 Machine Learning Privacy Solutions

Current solutions only focus on training data protection:

Enhancing the algorithms. Distributed training was introduced to protect the training data [15, 40], as different participants can use their own data for model training. The SGX security enclaves in Intel processors were used to protect the training tasks against privileged adversaries [24, 34]. Cao et al. [5] proposed a methodology to remove the effects of sensitive training samples on the models. Abadi et al. [2] applied differential privacy to add noise in the stochastic gradient descent process to eliminate the parameters' dependency on the training data.

Enhancing the dataset. Bost et al. [4] proposed to encrypt the data before feeding them into the training algorithm. They designed machine learning operators which can operate on the encrypted data. Zhang et al. [51] showed that adding noise to the training dataset is effective in protecting training data privacy. Generative Adversarial Network with differential-privacy is adopted [45, 52] to generate artificial data for training DNN models while removing sensitive information from the original data.

9 CONCLUSIONS

While the privacy threat of training data in deep learning is well studied, and defenses have been investigated, the privacy of inference data is less studied. In this paper, we explore the feasibility of recovering sensitive data in the deep learning inference process. We discover that in a collaborative inference system, an adversary who controls one participant can easily recover the inference samples from intermediate values. We propose a new set of attack techniques to compromise the inference data privacy in collaborative deep learning systems, under different attack settings. We systematically compare these different techniques, demonstrating that the adversary can successfully and reliably recover the inputs with very few prerequisites.

We hope that the importance of inference data privacy protection can be addressed through this study. For instance, when selecting the split point for edge-cloud offloading, previous work only considered the power and performance requirements. With the feasibility of stealing the inference data, privacy should also be an important factor for partitioning the neural network. Future work could include the study of the trade-off among power, performance and security for edge-cloud offloading, exploration of more powerful attacks, and realization of possible defense mechanisms.

REFERENCES

[1] 2018. https://pytorch.org/docs/0.4.0/torchvision/datasets.html.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *ACM Conference on Computer and Communications Security.*

[3] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. 2015. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks* (2015).

[4] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine learning classification over encrypted data.. In *Network and Distributed System Security Symposium.*

[5] Yinzhi Cao and Junfeng Yang. 2015. Towards Making Systems Forget with Machine Unlearning. In *IEEE Symposium on Security and Privacy.*

[6] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project adam: Building an efficient and scalable deep learning training system. In *USENIX Symposium on Operating Systems*

Design and Implementation.

[7] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems.*

[8] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2018. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *arXiv preprint arXiv:1801.08618* (2018).

[9] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM Conference on Computer and Communications Security.*

[10] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing.. In *USENIX Security Symposium.*

[11] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. 2018. Property Inference A acks on Fully Connected Neural Networks using Permutation Invariant Representations. In *ACM Conference on Computer and Communications Security.*

[12] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. 2018. Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. In *ACM conference on computer and communications security.* 619–633.

[13] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics.* 249–256.

[14] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning.* Vol. 1. MIT press Cambridge.

[15] Jihun Hamm, Adam C Champion, Guoxing Chen, Mikhail Belkin, and Dong Xuan. 2015. Crowd-ml: A privacy-preserving learning framework for a crowd of smart devices. In *IEEE International Conference on Distributed Computing Systems.*

[16] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling Up End-to-end Speech Recognition. *CoRR* abs/1412.5567 (2014). arXiv:1412.5567 http://arxiv.org/abs/1412.5567

[17] Johann Hauswald, Thomas Manville, Qi Zheng, Ronald Dreslinski, Chaitali Chakrabarti, and Trevor Mudge. 2014. A hybrid approach to offloading mobile image classification. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 8375–8379.

[18] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. 2017. LOGAN: evaluating privacy leakage of generative models using generative adversarial networks. *arXiv preprint arXiv:1705.07663* (2017).

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[20] Zecheng He, Aswin Raghavan, Guangyuan Hu, Sek Chai, and Ruby Lee. 2019. Power-Grid Controller Anomaly Detection with Enhanced Temporal Deep Learning. In *18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications.*

[21] Zecheng He, Tianwei Zhang, and Ruby Lee. 2019. Sensitive-Sample Fingerprinting of Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 4729–4737.

[22] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *ACM Conference on Computer and Communications Security.*

[23] Weizhe Hua, Zhiru Zhang, and G Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *ACM/ESDA/IEEE Design Automation Conference.*

[24] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving Machine Learning as a Service. *arXiv preprint arXiv:1803.05961* (2018).

[25] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *Acm Sigplan Notices* 52, 4 (2017), 615–629.

[26] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[27] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. 2018. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In *IEEE International Conference on Advanced Video and Signal Based Surveillance.*

[28] Yann Le Cun, LD Jackel, B Boser, JS Denker, HP Graf, I Guyon, D Henderson, RE Howard, and W Hubbard. 1989. Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning. *IEEE Communications Magazine* 27, 11 (1989), 41–46.

[29] Kin Sum Liu, Bo Li, and Jie Gao. 2018. Generative Model: Membership Attack, Generalization and Diversity. *arXiv preprint arXiv:1805.09898* (2018).

[30] Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A Gunter, and Kai Chen. 2018. Understanding Membership Inferences on Well-Generalized Learning Models. *arXiv preprint arXiv:1802.04889* (2018).

[31] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *CoRR* abs/1508.04025 (2015). arXiv:1508.04025 http://arxiv.org/abs/1508.04025

[32] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *IEEE Symposium on Security and Privacy.*

[33] Seong Joon Oh, Max Augustin, Mario Fritz, and Bernt Schiele. 2018. Towards reverse-engineering black-box neural networks. In *INternational Conference on Learning Representations.*

[34] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors.. In *USENIX Security Symposium.*

[35] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.

[36] Frank Rosenblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological review* 65, 6 (1958), 386.

[37] Leonid I Rudin, Stanley Osher, and Emad Fatemi. 1992. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena* 60, 1-4 (1992), 259–268.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning Representations by Back-propagating Errors. *nature* 323, 6088 (1986), 533.

[39] Ahmed Salem, Yang Zhang, Mathias Humbert, Mario Fritz, and Michael Backes. 2018. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *Network and Distributed System Security Symposium.*

[40] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *ACM conference on computer and communications security.*

[41] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models.

In *IEEE Symposium on Security and Privacy*.

[42] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine Learning Models that Remember Too Much. In *ACM Conference on Computer and Communications Security*.

[43] Surat Teerapittayanon, Bradley McDanel, and HT Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *IEEE International Conference on Distributed Computing Systems*.

[44] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs.. In *USENIX Security Symposium*.

[45] Aleksei Triastcyn and Boi Faltings. 2018. Generating Artificial Data for Private Deep Learning. *arXiv preprint arXiv:1803.03148* (2018).

[46] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing Hyperparameters in Machine Learning. In *IEEE Symposium on Security and Privacy*.

[47] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[48] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. 2018. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Annual Computer Security Applications Conference*.

[49] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *IEEE Computer Security Foundations Symposium*.

[50] Hongxu Yin, Zeyu Wang, and Niraj K Jha. 2018. A hierarchical inference model for internet-of-things. *IEEE Transactions on Multi-Scale Computing Systems* 4, 3 (2018), 260–271.

[51] Tianwei Zhang, Zecheng He, and Ruby B Lee. 2018. Privacy-preserving machine learning through data obfuscation. *arXiv preprint arXiv:1807.01860* (2018).

[52] Xinyang Zhang, Shouling Ji, and Ting Wang. 2018. Differentially Private Releasing via Deep Generative Model (Technical Report). *arXiv preprint arXiv:1801.01594* (2018).