The Pennsylvania State University
The Graduate School

# FISHER INFORMATION FOR PRIVATE TRAINING OF MACHINE LEARNING MODELS

A Thesis in
Computer Science and Engineering
by
Brandon Edmunds



Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2024

The thesis of Brandon Edmunds was reviewed and approved by the following:

Mehrdad Mahdavi
Assistant Professor of Computer Science and Engineering
Thesis Advisor

Kiwan Maeng
Assistant Professor of Computer Science and Engineering

Chita Das
Distinguished Professor of Computer Science and Engineering
Department Head of Computer Science and Engineering

# Abstract

Machine learning models leak information about the data they are trained on. Given a trained model, an adversary can potentially reconstruct the training data. Fisher Information leakage is a measure of the privacy leakage of the data used in training a machine learning model. A loss function incorporating Fisher Information leakage and pruning are used to train machine learning models that both perform well and leak minimal information about their training data through leveraging public data. Incorporating public data through Fisher Information is seen to yield increased privacy without significant decreases in utility compared to differentially private stochastic gradient descent.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

I would like to thank Mehrdad Mahdavi and Kiwan Maeng for their guidance, knowledge, and experience, which played an integral role in the successful completion of this project.

# Chapter 1 Introduction

As machine learning continues to become increasingly employed, the need for more data grows. As a part of encouraging more individuals to allow use of their data, steps are taken to help ensure an individual's data remains private, even when used to train a machine learning model. Given a machine learning model, and potentially other auxiliary information, adversaries can infer membership status of an individual in a data set used to train a machine learning model [24], or even reconstruct an individual's data [5]. To guarantee a bound on the success of any adversary, one of the most popular theoretical guarantees is provided by differential privacy [9, 22]. Differential privacy ensures that if a model is trained on a data set where one data point is removed from the data set (or the data point is modified), then the resulting model will be similar to the model that was trained on the data set including the original data point. As shown in [5], Fisher Information leakage (FIL) provides a tighter guarantee than differential privacy when considering reconstruction error, that is, an adversaries ability to reconstruct training data given information about the trained model. This motivates using the FIL as a part of a loss function, allowing for choosing a model that is both private and has good utility. To leverage FIL to impede training data reconstruction attacks, public data is used to train a machine learning model that is then used to update another machine learning model training on the private data. Additionally, pruning is considered as a method to gain privacy with minimal degradation of model utility.

## 1.1 Related Work

FIL has previously been incorporated into algorithms to train private models with good utility. [12] used FIL to train a model where its training data points have equivalent FIL. [20] used FIL to train a model robust to input reconstruction attacks. Public

data has been leveraged in many works to aid in the training of private models. For example, [8, 23] use public data to pre-train models before fine-tuning on private data, allowing for the resulting models to have better utility for some privacy guarantee. Additionally, [3, 7, 19, 23] demonstrate that public data can be leveraged past only pre-training the model. For example, [3] leverages public data in determining the gradients of the private data. To the best of the author's knowledge, all of such works devise strategies for differential privacy rather than FIL, and are additionally orthogonal to this work, allowing the methods to be used together. The primary algorithm used to train private models is DPSGD from [1], which will be leveraged in this work, as [11] showed that FIL can be calculated for this algorithm. Also, [25] showed that hyperparameter tuning leaks private information for differential privacy. Similarly, hyperparameter tuning leaks private information for FIL privacy, but this will not be considered in this work.

Similarly, there has been work leveraging pruning for increased privacy, as in [2,26,27]. In [2], public data is used to pre-train a model, then the model is pruned (or weights are frozen) before being fine-tuned on the private data, producing models with better utility for target differential privacy parameters. Similar pruning methodologies will be used in this work, except the privacy measure will be FIL rather than differential privacy.

## 1.2 Background

### 1.2.1 DPSGD

DPSGD from [1] is similar to mini-batch SGD (stochastic gradient descent), but does per sample gradient clipping, adds noise to the gradients summed over the batch, then takes the average as in

$$g(\boldsymbol{\theta}_t, B_t) = \frac{1}{|B_t|}\left(\sum_{\mathbf{z}\in B_t} \frac{\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{z})}{\max(1, \frac{\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{z})}{C})} + \mathcal{N}(\mathbf{0}, \sigma^2 C^2 I)\right)\Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \tag{1.1}$$

where $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda g(\boldsymbol{\theta}_t, B_t)$ is the update step from SGD, except the gradient is computed as in Equation (1.1). $\boldsymbol{\theta}_t$ is the model parameters at iteration $t$, $B_t$ is a randomly drawn batch of examples, and $\lambda$ is the learning rate.

### 1.2.2 Fisher Information Leakage

In the case of an unbiased attacker, given the model $\mathbf{h}$, every data point in data set $D$ except for one data point $\mathbf{z}$, and all algorithm hyperparameters for the algorithm used to generate $\mathbf{h}$, then, [11] showed that for an unbiased estimate $\hat{\mathbf{z}}(\mathbf{h}, D')$ of $\mathbf{z}$ where $D' = D \setminus \{\mathbf{z}\}$ and $\mathbf{z} \in D$ and $\mathbf{z} \in \mathbb{R}^d$

$$\mathbb{E}_{\mathbf{h}}\left[\frac{\|\hat{\mathbf{z}}(\mathbf{h}, D') - \mathbf{z}\|_2^2}{d}\right] \geq \frac{d}{\mathrm{Tr}(\mathcal{I}_{\mathbf{h}}(\mathbf{z}))} \tag{1.2}$$

where $\mathcal{I}_{\mathbf{h}}(\mathbf{z})$ is the Fisher Information matrix. Note that in the unbiased case, $\mathbb{E}[\hat{\mathbf{z}}(\mathbf{h}, D')] = \mathbf{z}$. Also, let $\mathrm{dFIL}_{\mathbf{h}}(\mathbf{z}) = \frac{\mathrm{Tr}(\mathcal{I}_{\mathbf{h}}(\mathbf{z}))}{d}$. Abusing notation, when $\mathbf{h}$ and $\mathbf{z}$ are clear from context, $\mathrm{dFIL}_{\mathbf{h}}(\mathbf{z})$ will be abbreviated as dFIL.

Additionally, there is a bound for a biased attacker shown by [21]. Given the input data distribution $\pi$, where $\mathbf{z} \sim \pi$, and the density function of $\pi$ with respect to Lebesgue measure, $f_\pi(\mathbf{z})$, assume that the regularity conditions of van Trees inequality from [10] are satisfied. That is, given $p(\mathbf{h}|\mathbf{z})$ is the probability density of $\mathbf{h}$ given $\mathbf{z}$, $p(\mathbf{h}|\mathbf{z})$ and $f_\pi$ are absolutely almost surely continuous and $f_\pi$ converges to 0 at the endpoints of $\mathbf{z}$ (readers are referred to [10, 21] for more precise conditions). The information theorist's Fisher Information from [4] of $\pi$ is $\mathcal{J}(f_\pi) = \mathbb{E}_\pi[\nabla_{\mathbf{z}} \log f_\pi(\mathbf{z})(\nabla_{\mathbf{z}} \log f_\pi(\mathbf{z}))^\top]$. Otherwise following the setting of Equation 1.2, but with a biased attacker

$$\mathbb{E}_\pi\left[\mathbb{E}_{\mathbf{h}}\left[\frac{\|\hat{\mathbf{z}}(\mathbf{h}, D') - \mathbf{z}\|_2^2}{d}\right]\right] \geq \frac{1}{\mathbb{E}_\pi[\mathrm{dFIL}_{\mathbf{h}}(\mathbf{z})] + \mathrm{Tr}(\mathcal{J}(f_\pi))/d}. \tag{1.3}$$

Additionally, [12] showed that the Fisher Information matrix of the Gaussian mechanism is

$$\mathcal{I}_{\mathbf{h}}(D) = \frac{1}{\sigma^2}(\nabla_{\mathcal{D}} f(\mathcal{D}))^\top \nabla_{\mathcal{D}} f(\mathcal{D})\Big|_{\mathcal{D}=D} \tag{1.4}$$

where $f(\mathcal{D})$ is a deterministic function of inputs $\mathcal{D}$, $\widetilde{f}(\mathcal{D}) = f(\mathcal{D}) + \mathbf{r}$, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, and $\mathbf{h} \sim \widetilde{f}(D)$. Here, $\widetilde{f}(\mathcal{D})$ resembles the Gaussian mechanism.

#### 1.2.2.1 Signal-to-noise-ratio

Taking $\frac{\mathrm{Tr}(\mathcal{I}_{\mathbf{h}}(\mathbf{z}))}{d} = \mathrm{dFIL}$ and applying Equation (1.4), solving for $\sigma^2$ gives $\sigma^2 = \frac{\|\nabla_{\boldsymbol{\zeta}} f(\boldsymbol{\zeta})\|_{\mathrm{F}}^2}{d \cdot \mathrm{dFIL}}\Big|_{\boldsymbol{\zeta}=\mathbf{z}}$.

Maximizing the signal-to-noise-ratio (SNR), $\frac{\|f(\mathbf{z})\|_2^2}{\sigma^2}$, is equivalent to minimizing $\frac{\|\nabla_{\boldsymbol{\zeta}} f(\boldsymbol{\zeta})\|_{\mathrm{F}}^2}{\|f(\boldsymbol{\zeta})\|_2^2}\Big|_{\boldsymbol{\zeta}=\mathbf{z}}$ for fixed dFIL, as done in [20].

# Chapter 2
# Fisher Information Minimization

To train a model that has both good utility and resistance to training data reconstruction attacks, Fisher Information is leveraged. Equation (1.2) shows that the mean squared error (MSE) of such a reconstruction attack is bounded and dependent on the trace of the Fisher Information matrix. Additionally, Equation (1.3) has the trace of the Fisher Information matrix in the denominator, implying that decreasing the trace makes the reconstruction task more difficult. These bounds motivate minimizing the trace of the Fisher Information matrix. Then, given some function of the data set, such as SGD (stochastic gradient descent), that outputs model parameters, when the Gaussian mechanism is applied to such an algorithm (treating SGD as deterministic in this case), that is, adding Gaussian noise to the outputted model parameters, applying the trace to Equation (1.4) allows for the following objective formulation

$$\boldsymbol{\phi}^* = \text{argmin}_{\boldsymbol{\phi}} L(\boldsymbol{\phi}, \boldsymbol{\theta}, \mathcal{D}) + \lambda \sum_{i=1}^{n} \|\nabla_{\boldsymbol{\zeta}_i} f_{\boldsymbol{\phi}}(\mathcal{D})\|_{\text{F}}^2 \Big|_{\mathcal{D}=D, \boldsymbol{\zeta}_i = \mathbf{z}_i} \tag{2.1}$$

where $L$ captures some objective other than privacy, $D \in \mathbb{R}^{n \times d}$, and $\lambda \in \mathbb{R}$. This formulation requires learning a function that, when given a data set, outputs parameters $\boldsymbol{\theta}$ for a machine learning model. Learning such a function requires its own machine learning model with parameters $\boldsymbol{\phi}$. The learned machine learning model that outputs model parameters is analogous to an algorithm that trains a private model with good utility. Training such a model is difficult since a single training data point is an entire data set. Additionally, the test data set, which is the private data, cannot be used to train the model since it leaks privacy, hence requiring multiple public data sets.

Instead, if the Gaussian mechanism is applied to the model output

$$\boldsymbol{\phi}^* = \text{argmin}_{\boldsymbol{\phi}} \sum_{i=1}^{n} L(\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\zeta}_i) + \lambda \sum_{j=1}^{d} \|\nabla_{\boldsymbol{\zeta}_{ij}} f_{\boldsymbol{\phi}}(\boldsymbol{\zeta}_i)\|_{\text{F}}^2 \Big|_{\boldsymbol{\zeta}=\mathbf{z}} \tag{2.2}$$

where $\mathbf{z}_{ij}$ refers to data point $i$ of $D$ at dimension $j$. If the privacy of the objective generalizes to data points other than the data points used in training, applying Equation (1.2), minimizing Equation (2.2) could make it so that when an adversary is given the model output of a non-training data point and all of the non-training data point except for 1 feature, the reconstruction error of input reconstruction attacks will be lower bounded. This idea was used in [14, 20].

Lastly, the Gaussian mechanism can be applied to the gradients. [11] showed that if a model $\mathbf{h}$ is obtained through DPSGD on data set $D$, applying the Gaussian mechanism to clipped gradients

$$\text{Tr}(\mathcal{I}_{\mathbf{h}}(\mathbf{z})) \leq \mathbb{E}\left[\sum_{t=1}^{T} \frac{q}{(q + (1-q)e^{-\epsilon})} \text{Tr}(\mathcal{I}_{\mathbf{g}}(\boldsymbol{\theta}_t, \mathbf{z}))\right] \tag{2.3}$$

where $T$ is the number of iterations of DPSGD, $\mathbf{g} \sim \widetilde{g}(\boldsymbol{\theta}, \mathbf{z})$, and $\widetilde{g}(\boldsymbol{\theta}, \mathbf{z}) = \bar{g}(\boldsymbol{\theta}, \mathbf{z}) + \mathbf{r}$, with $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, where $\bar{g}(\boldsymbol{\theta}, \mathbf{z})$ is the clipped gradient. $\epsilon = 1.115 \cdot 2\frac{\sqrt{2\log(\frac{1.25}{\delta})}}{\sigma}$ as in [11] (note that this holds for the cases used in this work), and, for batch size $B$, $q = \frac{B}{n}$, where $n$ is the number of data points in the data set. As made apparent by Equation (2.3), after clipping, $\text{Tr}(\mathcal{I}_{\mathbf{g}}(\boldsymbol{\theta}_t, \mathbf{z}))$ for the iteration is computed for a data point with the Gaussian mechanism being applied to the clipped gradient. $\text{Tr}(\mathcal{I}_{\mathbf{g}}(\boldsymbol{\theta}_t, \mathbf{z}))$ for the iteration is then added to the previous iterations to handle the privacy accounting. To clip, GeLU from [13] is used so that the clipping operation is differentiable, which is required for computing FIL. Using GeLU introduces the 1.115 overhead in computing $\epsilon$, as shown in [11]. To clip with GeLU, the operation becomes $\bar{g}(\boldsymbol{\theta}, \mathbf{z}) = \frac{g(\boldsymbol{\theta}, \mathbf{z})}{\text{GeLU}(\frac{\|g(\boldsymbol{\theta}, \mathbf{z})\|_2}{C} - 1) + 1}$, where $C$ is the clipping threshold, giving $\|\bar{g}(\boldsymbol{\theta}, \mathbf{z})\|_2 \leq 1.115C$.

So, instead of directly minimizing $\text{Tr}(\mathcal{I}_{\mathbf{h}}(\mathbf{z}))$ like in Equation (2.1), $\text{Tr}(\mathcal{I}_{\mathbf{g}}(\boldsymbol{\theta}, \mathbf{z}))$ can be minimized, giving

$$\boldsymbol{\phi}^* = \text{argmin}_{\boldsymbol{\phi}} \sum_{\{t,i\} \in S} L'(\boldsymbol{\phi}, \boldsymbol{\theta}_t, \mathbf{z}_i) + \lambda \|\nabla_{\boldsymbol{\zeta}_i} g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i)\|_{\text{F}}^2 \Big|_{\boldsymbol{\zeta}_i=\mathbf{z}_i} \tag{2.4}$$

where $\boldsymbol{\phi}$ parameterizes the new gradient $g_{\boldsymbol{\phi}}$ and $S \subseteq [T] \times [n]$ indexes the set of all pairs $(\boldsymbol{\theta}_t, \mathbf{z}_i)$ that would be input to $g_{\boldsymbol{\phi}}$ if $g_{\boldsymbol{\phi}}$ was used to train a model, where $T$ is the number of

iterations (not epochs) of SGD. A function for $L'$ could be $\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{z}_i) - g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \mathbf{z}_i)\|_2^2 \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$, where $L$ captures some objective other than privacy. Then, $g_{\boldsymbol{\phi}^*}$ can be used to train the model. Also, notice that $\boldsymbol{\theta}_t$ is not included in the privacy term's gradient since FIL is closed under post-processing, as shown in [12]. To solve Equation (2.4), the private data cannot be used, instead, public data can be used, as in Equation (2.1) and Equation (2.2). Additionally, $S$ is unknown, but can be approximated with public data. This motivates Algorithm 1. In Algorithm 1, first, the model is pre-trained on the public data. Next,

**Algorithm 1** Gradient Fisher Information Minimization

**Input:** Public data set $D_{public}$, private data set $D_{private}$, numbers of iterations $T_1, T_2, T_3$, batch sizes $B_1, B_2, B_3$, noise multiplier $\sigma$, clip threshold $C$, learning rates $\eta_1, \eta_2, \eta_3$, losses $L, L'$
Initialize $\boldsymbol{\theta}_0$ randomly
**for** $t_1$ in $[T_1]$ **do**
    Sample $B_{t_1} \subseteq D_{public}$, with $B_{t_1} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_{B_1}\}$
    $\boldsymbol{\theta}_{t_1} \leftarrow \boldsymbol{\theta}_{t_1-1} - \eta_1 \frac{1}{B_1} \sum_{i\in[B_1]} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{t_1-1}, \boldsymbol{\zeta}_i=\mathbf{z}_i}$
**end for**
Initialize $\boldsymbol{\phi}_0$ randomly
**for** $t_2$ in $[T_2]$ **do**
    **for** $t_3$ in $[T_3]$ **do**
        Sample $B_{t_3} \subseteq D_{public}$, with $B_{t_3} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_{B_3}\}$
        $\boldsymbol{\phi}_{(t_2-1)\times T_3+t_3} \leftarrow \boldsymbol{\phi}_{(t_2-1)\times T_3+t_3-1} - \eta_3 \frac{1}{B_3} \sum_{i\in[B_3]} \nabla_{\boldsymbol{\phi}} (L'(\boldsymbol{\phi}, \boldsymbol{\theta}_{T_1+t_2-1}, \mathbf{z}_i) + \lambda \|\nabla_{\boldsymbol{\zeta}_i} g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_{T_1+t_2-1}, \boldsymbol{\zeta}_i)\|_{\mathrm{F}}^2) \Big|_{\boldsymbol{\phi}=\boldsymbol{\phi}_{(t_2-1)\times T_3+t_3-1}, \boldsymbol{\zeta}_i=\mathbf{z}_i}$
    **end for**
    Sample $B_{t_2} \subseteq D_{private}$, with $B_{t_2} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_{B_2}\}$
    Sample $\mathbf{r}$ from $\mathcal{N}(\mathbf{0}, \sigma^2 C^2 I)$
    $\boldsymbol{\theta}_{T_1+t_2} \leftarrow \boldsymbol{\theta}_{T_1+t_2-1} - \eta_2 \frac{1}{B_2} (\mathbf{r} + \sum_{i\in[B_2]} \mathrm{CLIP}(g_{\boldsymbol{\phi}_{t_2\times T_3}}(\boldsymbol{\theta}_{T_1+t_2-1}, \mathbf{z}_i)))$
**end for**

the model being used as the gradient is trained using the public data, and the current main model weights (as FIL is closed under post-processing). After training the gradient model, the main model is updated using the private data and the gradient model, which has been trained on the current weights. The next time the gradient model is trained, it has been pre-trained on the previous weights.

## 2.1 Improvements

Instead of using Equation (2.4) directly, a modification is made to improve performance. Equation (2.4) can be rewritten as

$$\boldsymbol{\phi}^* = \text{argmin}_{\boldsymbol{\phi}} \sum_{\{t,i\}\in S} L'(\boldsymbol{\phi}, \boldsymbol{\theta}_t, \mathbf{z}_i) + \lambda \|\nabla_{\boldsymbol{\zeta}_i}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i) + g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i))\|_{\text{F}}^2 \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i=\mathbf{z}_i} \tag{2.5}$$

with $L' = \|g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \mathbf{z}_i)\|_2^2$. Equation (2.5) makes $g_{\boldsymbol{\phi}}$ perturb the existing gradient rather than output an entire gradient. The intention is to make it so that if $g_{\boldsymbol{\phi}}$ is poorly learnt, the original gradient is at least still present, so it may perform better.

To compute the Jacobian, the approximation method from [15] is employed by reformulating the trace of the Fisher Information matrix as a Jacobian-vector product as done in [11], giving

$$\begin{aligned}\text{Tr}(\mathcal{I}_{\mathbf{g}}(\boldsymbol{\theta}, \mathbf{z})) &= \frac{1}{\sigma^2}\text{Tr}((\nabla_{\boldsymbol{\zeta}}\bar{g}(\boldsymbol{\theta}, \boldsymbol{\zeta}))^\top \nabla_{\boldsymbol{\zeta}}\bar{g}(\boldsymbol{\theta}, \boldsymbol{\zeta}))\Big|_{\boldsymbol{\zeta}=\mathbf{z}} = \frac{1}{\sigma^2}\mathbb{E}[\mathbf{u}^\top(\nabla_{\boldsymbol{\zeta}}\bar{g}(\boldsymbol{\theta}, \boldsymbol{\zeta}))^\top \nabla_{\boldsymbol{\zeta}}\bar{g}(\boldsymbol{\theta}, \boldsymbol{\zeta})\mathbf{u}]\Big|_{\boldsymbol{\zeta}=\mathbf{z}} \\ &= \frac{1}{\sigma^2 s}\sum_{i=1}^{s} \mathbf{u}^\top(\nabla_{\boldsymbol{\zeta}}\bar{g}(\boldsymbol{\theta}, \boldsymbol{\zeta}))^\top \nabla_{\boldsymbol{\zeta}}\bar{g}(\boldsymbol{\theta}, \boldsymbol{\zeta})\mathbf{u}\Big|_{\boldsymbol{\zeta}=\mathbf{z}} = \frac{1}{\sigma^2 s}\sum_{i=1}^{s} \|\nabla_{\boldsymbol{\zeta}}\bar{g}(\boldsymbol{\theta}, \boldsymbol{\zeta})\mathbf{u}\|_2^2\Big|_{\boldsymbol{\zeta}=\mathbf{z}}\end{aligned} \tag{2.6}$$

where $\mathbb{E}[\mathbf{u}] = \mathbf{0}$, $\mathbb{E}[\mathbf{u}\mathbf{u}^\top] = I$, and $s$ is the number of samples for approximating the expectation. Specifically, $\mathbf{u}$ is chosen to be a standard normal random vector. As in [11], Equation (2.6) can be calculated using Jacobian-vector products so that the full Jacobian does not need to be instantiated.

To take advantage of the subsampling factor in Equation (2.3), the base algorithm considered is DPSGD, with the modification that the gradient computed (which is then clipped and noise is added to) is the new gradient that minimizes Equation (2.5). Since FIL is computed on the clipped gradient, Equation (2.5) is modified to

$$\boldsymbol{\phi}^* = \text{argmin}_{\boldsymbol{\phi}} \sum_{\{t,i\}\in S} L'(\boldsymbol{\phi}, \boldsymbol{\theta}_t, \mathbf{z}_i) + \lambda \|\nabla_{\boldsymbol{\zeta}_i}\text{CLIP}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i) + g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i))\|_{\text{F}}^2 \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i=\mathbf{z}_i}. \tag{2.7}$$

Additionally, as done in [20], the signal-to-noise-ratio (SNR) is used, giving

$$\boldsymbol{\phi}^* = \text{argmin}_{\boldsymbol{\phi}} \sum_{\{t,i\}\in S} L'(\boldsymbol{\phi}, \boldsymbol{\theta}_t, \mathbf{z}_i) + \lambda \frac{\|\nabla_{\boldsymbol{\zeta}_i}\text{CLIP}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i) + g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i))\|_{\text{F}}^2}{\|\text{CLIP}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i) + g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i))\|_2^2} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i=\mathbf{z}_i}. \tag{2.8}$$

Also, in an attempt to make learning the original gradient function easier, the original

gradient is input to the model, giving

$$\boldsymbol{\phi}^* = \text{argmin}_{\boldsymbol{\phi}} \sum_{\{t,i\}\in S} L'(\boldsymbol{\phi}, \boldsymbol{\theta}_t, \mathbf{z}_i) + \lambda \frac{\|\nabla_{\boldsymbol{\zeta}_i} \text{CLIP}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i) + g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i, \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i)))\|_{\text{F}}^2}{\|\text{CLIP}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i) + g_{\boldsymbol{\phi}}(\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i, \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{\zeta}_i)))\|_2^2}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t, \boldsymbol{\zeta}_i=\mathbf{z}_i}. \quad (2.9)$$

Auto-differentiation is used to take care of the additional dependence on the input.

## 2.2 Pruning

As made apparent in Equation (2.4), pruning the gradient increases FIL privacy. Additionally, as shown in Equation (2.9), the learned gradient model requires the parameters and corresponding gradients of the main model as inputs, and the output has the same dimension as the number of parameters, hence pruning gradients decreases the dimension of the learning problem since constants are not useful features and the model does not need to output the gradient for all the parameters. When pruning based on a pre-trained model on public data, the learned gradient model can be initialized using only the parameters and gradients that are non-constant, decreasing the size of the model.

# Chapter 3
# Experiments

The code is based off the code from [11] and is based in JAX [6]. The code can be found at `https://github.com/brandonedmunds2/Thesis`. Experiments are done using the MNIST [18] and CIFAR-10 [17] data sets. MNIST has 60,000 training images and 10,000 test images, each with 28x28 features. The classes are the 10 digits. CIFAR-10 has 50,000 training images and 10,000 test images, which are color images of size 32x32. The 10 classes are airplane, bird, car, cat, dog, frog, deer, horse, ship, and truck.

The following applies to all experiments. To approximate the expectation in Equation (2.6), 2 samples are taken. Whenever error bars (standard deviations) are reported, experiments are run 3 times, taking the average of the runs for the reported metrics (accuracy and dFIL). For pruning, the associated parameters are frozen rather than pruned. Pruning is only done in addition to DPSGD. "FIM" indicates that pruning was done and that a new gradient function was learned. Images are normalized between 0 and 1. Additionally, CIFAR-10 is standardized. The model is first pre-trained on the public data set, then fine-tuned on the private data set. Magnitude pruning indicates that the parameters with the smallest magnitudes are frozen immediately after pre-training. The pruning mask is not changed during training. To measure privacy, the label is considered public as done in [11]. All experiments are done using $10^{-9}$ for delta. The model used for FIM is a convolutional neural network (CNN) for all MNIST experiments and a multilayer-perceptron for CIFAR-10. To train the classification models, entropy loss is used. Additional details and hyperparameters can be found in the Appendix.

## 3.1 Verifying Decreased dFIL

The effect of varied amounts of public data is considered in Figure 3.1. Increasing the amount of public data improves the performance of FIM. Using smaller amounts of public

data does not work well with the given hyperparameters, as the accuracy of FIM is lower than the pre-train accuracy. For 50% public data, FIM works well, as shown by the values from Figure 3.1 which are presented in Table 3.1. Table 3.2 uses a smaller amount of public data more successfully. Additionally, the effect of different values of $\lambda$ is shown in Figure 3.2, which indicates that increasing the value of lambda generally increases the privacy and decreases the accuracy.

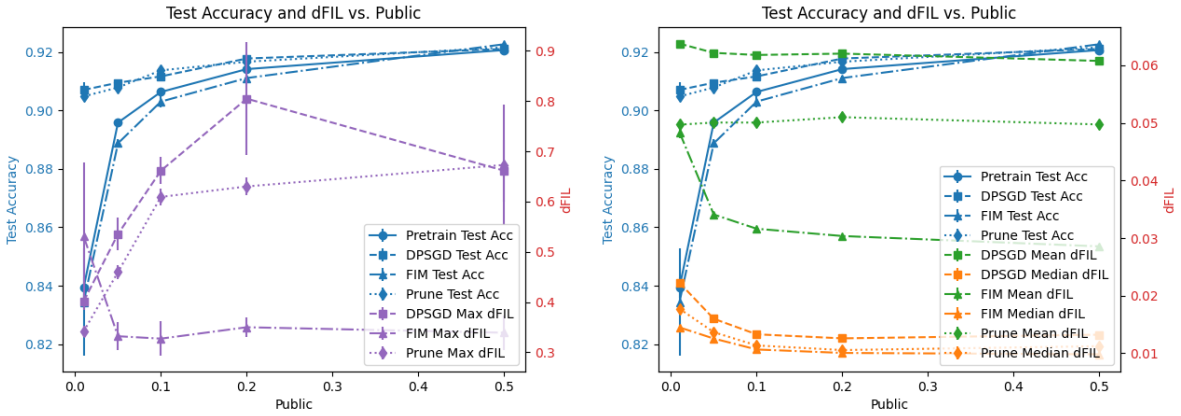


**Figure 3.1.** The plots show the test accuracy and dFIL taken across all the private data points for configurations with different fractions of the MNIST data set used as public data. A linear model is used. The other hyperparameters are equivalent for all configurations. The values of $\epsilon$ for public data of 0.5, 0.2, 0.1, 0.05, and 0.01 are 20.5318, 18.4916, 18.0058, 17.8031, and 17.6615, respectively.

**Table 3.1.** 50% Public Data MNIST Linear Model Values from Figure 3.1; $\epsilon = 20.5318$; Mean (STD)

| Method | Accuracy | Max dFIL | Mean dFIL | Median dFIL |
|---|---|---|---|---|
| Pretrain | 0.9208 (0.0005) | - | - | - |
| SGD | 0.9226 (0.0005) | - | - | - |
| DPSGD | 0.9211 (0.0005) | 0.6615 (0.0498) | 0.0608 (0.0002) | 0.0132 (0.0002) |
| Prune | 0.9217 (0.0002) | 0.6731 (0.0980) | 0.0497 (0.0002) | 0.0112 (0.0002) |
| FIM | 0.9228 (0.0005) | 0.3391 (0.0284) | 0.0286 (0.0001) | 0.0098 (0.0002) |

Additionally, FIM works for other models. The results in Table 3.2 are done using a convolutional neural network (CNN) with a smaller value of $\epsilon$. Note that while the maximum dFIL increases, the mean decreases, meaning that the total privacy gets better at the cost of making some data points have decreased privacy.

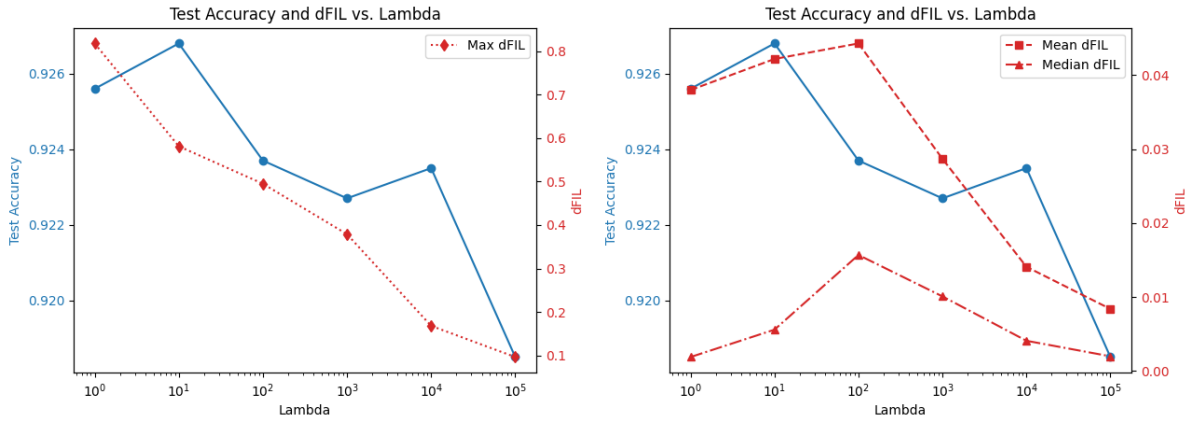


**Figure 3.2.** The plots show the test accuracy and dFIL taken across all the private data points for configurations with different values of lambda with a public data fraction of 0.5. A linear model is used. The other hyperparameters are equivalent for all configurations, following the hyperparameters from Figure 3.1; $\epsilon = 20.5318$.

**Table 3.2.** 5% Public Data MNIST CNN; $\epsilon = 0.6310$; Mean (STD)

| Method | Accuracy | Max dFIL | Mean dFIL | Median dFIL |
|---|---|---|---|---|
| Pretrain | 0.9524 (0.0038) | - | - | - |
| SGD | 0.9717 (0.0007) | - | - | - |
| DPSGD | 0.9597 (0.0007) | 0.1961 (0.0158) | 0.0106 (0.0001) | 0.0034 (0.0001) |
| Prune | 0.9618 (0.0011) | 0.1865 (0.0068) | 0.0093 (0.0004) | 0.0026 (0.0002) |
| FIM | 0.9705 (0.0014) | 0.3211 (0.0405) | 0.0057 (0.0002) | 0.0007 (0.0001) |

Lastly, CIFAR-10 is considered using a linear model with entropy loss for classification. The results are shown in Table 3.3. As shown in Table 3.3, the effectiveness of FIM is more limited with CIFAR-10.

## 3.2 Reconstruction Attack

To verify that using FIM defends against reconstruction attacks, the attack from [5] is considered. The attacker uses a fixed data set, $D_{fixed}$, along with disjoint sample data points from the same distribution as the data points in $D_{fixed}$ to train many machine learning models. The attack trains on $D_{fixed} \cup \{\mathbf{z}_i\}$ to get model $\boldsymbol{\theta}_i$, creating a new data set $\{(\boldsymbol{\theta}_i, \mathbf{z}_i)\}_{i=1}^{m}$. Next, the attacker model is trained on this new data set to predict $\mathbf{z}$ given $\boldsymbol{\theta}$. In this case, since the label, $\mathbf{y}$, is being treated as public, the input is the model and label together, $(\boldsymbol{\theta}, \mathbf{y})$, and the output is the image, $\mathbf{x}$. To train the attacker model,

**Table 3.3.** 50% Public Data CIFAR-10 Linear Model; $\epsilon = 3.9551$; Mean (STD)

| Method | Accuracy | Max dFIL | Mean dFIL | Median dFIL |
|---|---|---|---|---|
| Pretrain | 0.3751 (0.0038) | - | - | - |
| SGD | 0.3799 (0.0013) | - | - | - |
| DPSGD | 0.3992 (0.0013) | 0.0758 (0.0010) | 0.0165 (0.0000) | 0.0156 (0.0000) |
| Prune | 0.3977 (0.0021) | 0.0633 (0.0006) | 0.0166 (0.0000) | 0.0156 (0.0000) |
| FIM | 0.4011 (0.0008) | 0.0540 (0.0014) | 0.0151 (0.0000) | 0.0149 (0.0000) |

the inputs are first standardized.

As shown in [5], there are some methods of training that make the attacker unsuccessful, and due to the large number of models being trained, compute is limited, motivating the following modifications. First, the model initialization is the same (identical random seed) across all models used to train the attacker model, including the models for FIM. Additionally, pre-training is not done. Linear models with entropy loss are trained on binary MNIST with the CNN model used for the new gradient function for FIM. The attacker model is a MLP (multilayer perceptron). The reported metrics include MSE (mean squared error), which dFIL is directly related to, and LPIPS, which was shown to capture perceptual image similarity in [28].

As indicated in Table 3.4, FIM is able to perform well for binary MNIST, yielding improved accuracy and privacy. In Table 3.5, the attack was not able to reconstruct the images when FIM was used, even with a high $\epsilon$ which allowed DPSGD images to be reconstructed. The associated images are shown in Figure 3.3. Note that since the labels are given as inputs to the attacker, perfect privacy corresponds to an average of the training images for the respective classes.

**Table 3.4.** Binary MNIST Linear Model Reconstruction Attack dFILs; $\epsilon = 1571.4291$; Mean (STD)

| Method | Accuracy | Max dFIL | Mean dFIL | Median dFIL |
|---|---|---|---|---|
| SGD | 0.9964 (0.0005) | - | - | - |
| DPSGD | 0.9951 (0.0011) | 4.1469 (0.0622) | 1.4096 (0.1710) | 1.4521 (0.1506) |
| Prune | 0.9924 (0.0031) | 2.9422 (0.2106) | 1.0357 (0.0401) | 1.0180 (0.0670) |
| FIM | 0.9975 (0.0008) | 2.3875 (0.0286) | 0.5713 (0.0217) | 0.5339 (0.0392) |

**Table 3.5.** Binary MNIST Linear Model Reconstruction Attack MSE and LPIPS; $\epsilon = 1571.4291$

| Method | MSE | LPIPS |
|---|---|---|
| SGD | 0.0109 | 0.0179 |
| DPSGD | 0.0376 | 0.1020 |
| Prune | 0.0396 | 0.1261 |
| FIM | 0.0449 | 0.1630 |
| Perfect Privacy | 0.0439 | 0.1600 |

**Figure 3.3.** The images from top to bottom follow Table 3.5. The top image is the target images to reconstruct, the next image is reconstruction without any privacy, followed by DPSGD, pruning, FIM, and lastly perfect privacy, which is reconstructed using only the labels.

### 3.2.1 Exploratory Data Analysis

To better characterize FIM, the average of the public gradients is computed and the cosine similarity is taken between these gradients and the perturbation outputted by the FIM model, yielding Figure 3.4. The high cosine similarity implies that FIM is learning the average gradient of the public data for the perturbation, which is studied in [19], and allows for a more efficient algorithm. Additional investigation of the behavior of FIM is present in the Appendix.

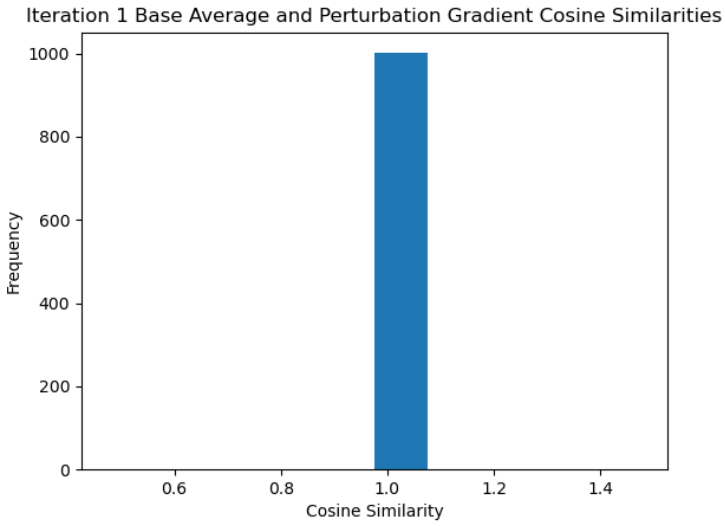


**Figure 3.4.** Binary MNIST Linear Model Reconstruction Attack Cosine Similarity of FIM Perturbation and Average Public Gradient For The First Iteration

# Chapter 4
# Conclusion

This work explored using Fisher Information leakage (FIL) as a part of a loss function to learn models with good utility and privacy by leveraging public data. Using FIL as a part of a loss function is seen to increase the privacy of the training data with minimal impact on model utility. Additionally, pruning, which also increases privacy with minimal impact on model utility, is shown to work well when paired with FIL.

Some limitations and subjects for future work include that FIL most readily applies to unbiased adversaries, whereas real-world adversaries are biased. Also, computing FIL is expensive, limiting the effectiveness of the method for larger models. Furthermore, for the best results for learning a new gradient, the representative model can be expensive to train. Additionally, the attack from [5] can more easily be prevented by techniques such as pre-training and using different activation functions, making a situation for using FIM in real-world applications unlikely due to the weakness of current attacks. Also, FIM is seen to calculate the average of the public gradients which can be more efficiently computed without training a machine learning model and has been studied in [19]. Lastly, learning new gradients requires additional knowledge about the data distribution, in this case in the form of public data, which is not always readily available.

# Appendix A
# Exploratory Data Analysis Continued

The experiments in this section all follow the same setting and hyperparameters as in Section 3.2 unless otherwise indicated. In the Figures, "Base" indicates the normal gradients that FIM perturbs, "Perturbation" indicates the prediction that the FIM model makes which gets added to the base gradient. "Final" indicates that gradient after adding the perturbation and after clipping, but before adding noise. Additionally, pruning is used in all cases for all gradients, hence "Base" and "Prune" are used interchangeably. The random seeds indicate different runs that the Figures are created from. The values of $\epsilon$ each iteration are 389.9752, 697.6610, 996.0350, 1294.4090, and 1571.4291, respectively, across all random seeds.

As indicated in the norm plots, such as Figure A.1, FIM encourages the gradients to be larger, but does not linearly scale the gradients as indicated in the cosine similarity plots, such as Figure A.3. Additionally, as indicated by comparing Figure A.1 and Figure A.8, increasing $\lambda$ increases the norms. Pairing this information with Figure 3.4 indicates that FIM predicts the average public gradient and scales by $\lambda$. FIM also decreases the variance of the most important principal components as shown in the PCA plots, particularly in Figure A.3. Lastly, the majority of the privacy is seen to be leaked in the first iteration, with less mean privacy typically being leaked per iteration, as shown in the accuracy and dFIL plots, such as Figure A.2. On the other hand, the maximum privacy scales linearly.

## A.1 Random Seed: 0

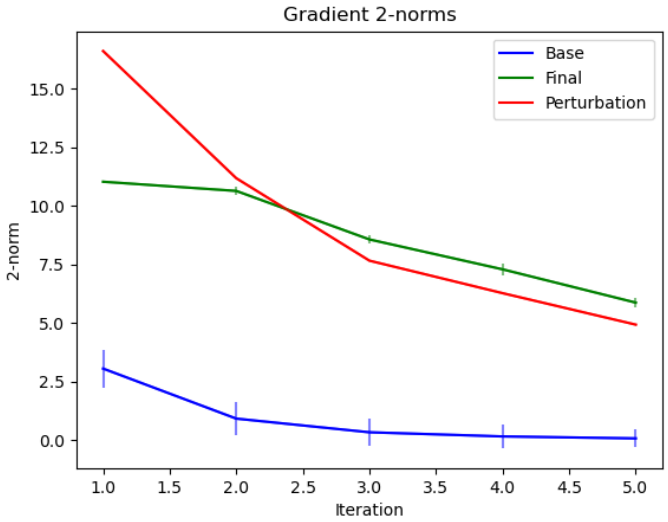


**Figure A.1.** Binary MNIST Linear Model Reconstruction Attack 2-Norms. Random Seed: 0

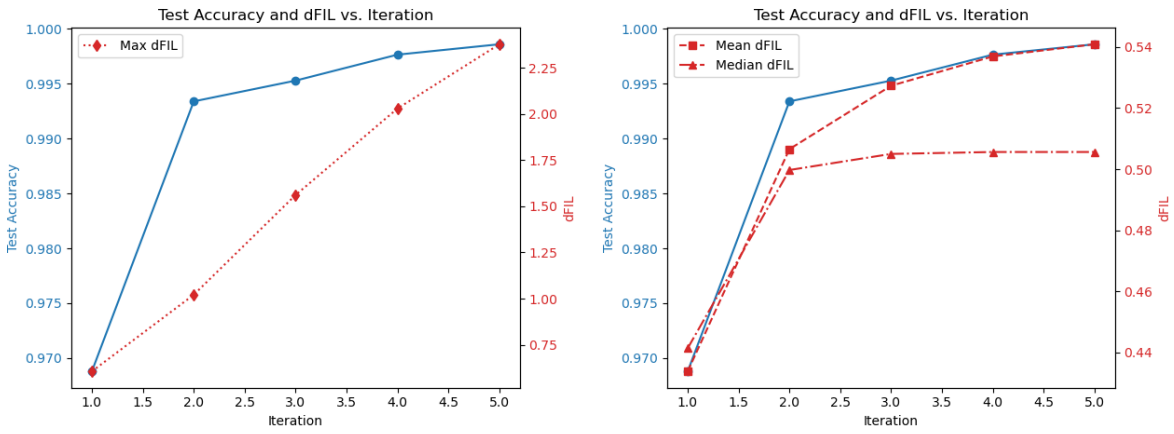


**Figure A.2.** Binary MNIST Linear Model Reconstruction Attack Accuracy and dFIL. Random Seed: 0

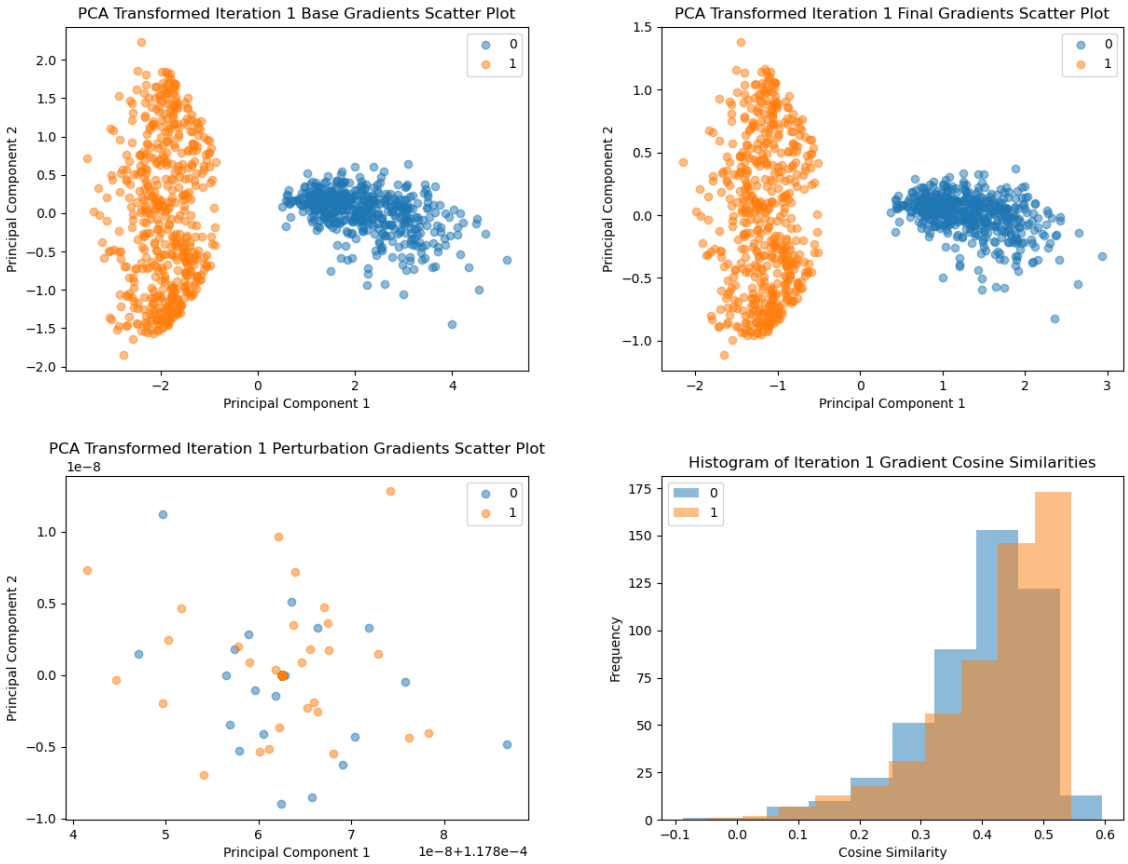


**Figure A.3.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 1; Random Seed: 0

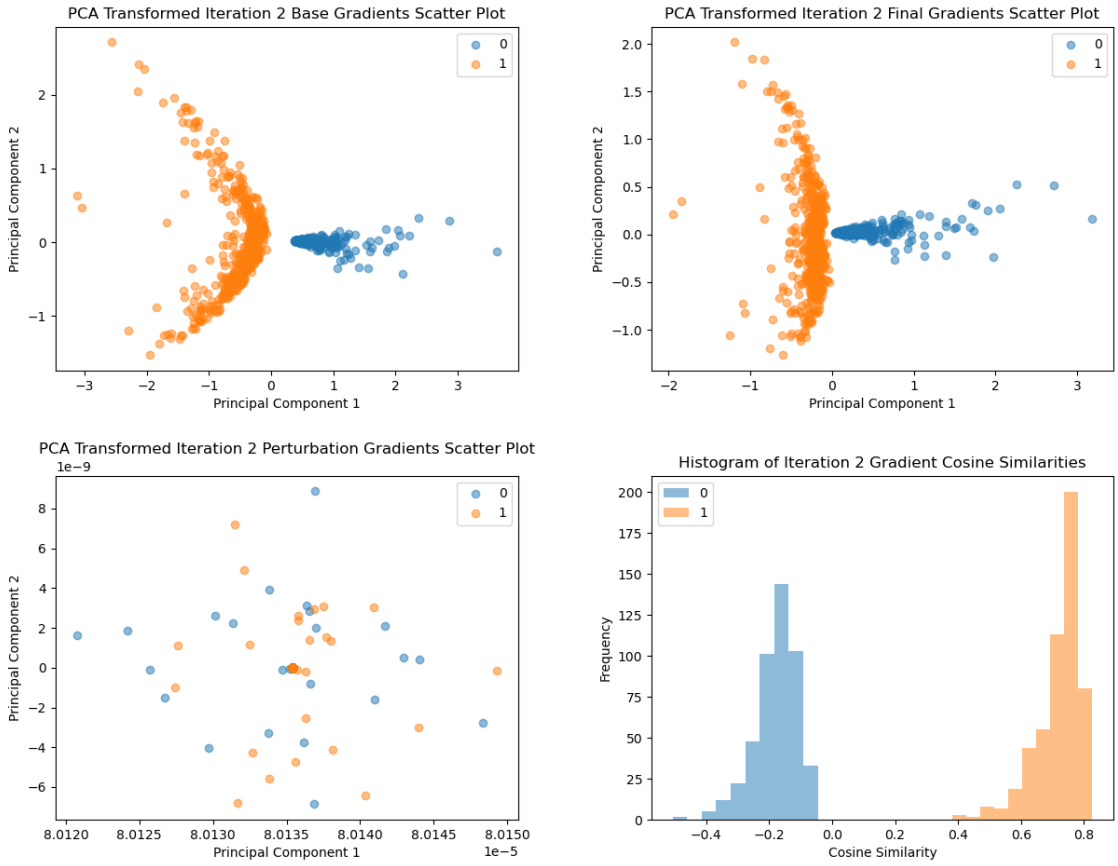


**Figure A.4.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 2; Random Seed: 0

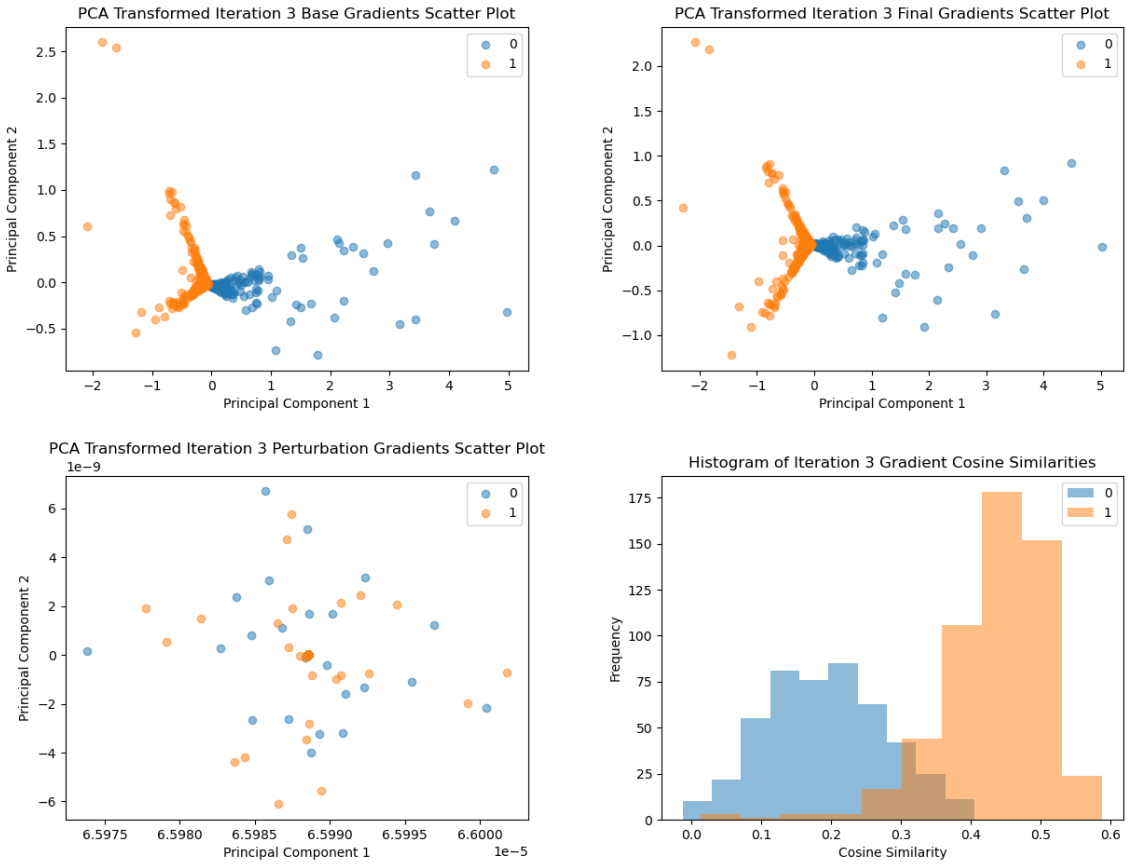


**Figure A.5.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 3; Random Seed: 0

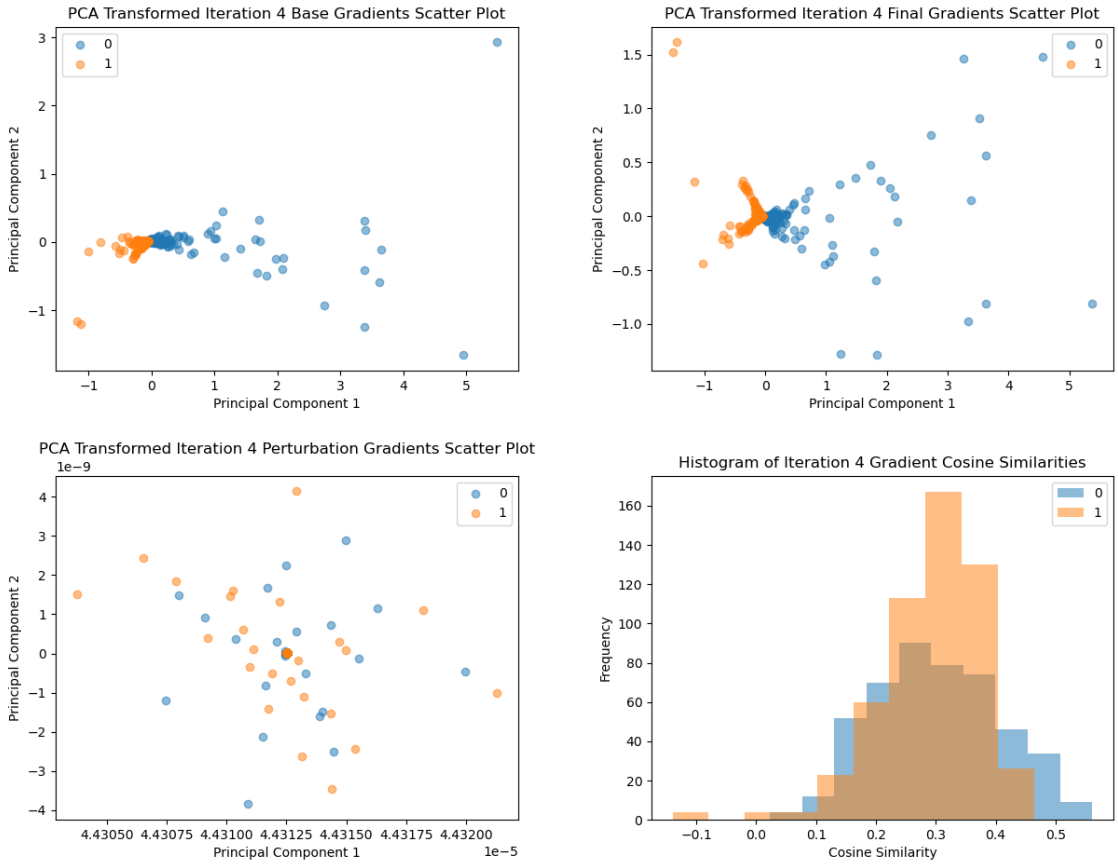


**Figure A.6.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 4; Random Seed: 0

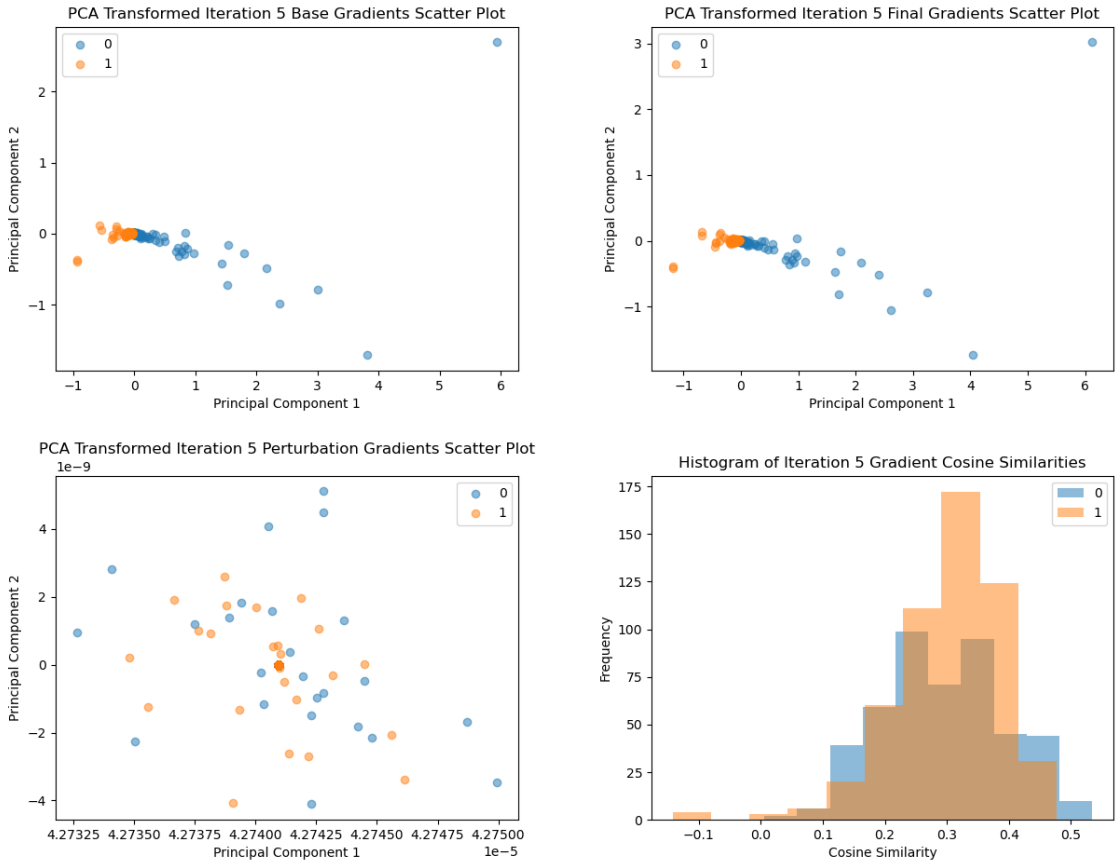


**Figure A.7.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 5; Random Seed: 0

### A.1.1 Increased $\lambda$; Random Seed: 0

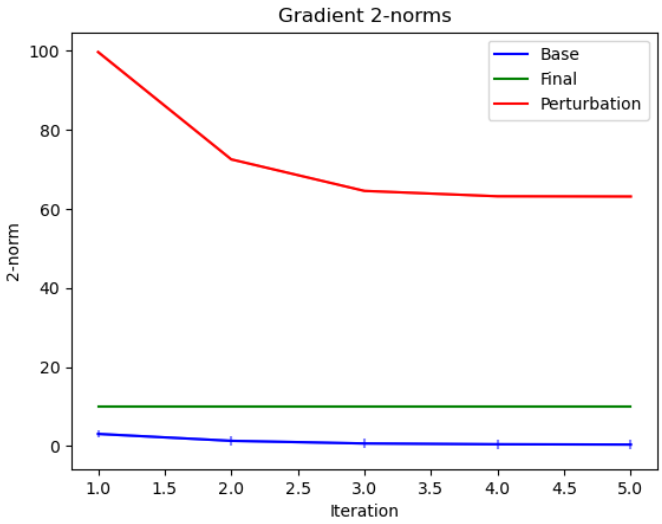


**Figure A.8.** Binary MNIST Linear Model Reconstruction Attack 2-Norms. $\lambda = 10^6$; Random Seed: 0

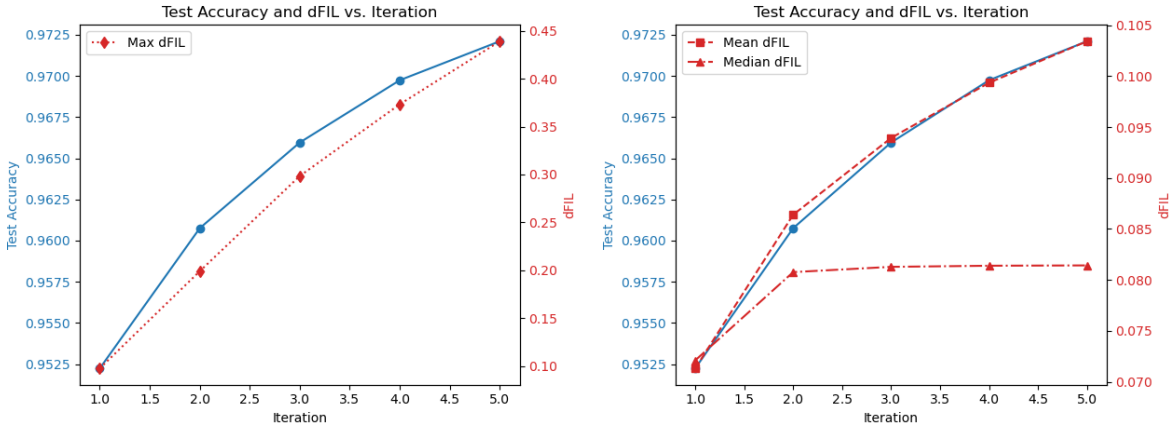


**Figure A.9.** Binary MNIST Linear Model Reconstruction Attack Accuracy and dFIL. $\lambda = 10^6$; Random Seed: 0


**Figure A.10.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 1; $\lambda = 10^6$; Random Seed: 0

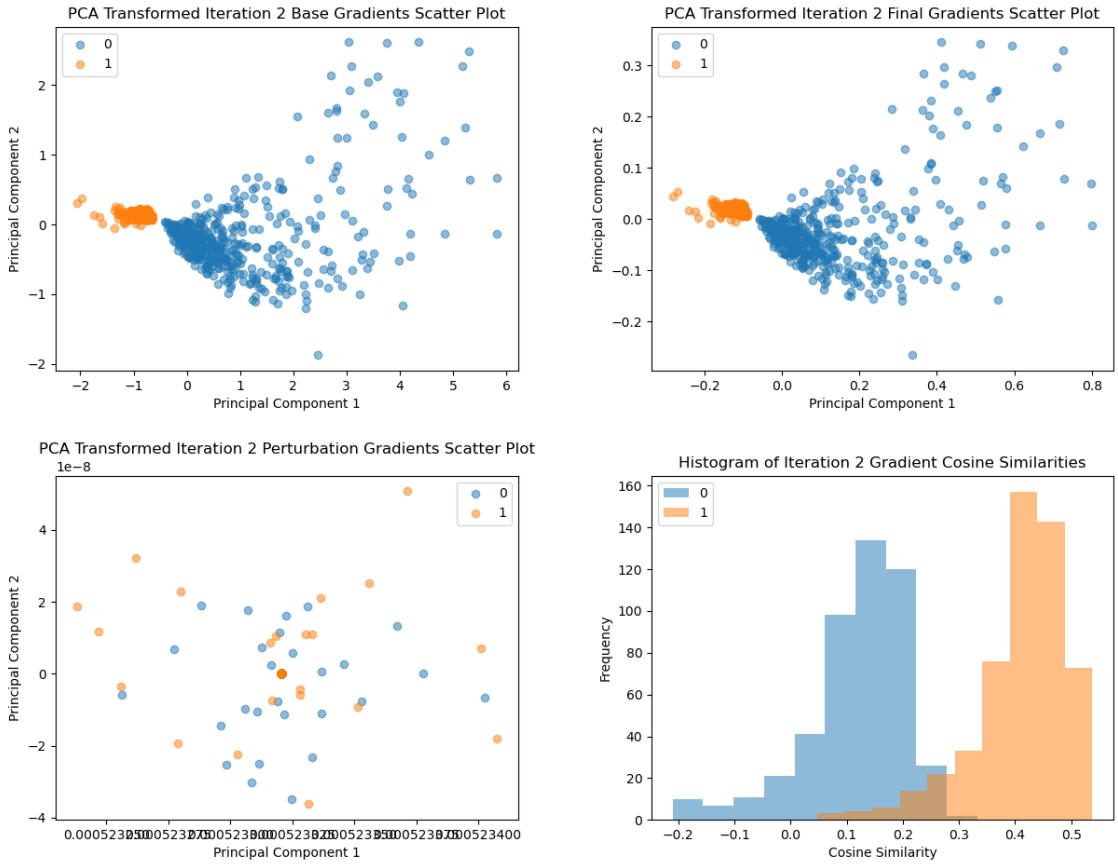


**Figure A.11.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 2; $\lambda = 10^6$; Random Seed: 0

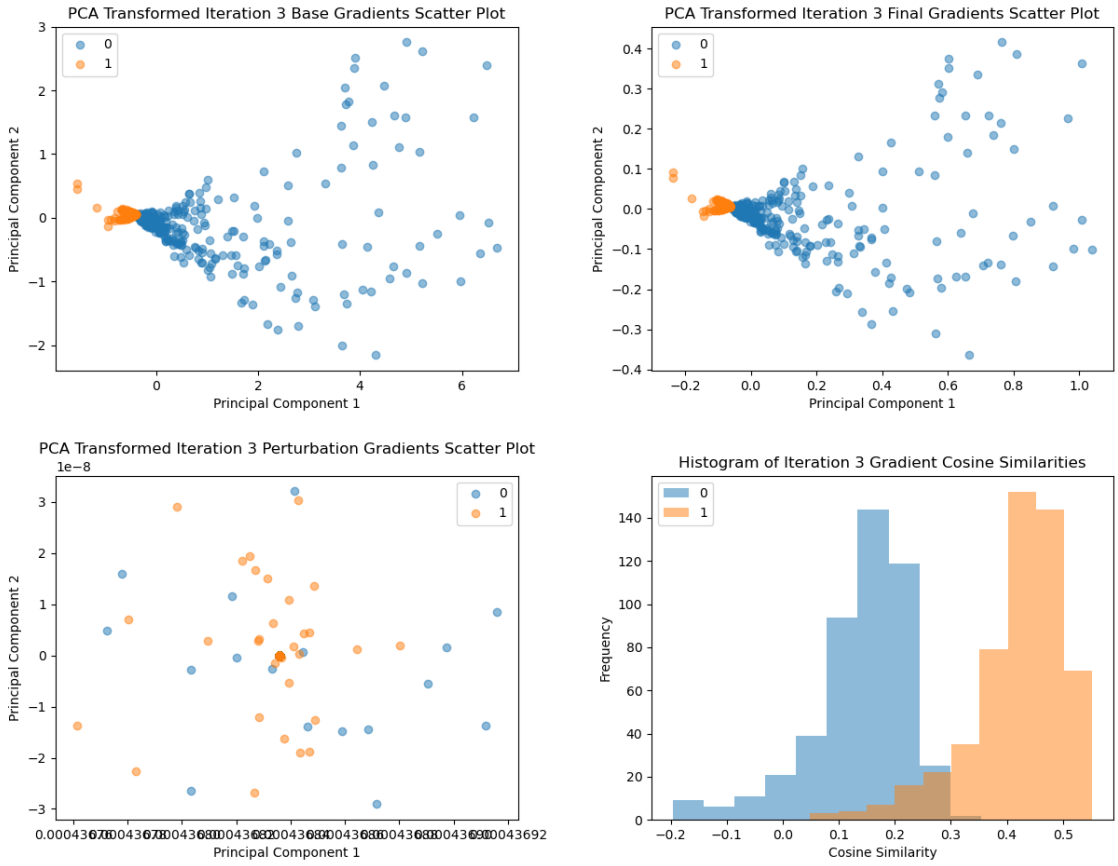


**Figure A.12.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 3; $\lambda = 10^6$; Random Seed: 0

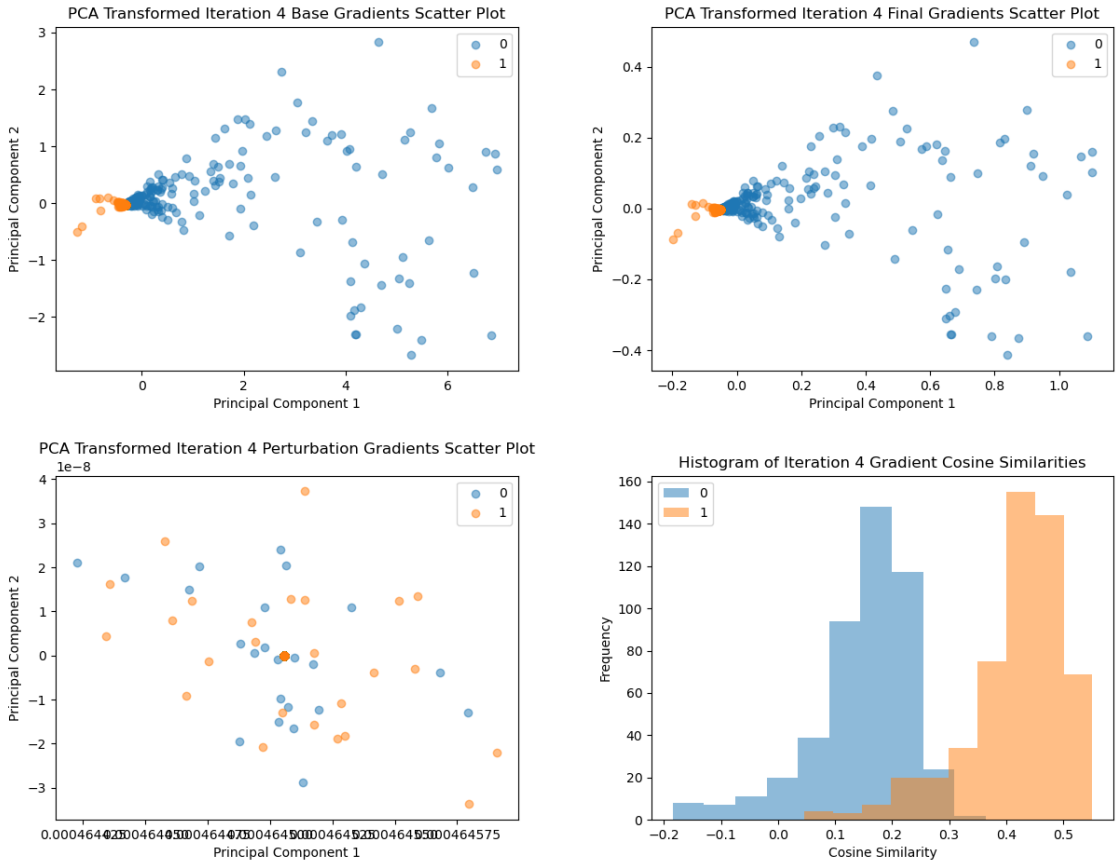


**Figure A.13.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 4; $\lambda = 10^6$; Random Seed: 0

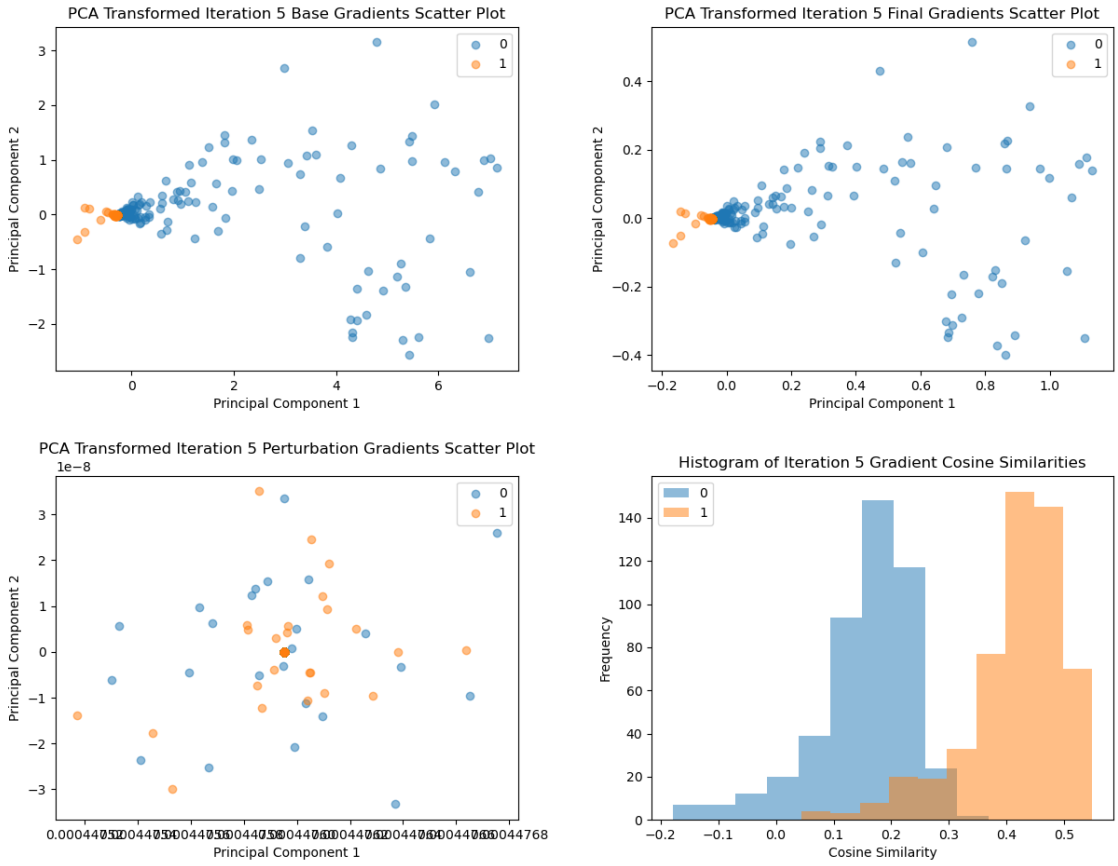


**Figure A.14.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 5; $\lambda = 10^6$; Random Seed: 0

## A.2 Random Seed: 1

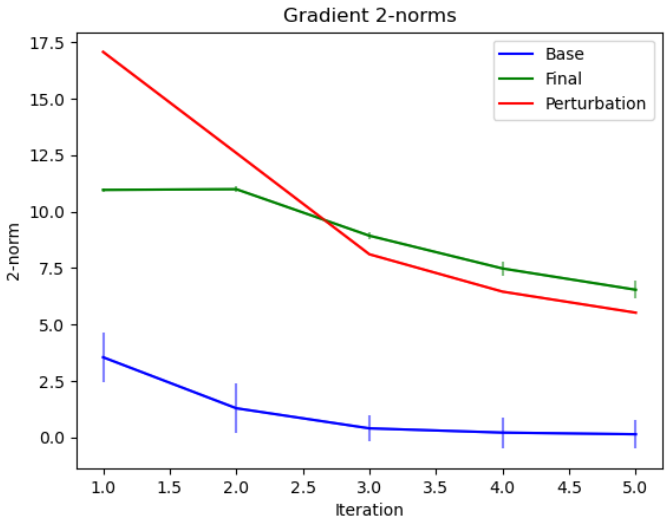


**Figure A.15.** Binary MNIST Linear Model Reconstruction Attack 2-Norms. Random Seed: 1

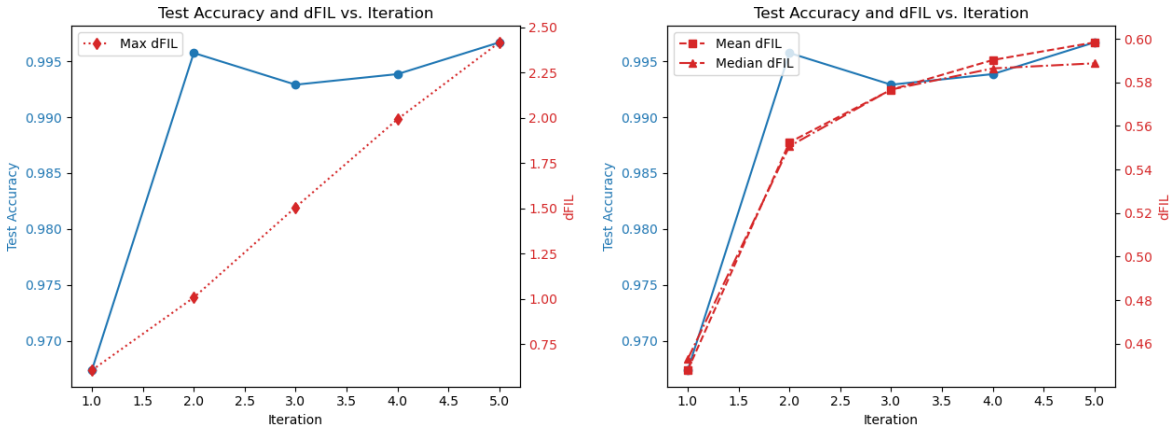


**Figure A.16.** Binary MNIST Linear Model Reconstruction Attack Accuracy and dFIL. Random Seed: 1

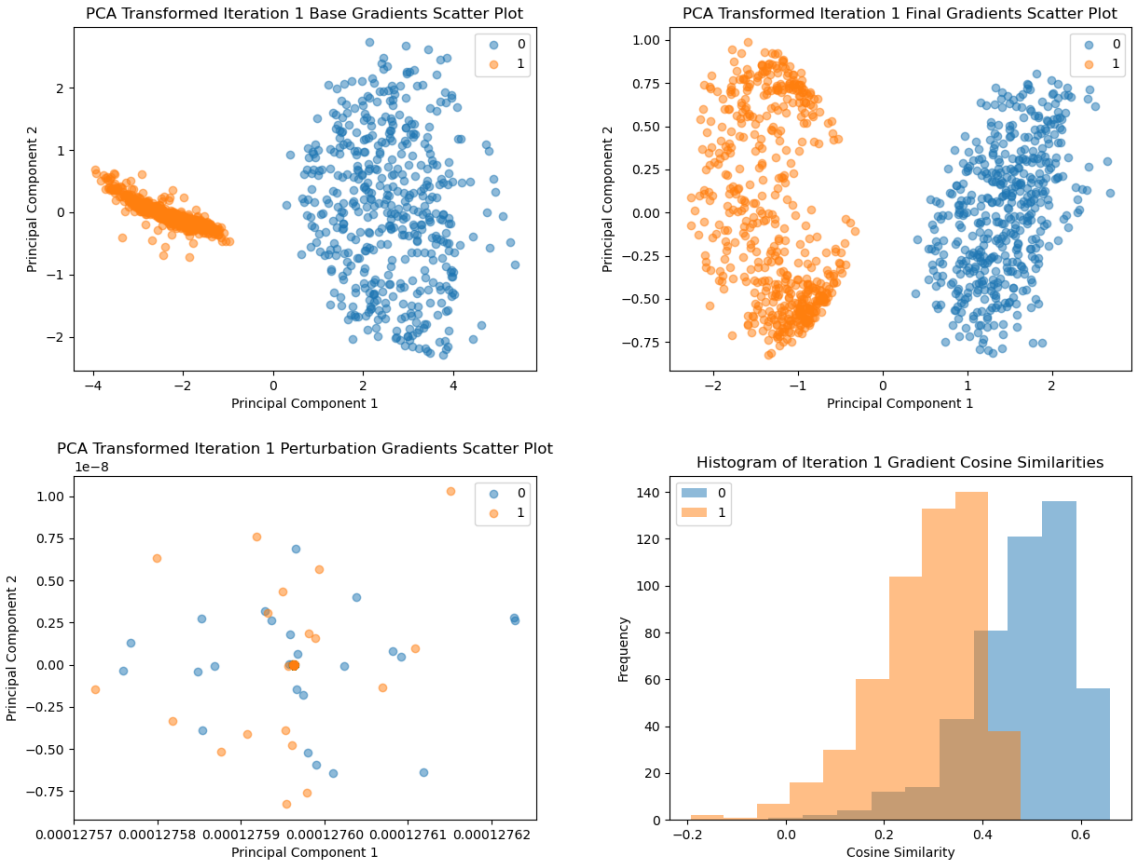


**Figure A.17.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 1; Random Seed: 1

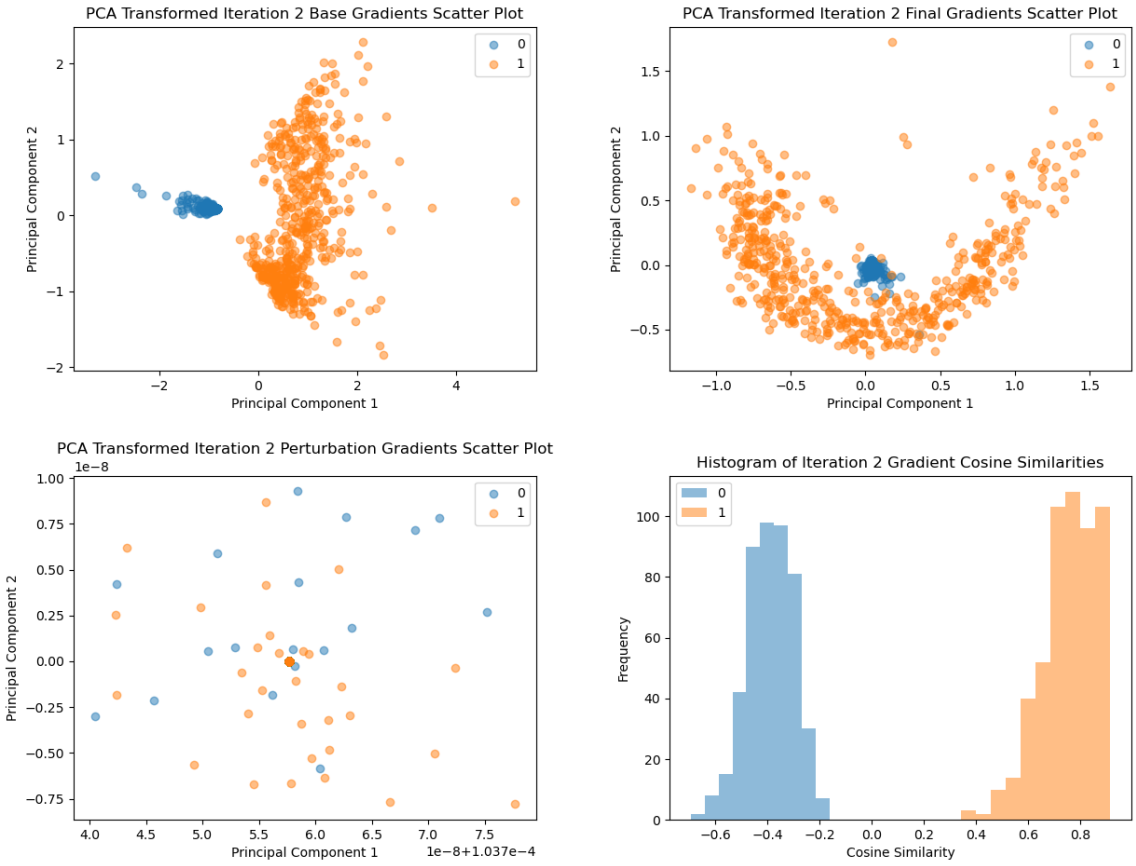


**Figure A.18.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 2; Random Seed: 1

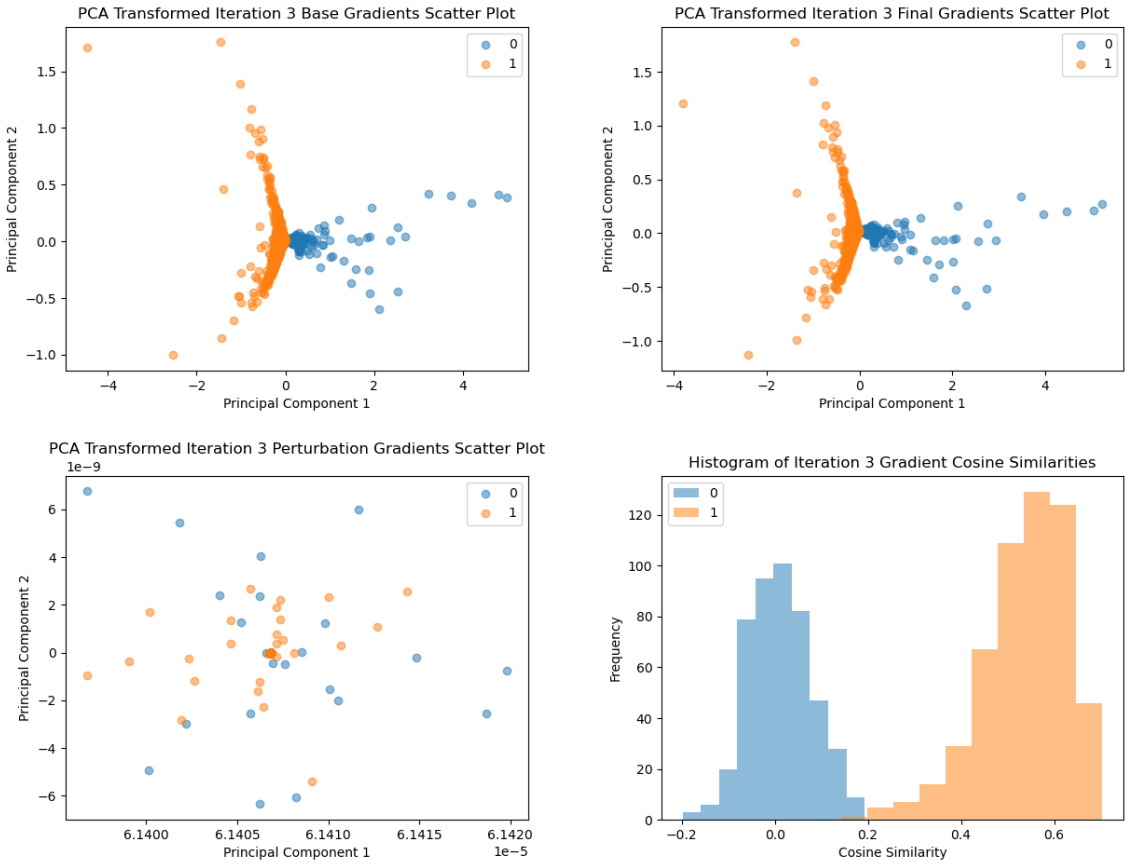


**Figure A.19.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 3; Random Seed: 1

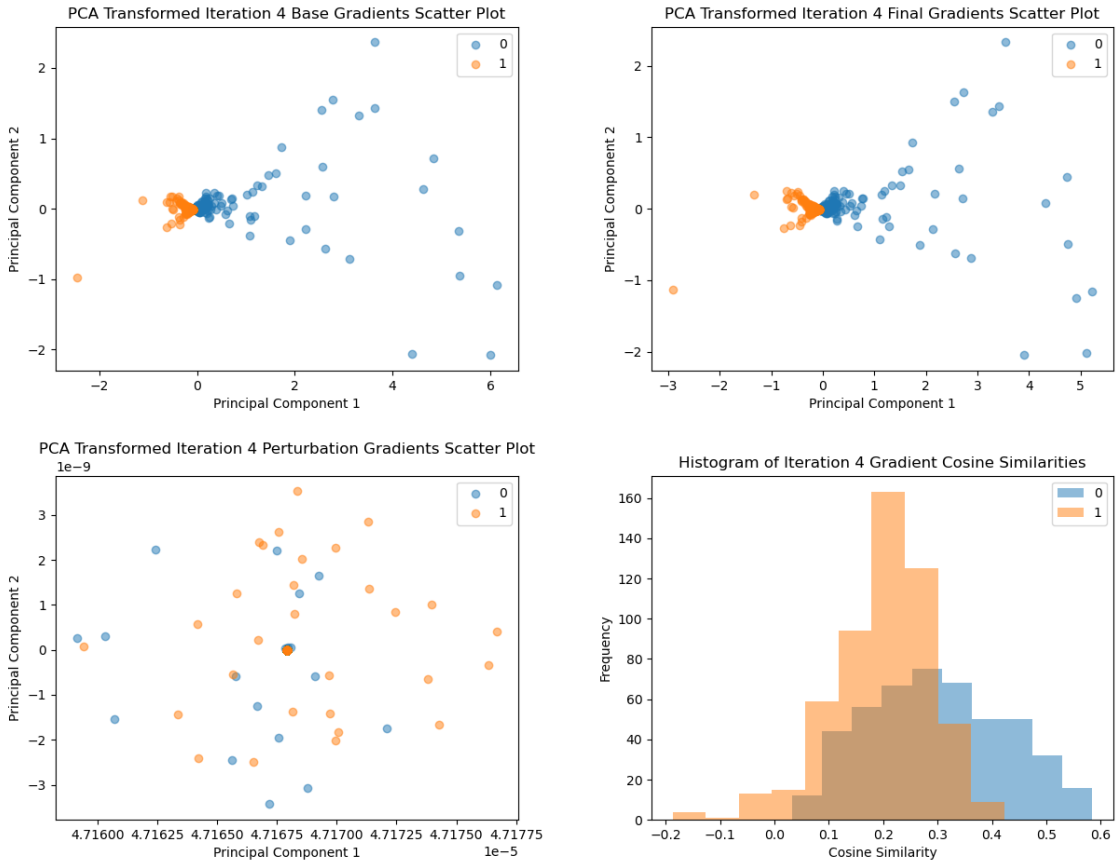


**Figure A.20.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 4; Random Seed: 1

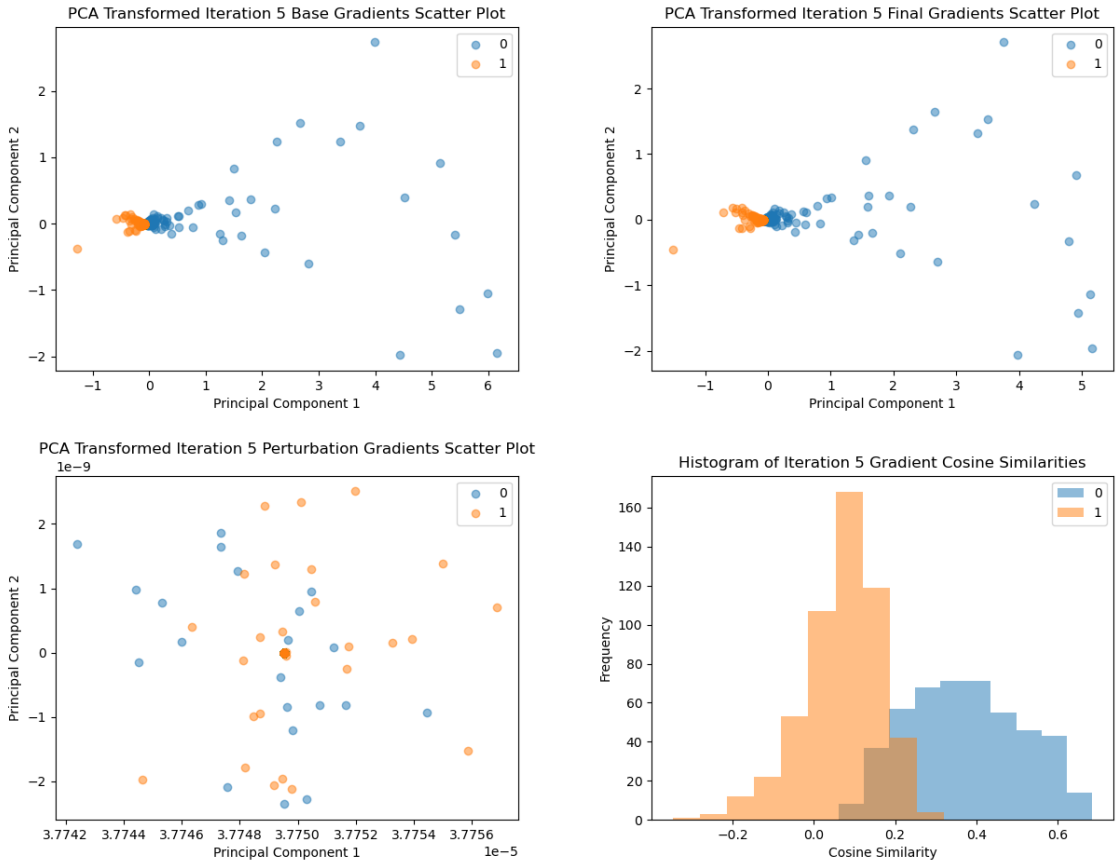


**Figure A.21.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 5; Random Seed: 1

## A.3 Random Seed: 2

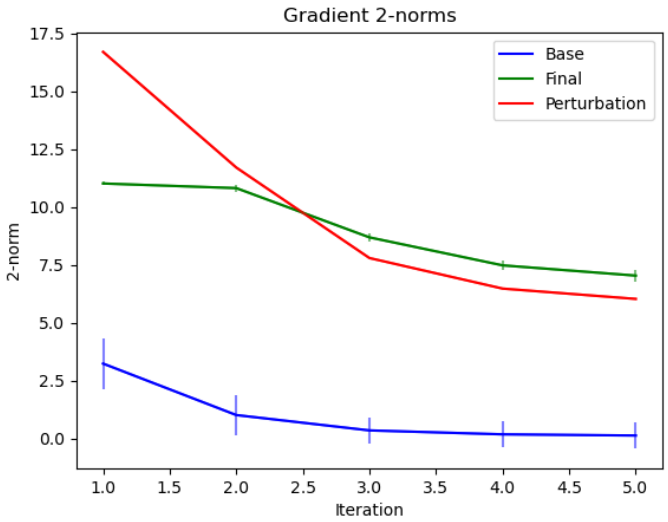


**Figure A.22.** Binary MNIST Linear Model Reconstruction Attack 2-Norms. Random Seed: 2

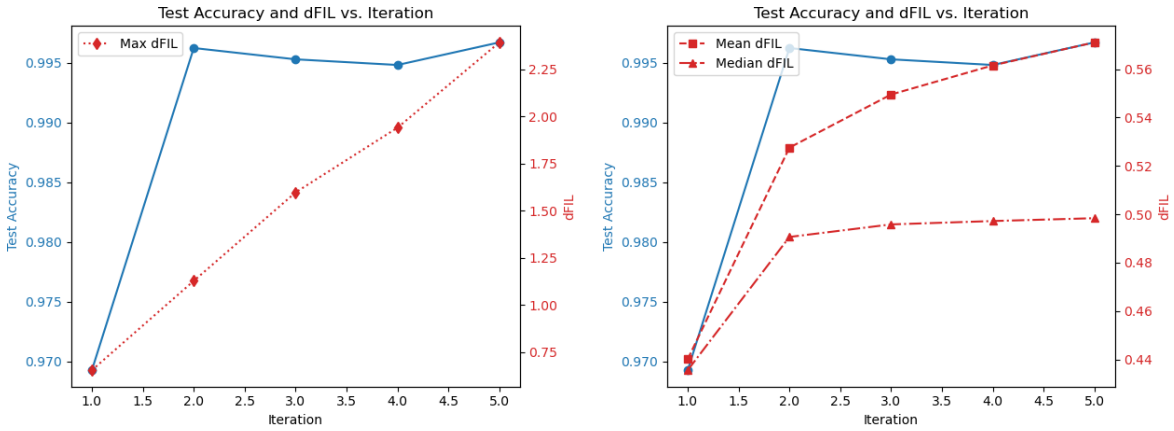


**Figure A.23.** Binary MNIST Linear Model Reconstruction Attack Accuracy and dFIL. Random Seed: 2

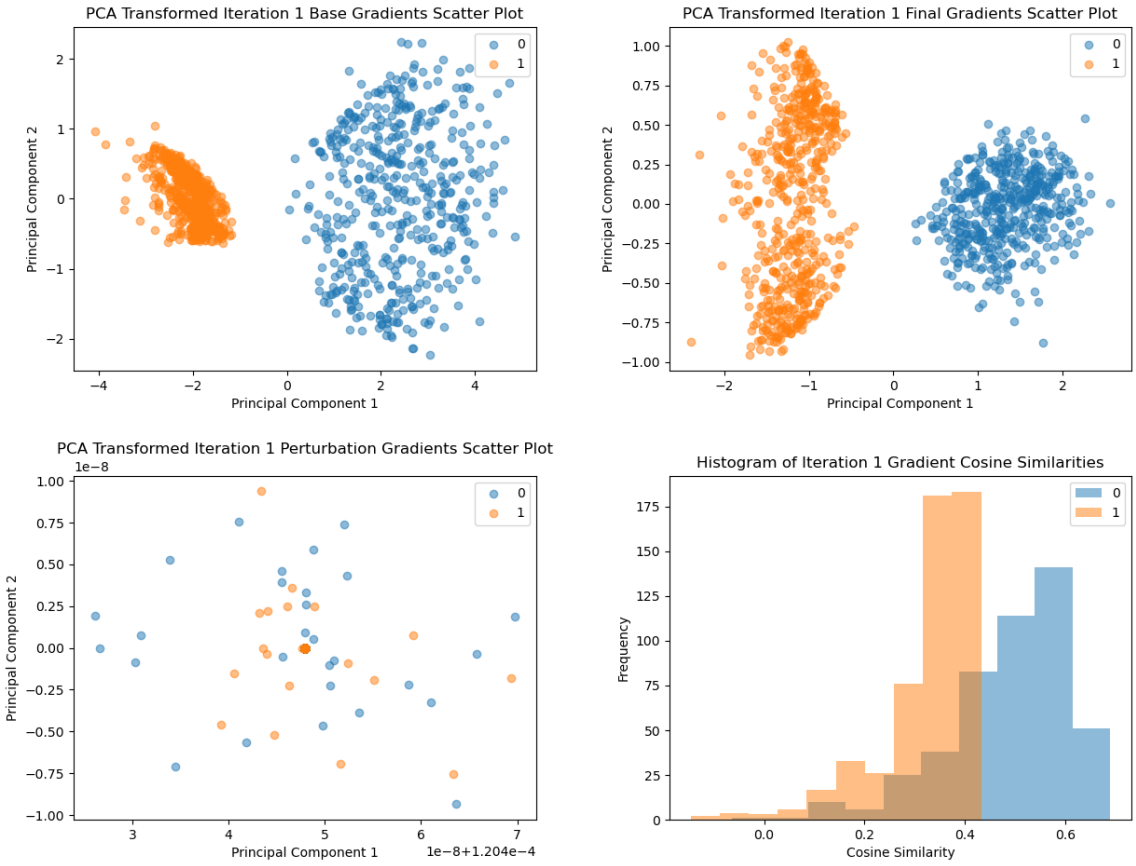


**Figure A.24.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 1; Random Seed: 2

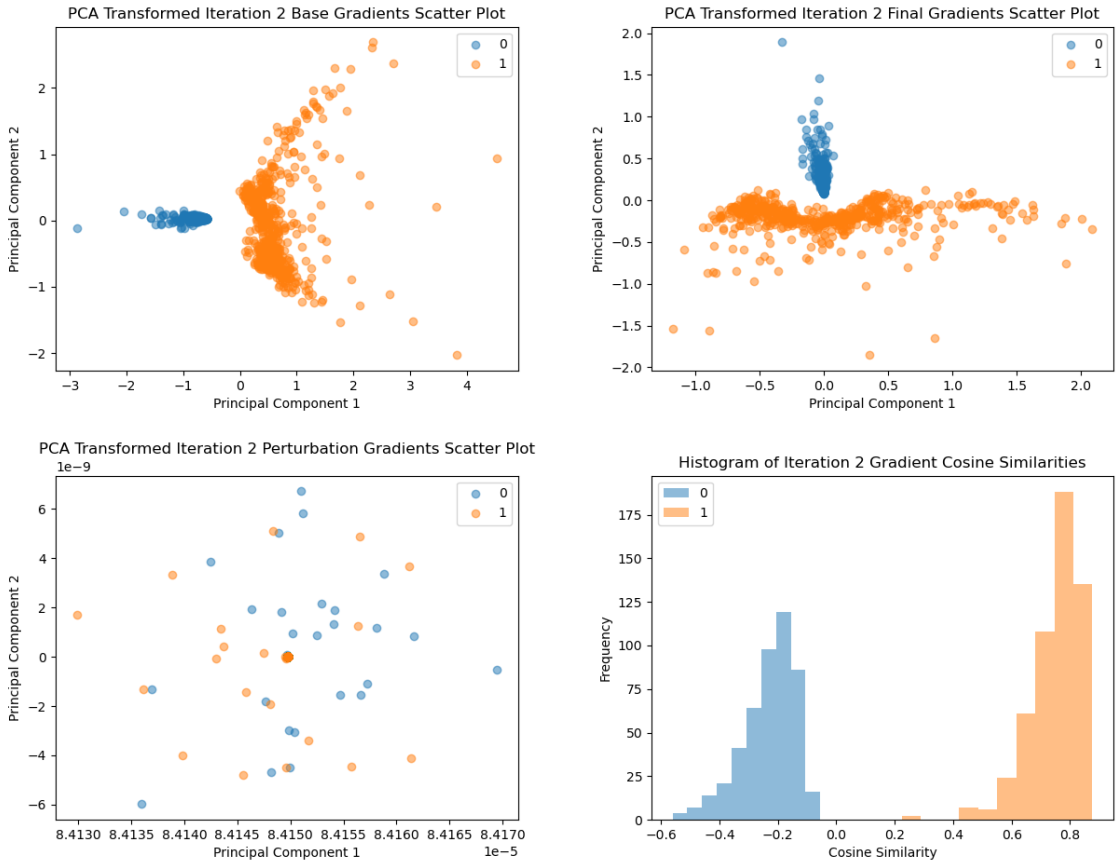


**Figure A.25.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 2; Random Seed: 2

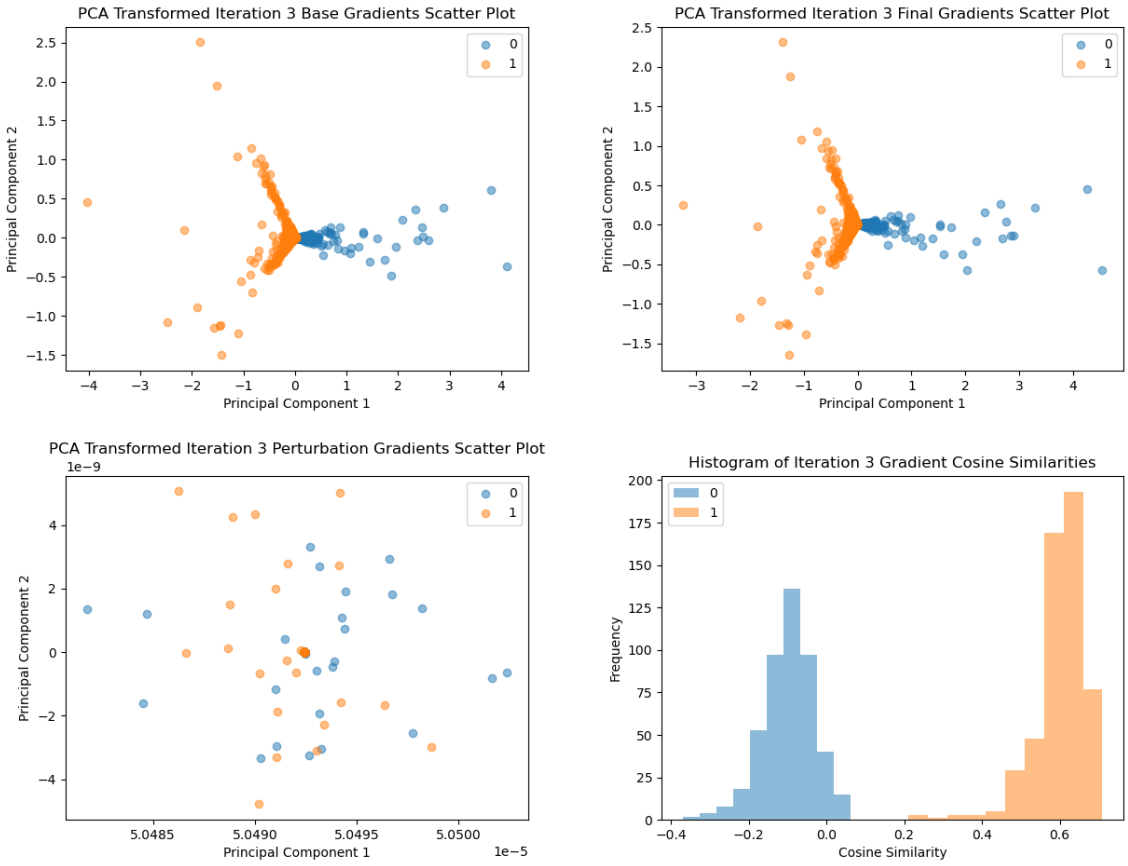


**Figure A.26.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 3; Random Seed: 2

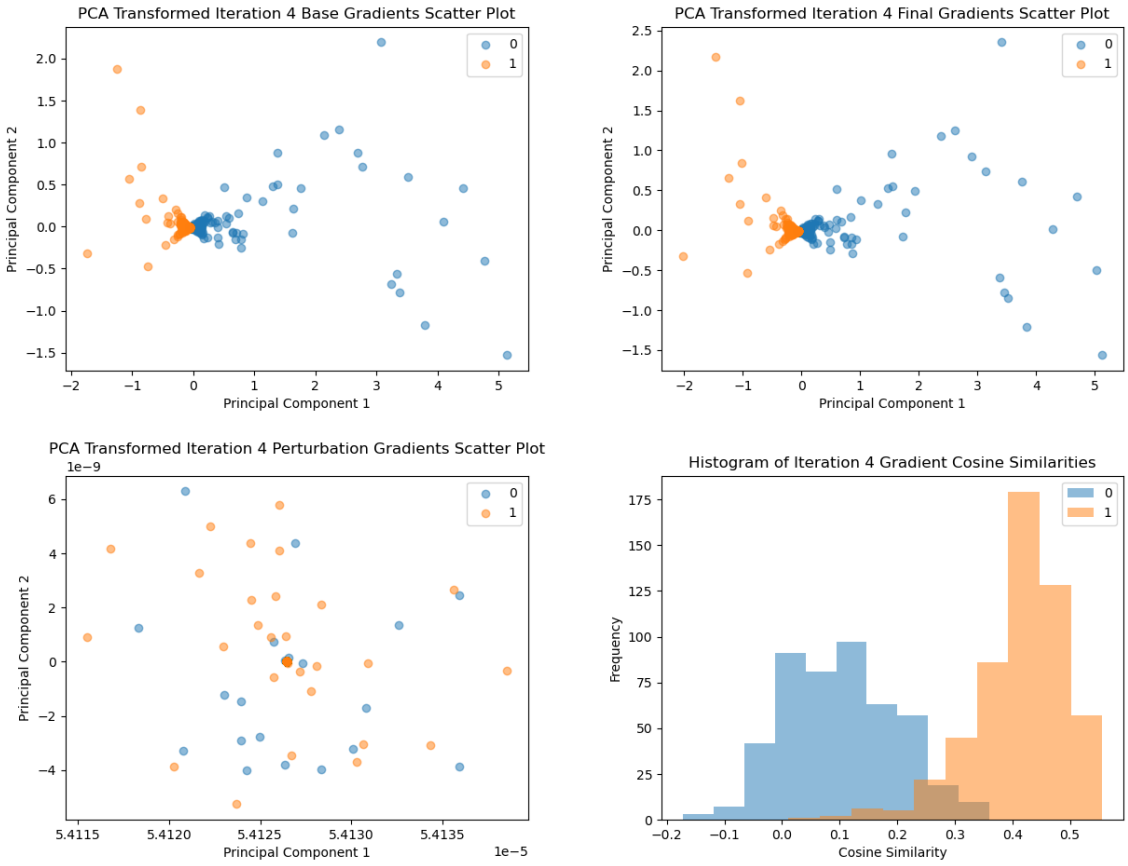


**Figure A.27.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 4; Random Seed: 2

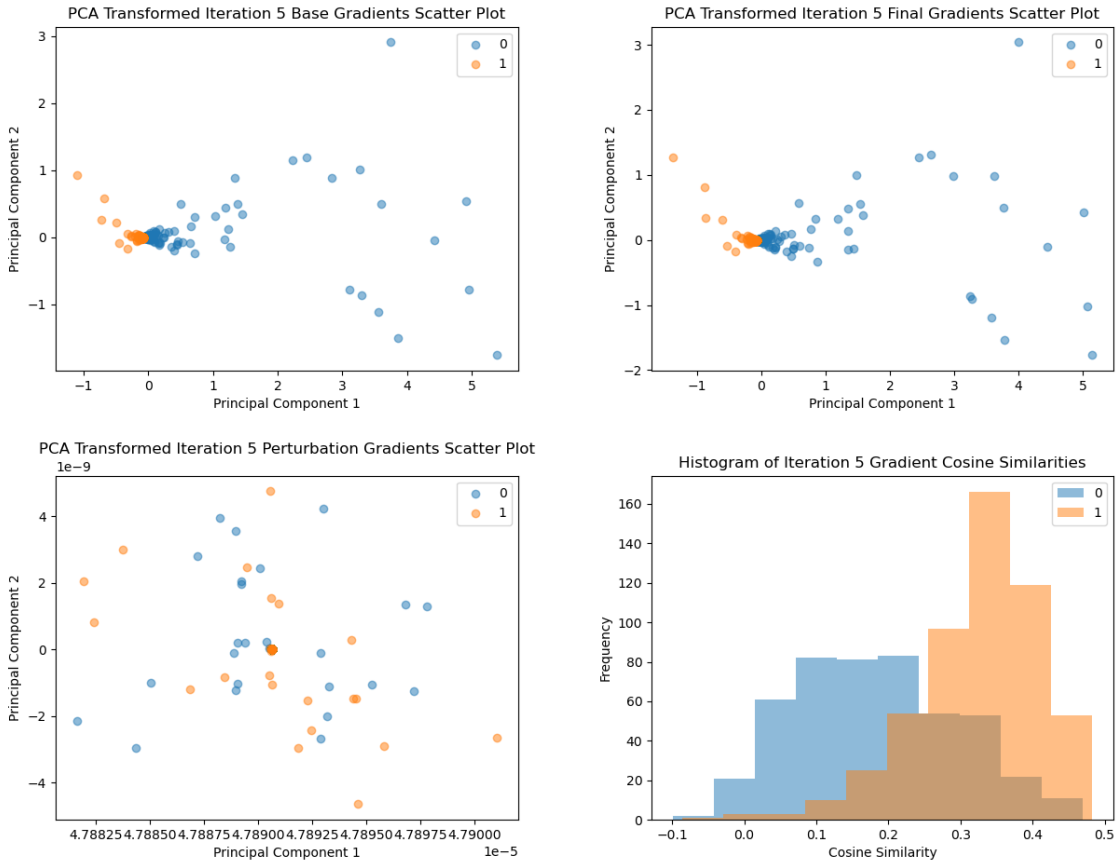


**Figure A.28.** Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 5; Random Seed: 2

## A.4 No Batch Normalization Layer

FIM behaves differently with static input standardization instead of using a Batch Normalization layer. In this section, the Batch Normalization layers are removed from the CNN used to learn the new gradient function, instead standardizing the inputs statically. Static standardization does not allow for using the weights as inputs since their values are not known at initialization. Similarly, only the gradients in the current iteration are known, and are standardized accordingly, hence a new FIM model is trained from scratch at every iteration, and the weights become a constant, and thus unnecessary input.

The FIM hyperparameters are modified when removing Batch Normalization. The learning rate is 0.01, the weight decay is 0.2, and the number of epochs is 1000. In Figure A.29, the perturbation is seen to no longer be constant, and the first PCA dimension seems to be disparately impacted, showing that FIM can behave differently and still maintain decent performance, as shown in Table A.1, but due to decreased performance compared to using Batch Normalization, this setting is not further considered.

**Table A.1.** No Batch Normalization (BN) Binary MNIST Linear Model Reconstruction Attack; $\epsilon = 389.9752$; Iteration: 1; Mean (STD)

| Method | Accuracy | Max dFIL | Mean dFIL | Median dFIL |
|---|---|---|---|---|
| BN | 0.9685 (0.0008) | 0.6216 (0.0223) | 0.4406 (0.0058) | 0.4433 (0.0072) |
| No BN | 0.9614 (0.0238) | 3.2741 (1.9908) | 0.4239 (0.0124) | 0.4086 (0.0285) |

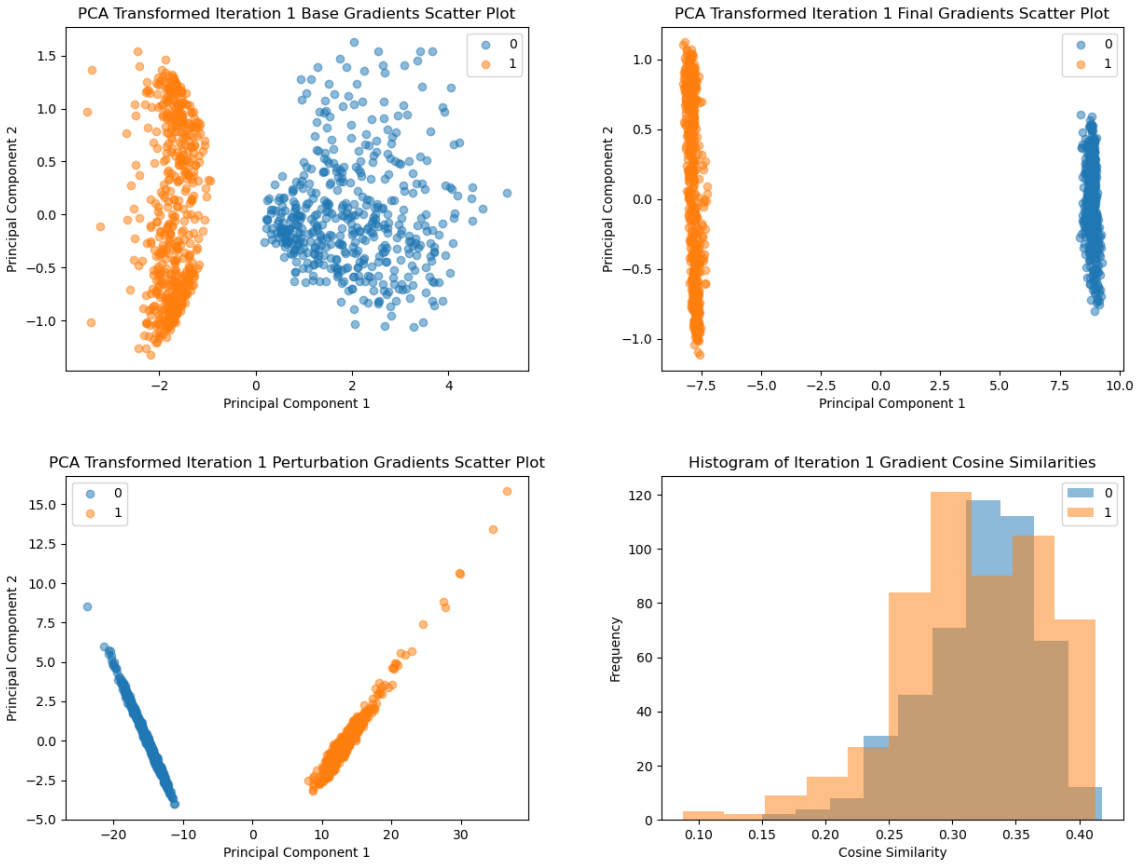


**Figure A.29.** No Batch Normalization Binary MNIST Linear Model Reconstruction Attack. Prune (top left), FIM before noise (top right), Perturbation (bottom left), Cosine Similarity between Perturbation and Prune (bottom right). Iteration: 1

# Appendix B
# Additional Experiment Details

Throughout the Appendix, any unspecified values are left as the default in the repository. A lower precedence indicates that the lower precedence layer comes before a higher precedence layer. Layers with equal precedence are done in parallel, with the results concatenated immediately afterwards. That is, there are two separate inputs to the model. In the case of Table B.6, the image is given to the convolutional layer, and the rest of the input is given to the identity layer (which does nothing). The Batch Normalization (BN) layer is from [16]. The Batch Normalization layer is essential since the inputs cannot be statically normalized. To train the gradient model, the model is first trained with the full number of epochs, then for the next batch of private examples, the number of epochs is decreased by the value of epoch decay, then the number of epochs continues to decrease each iteration until reaching the minimum number of epochs. For Table 3.5, note that different hyperparameters were used to optimize the attack for each case.

**Table B.1.** Section 3.1 Hyperparameters

| | Hyperparameter | Figure 3.1 | Table 3.2 | Table 3.3 |
|---|---|---|---|---|
| Pre-train | Batch Size | 512 | 512 | 512 |
| | Epochs | 30 | 100 | 10 |
| | Learning Rate | 0.01 | 0.01 | 0.001 |
| | Weight Decay | 0.0001 | 0.01 | 0.1 |
| | Optimizer | Adam | Adam | Adam |
| Train | Batch Size | 512 | 512 | 512 |
| | Prune Type | Magnitude | Magnitude | Magnitude |
| | Parameter density | 0.5 | 0.5 | 0.5 |
| | Epochs | 1 | 1 | 1 |
| | Learning Rate | 0.01 | 0.01 | 0.001 |
| | Weight Decay | 0.00001 | 0.0001 | 0.01 |
| | Optimizer | Adam | Adam | Adam |
| | $\sigma$ | 0.5 | 2 | 1 |
| | Clip | 10 | 8 | 10 |
| FIM | Batch Size | 512 | 512 | 512 |
| | Epochs | 200 | 200 | 200 |
| | Epoch Decay | 50 | 50 | 50 |
| | Min Epochs | 20 | 20 | 20 |
| | Learning Rate | 0.01 | 0.01 | 0.001 |
| | Weight Decay | 0.1 | 0.1 | 0.0 |
| | Optimizer | Adam | Adam | Adam |
| | $\lambda$ | 1e3 | 1e3 | 1e3 |

**Table B.2.** Binary MNIST Linear Model Reconstruction Attack Common Hyperparameters

| | Hyperparameter | Value |
|---|---|---|
| Train | Private Samples | 1000 |
| | Batch Size | 1024 |
| | Prune Type | Magnitude |
| | Parameter density | 0.5 |
| | Epochs | 5 |
| | Learning Rate | 0.1 |
| | Weight Decay | 0.0 |
| | Optimizer | Adam |
| | $\sigma$ | 0.05 |
| | Clip | 10 |
| FIM | Public Samples | 1000 |
| | Batch Size | 1024 |
| | Epochs | 200 |
| | Epoch Decay | 50 |
| | Min Epochs | 20 |
| | Learning Rate | 0.01 |
| | Weight Decay | 0.1 |
| | Optimizer | Adam |
| | $\lambda$ | 1e3 |
| Attack | Train Samples | 9500 |
| | Test Samples | 500 |

**Table B.3.** Table 3.5 Hyperparameters Part 1

| Hyperparameter | SGD | DPSGD | Prune |
|---|---|---|---|
| Batch Size | 64 | 64 | 64 |
| Epochs | 100 | 100 | 100 |
| Learning Rate | 0.001 | 0.001 | 0.001 |
| Weight Decay | 0.01 | 0.01 | 0.1 |
| Optimizer | Adam | Adam | Adam |

**Table B.4.** Table 3.5 Hyperparameters Part 2

| Hyperparameter | FIM | Perfect Privacy |
|---|---|---|
| Batch Size | 64 | 64 |
| Epochs | 30 | 100 |
| Learning Rate | 0.01 | 0.001 |
| Weight Decay | 0.1 | 0.01 |
| Optimizer | Adam | Adam |

**Table B.5.** MLP FIM

| Precedence | Layer | Hyperparameters |
|---|---|---|
| 1 | BN | axis=0 |
| 2 | Dense | 512 |
| 3 | GeLU | |
| 4 | BN | axis=0 |
| 5 | Dense | 256 |
| 6 | GeLU | |
| 7 | Dense | 10 |

**Table B.6.** CNN FIM

| Precedence | Layer | Hyperparameters |
|---|---|---|
| 1 | Conv | 16 8x8 filters, stride 2, padding 2 |
| 2 | GeLU | |
| 3 | AvgPool | 2x2, stride 1 |
| 4 | Conv | 32 4x4 filters, stride 2, padding 0 |
| 5 | GeLU | |
| 6 | AvgPool | 2x2, stride 1 |
| 1-6 | Identity | |
| 7 | BN | axis=0 |
| 8 | Dense | 256 |
| 9 | GeLU | |
| 10 | BN | axis=0 |
| 11 | Dense | 256 |
| 12 | GeLU | |
| 13 | Dense | 10 |

**Table B.7.** CNN MNIST

| Precedence | Layer | Hyperparameters |
|---|---|---|
| 1 | Conv | 16 8x8 filters, stride 2, padding 2 |
| 2 | Tanh | |
| 3 | AvgPool | 2x2, stride 1 |
| 4 | Conv | 32 4x4 filters, stride 2, padding 0 |
| 5 | Tanh | |
| 6 | AvgPool | 2x2, stride 1 |
| 8 | Dense | 32 |
| 9 | Tanh | |
| 10 | Dense | 10 |

**Table B.8.** MLP Attack

| Precedence | Layer | Hyperparameters |
|---|---|---|
| 1 | Dense | 512 |
| 2 | ReLU | |
| 3 | Dense | 256 |
| 4 | ReLU | |
| 5 | Dense | 10 |

# Bibliography


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

[2] Kamil Adamczewski and Mijung Park. Differential privacy meets neural network pruning, 2023.

[3] Ehsan Amid, Arun Ganesh, Rajiv Mathews, Swaroop Ramaswamy, Shuang Song, Thomas Steinke, Vinith M. Suriyakumar, Om Thakkar, and Abhradeep Thakurta. Public data-assisted mirror descent for private model training, 2022.

[4] Efe Aras, Kuan-Yun Lee, Ashwin Pananjady, and Thomas A. Courtade. A family of bayesian cramér-rao bounds, and consequences for log-concave priors. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2699–2703, 2019.

[5] Borja Balle, Giovanni Cherubin, and Jamie Hayes. Reconstructing training data with informed adversaries, 2022.

[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[7] Zhiqi Bu, Xinwei Zhang, Mingyi Hong, Sheng Zha, and George Karypis. Pre-training differentially private models with limited public data, 2024.

[8] Soham De, Leonard Berrada, Jamie Hayes, Samuel L. Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale, 2022.

[9] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[10] Richard Gill and Boris Levit. Applications of the van trees inequality: A bayesian cramér-rao bound. *Bernoulli*, 1:59–79, 03 1995.

[11] Chuan Guo, Brian Karrer, Kamalika Chaudhuri, and Laurens van der Maaten. Bounding training data reconstruction in private (deep) learning, 2022.

[12] Awni Hannun, Chuan Guo, and Laurens van der Maaten. Measuring data leakage in machine-learning models with fisher information, 2021.

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[14] Judy Hoffman, Daniel A. Roberts, and Sho Yaida. Robust learning with jacobian regularization, 2019.

[15] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communication in Statistics- Simulation and Computation*, 19:432–450, 01 1990.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Tian Li, Manzil Zaheer, Sashank J. Reddi, and Virginia Smith. Private adaptive optimization with side information, 2022.

[20] Kiwan Maeng, Chuan Guo, Sanjay Kariyappa, and Edward Suh. Measuring and controlling split layer privacy leakage using fisher information, 2022.

[21] Kiwan Maeng, Chuan Guo, Sanjay Kariyappa, and G. Edward Suh. Bounding the invertibility of privacy-preserving instance encoding using fisher information, 2023.

[22] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, August 2017.

[23] Milad Nasr, Saeed Mahloujifar, Xinyu Tang, Prateek Mittal, and Amir Houmansadr. Effectively using public data in privacy preserving machine learning, 2023.

[24] Milad Nasr, Shuang Songi, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlin. Adversary instantiation: Lower bounds for differentially private machine learning. In *2021 IEEE Symposium on security and privacy (SP)*, pages 866–882. IEEE, 2021.

[25] Nicolas Papernot and Thomas Steinke. Hyperparameter tuning with renyi differential privacy, 2022.

[26] Jasper Tan, Blake Mason, Hamid Javadi, and Richard G. Baraniuk. Parameters or privacy: A provable tradeoff between overparameterization and membership inference, 2022.

[27] Yijue Wang, Chenghong Wang, Zigeng Wang, Shanglin Zhou, Hang Liu, Jinbo Bi, Caiwen Ding, and Sanguthevar Rajasekaran. Against membership inference attack: Pruning is all you need, 2021.

[28] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.