
GRADIENT INVERSION TRANSCRIPT: LEVERAGING ROBUST GENERATIVE PRIORS TO RECONSTRUCT TRAINING DATA FROM GRADIENT LEAKAGE

Xinping Chen

Department of Computer Science

City University of Hong Kong

83 Tat Chee Avenue, Kowloon Tong, Hong Kong
xinpichen2-c@my.cityu.edu.hk

Chen Liu

Department of Computer Science

City University of Hong Kong

83 Tat Chee Avenue, Kowloon Tong, Hong Kong
chen.liu@cityu.edu.hk

May 27, 2025

ABSTRACT

We propose Gradient Inversion Transcript (GIT), a novel generative approach for reconstructing training data from leaked gradients. GIT employs a generative attack model, whose architecture is tailored to align with the structure of the leaked model based on theoretical analysis. Once trained offline, GIT can be deployed efficiently and only relies on the leaked gradients to reconstruct the input data, rendering it applicable under various distributed learning environments. When used as a prior for other iterative optimization-based methods, GIT not only accelerates convergence but also enhances the overall reconstruction quality. GIT consistently outperforms existing methods across multiple datasets and demonstrates strong robustness under challenging conditions, including inaccurate gradients, data distribution shifts and discrepancies in model parameters.

Keywords Gradient inversion · Training data reconstruction · Generative model · Image prior

1 Introduction

In distributed learning, each client trains its model on local data and shares the gradients with a central server, which aggregates them to update the global model [1, 2, 3]. Gradient sharing is also common in federated learning (FL) [4], but unlike distributed learning, which involves a more centrally coordinated distribution of data across clients, federated learning focuses on preserving data privacy by ensuring that client data remains localized. While these methods are effective in improving performance and efficiency without directly exposing the client’s data to public, recent research [5, 6, 7] has shown that the gradients leaked by sharing can still cause sensitive information leakage, as attackers may exploit them to reconstruct the original training data used by the individual client, posing significant privacy risks in real-world learning systems.

There is a considerable amount of works proposed to reconstruct the training data from its gradient [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] based on varying levels of model access, as shown in Table 1. These works generally fall into two major categories: iterative optimization methods, which iteratively optimize the reconstructed data to align its gradients with the leaked ones; and generative methods, which leverage a generative model to approximate the user data. Generative methods can be further subdivided into two types: latent space-based and generative model-based. The first type trains a latent space as the input of a pre-trained generative model. The second type trains a generative model to map leaked gradients to the corresponding user data.

Iterative optimization methods typically require repeated access to gradients from the target model [5, 6, 8, 9, 11, 4, 18] or full access to the model parameters [10, 13]. In contrast, the latent space-based methods [12, 14, 15, 16, 19] relies on a pre-trained generative model and public data that closely matches the distribution of the user data to train the latent space. The generative model-based methods [17] require input-gradient pairs from public datasets to train the generative model.

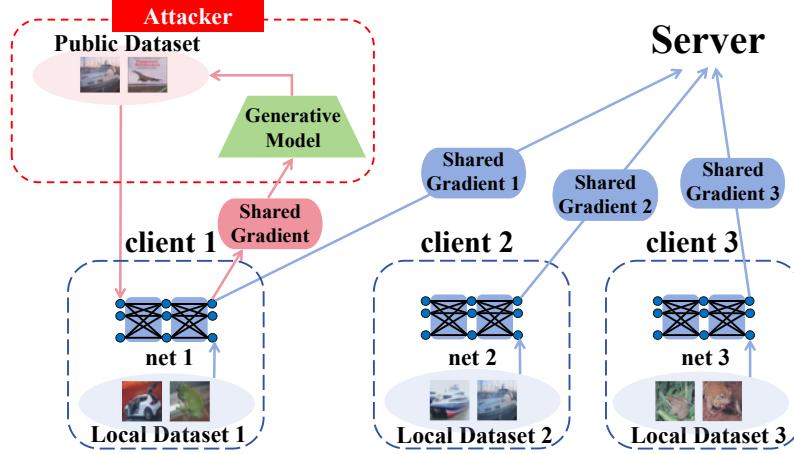


Figure 1: A flowchart of problem settings for GIT. The attacker hacks the channel of one client to inject data and utilizes the obtained input-gradient pair to train generative models. The attacker aims to reconstruct the data from *both the hacked client and other clients* by shared gradients.

We focus on the generative models for input data reconstruction in this work. We call the model under attack the *leaked model*. The existing generative model-based method, such as LTI [17], usually employs a fixed-architecture multi-layer perception (MLP) [20] as the generative model. However, this approach lacks justification for how gradients relate to the input data. We argue that the generative model to reconstruct the input data from the gradient should approximate the inverse of the gradient computation process and thus be adaptive to the architecture of the leaked model. In this context, we introduce **gradient inversion transcript (GIT)** in this work to adaptively choose the architecture of the generative model to improve its effectiveness. In addition, we can combine GIT with iterative optimization methods like IG [9], in which we use GIT's output as the initial estimation of the input data for iterative optimization methods to further refine the reconstructed input data.

Problem Settings: In general, we assume that the attacker only has access to the leaked gradients and the model architecture. Our method does not need the parameters of the leaked model or the labels of the training data. As in Figure 1, we adopt a similar premise to DLG [6]: the attacker hacks the channel to inject data to one client, which shares the gradient with the server and other clients. The attack aims to reconstruct the data from *both the hacked client and other clients* by shared gradients. When combining with iterative optimization methods, we additionally assume that the attacker can repeatedly query different clients to obtain gradients. The problem settings and comparison with existing literature are summarized in Table 1.

Challenges: Figure 1 provides an overview of the generative approach. Beyond the basic case where the training data for generative model follows the same distribution as the input data for recovery, we may encounter more challenging scenarios: (1) The clients may send defensive inaccurate gradients, like clipped gradients [6] and noisy gradients [21]; (2) When recovering data from other clients based on their shared gradients, challenges arise due to distributional shifts between data on different nodes and slight discrepancies in model parameters across nodes due to lack of synchronization. Generative methods can adapt to both scenarios, unlike iterative optimization-based methods, which are not applicable for scenario (2) since the attacker has no access to inject data to unhacked clients and is therefore unable to reconstruct training data from them.

Compared with iterative optimization-based methods, our method can train generative models offline and is much more efficient during deployment, making it well-suited for real-time reconstruction tasks with a small tolerated delay. Compared with latent space-based methods, GIT demonstrates greater robustness to out-of-distribution (OOD) scenarios, as it focuses on inverting the backpropagation process and implicitly recovering the model parameters. Compared with other generative model-based methods, GIT is broadly applicable to leaked models with diverse architectures and achieves better performance. Moreover, GIT remains effective under challenging reconstruction scenarios.

We summarize our contributions as following points:

- We propose **Gradient Inversion Transcript (GIT)**, which is inspired by back-propagation and constructs a generative model whose architecture is tailored to adapt the leaked model. GIT is shown to effectively reconstruct the input data given its gradient without the knowledge of model parameters and data labels.

- GIT can be efficiently deployed after offline training. Compared with existing methods, GIT can achieve the best performance in most cases. In addition, the outputs of GIT can serve as the prior for gradient matching, further improving the performance.
- GIT is generally applicable and has robust performance under some challenging performance. It remains effective under discrepancies in model parameters, and achieves best performance under inaccurate gradients and data distributional shift.

2 Related Work

Table 1: The comparison in terms of attacker’s access for different input data reconstruction methods. **dist.** means access to distribution of labels. The four categories of methods are separated by dashed lines and, from top to bottom, are: parameter-based methods, iterative optimization-based methods, latent space–based methods, and generative model–based methods.

Method	Model Parameters	Model Architecture	Shared Gradients	Gradient Query	Output Logit	Data Label	Public Data
[22]	✓	✓	✗	✗	✗	✗	✗
[5, 6, 8, 9, 11, 4, 18]	✗	✗	✓	✓	✗	✓	✗
[7, 23]	✗	✗	✓	✓	✗	dist.	✗
[24]	✓	✓	✓	✗	✓	✓	✗
[10, 13]	✓	✓	✓	✗	✗	✓	✗
[12, 19]	✗	✗	✓	✓	✗	dist.	✓
[14, 15, 16]	✗	✗	✓	✓	✗	✓	✓
[17], GIT	✗	✓	✓	✓	✗	✗	✓

To recover the individual training data, gradient-based reconstruction is the most commonly investigated scenario. Nevertheless, it is also worth noting that reconstructing datasets using model parameters only is also viable. Such parameter-based methods [22] do not utilize any data-dependent information. Therefore, the method is unable to recover high-quality data and fails to achieve pixel-wise accuracy. Consequently, gradient inversion attacks are more widely investigated in the context of the leaked gradients.

Optimization-Based Methods. The feasibility of optimization-based method for data reconstruction from gradient leakage was initially explored by [5] [6] demonstrated its practicality by proposing Deep Leakage from Gradients (DLG). DLG optimizes a randomly generated dummy input to estimate the training data by matching its gradients and the leaked ground truth gradients. There are several subsequent works improving DLG from optimization perspectives [8, 9, 11, 10, 13] with settings as shown in Table 1. [18] considers a more realistic scenario for data reconstruction, such as multiple local epochs, heterogeneous data and various optimizers.

Generative-Based Methods. Unlike optimization-based methods, generative approaches estimate the distribution of user data using a generative model, which maps the leaked gradient to input data estimation or the initial dummy input for subsequent optimization-based refinement. Generative-based methods generally have two major categories which are based on latent space and generative models, respectively. The first type trains a latent space representation and uses a pre-trained generative model to synthesize estimations of the user data. The second type directly trains a generative model to map leaked gradients to the corresponding user data.

Early attempts of the latent space-based method [12] use a pre-trained generative model to produce image priors for reconstruction. Building on this, GIAS [14, 4] employs a generative adversarial network (GAN) [25] as the generative model and alternately searches both the latent space and the parameter space of the generator. However, GIAS is computationally prohibitive, as it requires training a new generator for each reconstructed image. In this context, there are several works [15, 16, 19] on improving the efficiency and performance of GIAS under different settings as shown in Table 1.

The generative model-based method was originally proposed by [17] as Learning to Invert (LTI). Specifically, they use a three-layer multi-layer perceptron (MLP) with a fixed hidden size as the generative model. LTI employs a generative model of a fixed architecture regardless of the leaked model, which may not be optimal. In contrast, we introduce

gradient inversion transcript (GIT), which is a framework that dynamically selects the architecture of the threat model based on the leaked model to enhance performance.

3 Input Data Reconstruction by Back-Propagation

3.1 A General Framework

As in Figure 2, we consider the generic architecture of a multi-input multi-output (MIMO) layer with nonlinear elementwise activation functions as follows:

$$\begin{aligned} \mathbf{z} &= \sum_{i=1}^{N_{in}} \mathbf{W}_i^{(in)} \mathbf{a}_i^{(in)}, \quad \mathbf{z}_j^{(out)} = \mathbf{W}_j^{(out)} \mathbf{a}, \\ \mathbf{a} &= \sigma(\mathbf{z}), \quad \mathbf{a}_j^{(out)} = \sigma_j(\mathbf{z}_j^{(out)}), \quad j = 1, \dots, N_{out} \end{aligned} \quad (1)$$

The MIMO layer is connected with N_{in} input layers and N_{out} output layers. σ and $\{\sigma_i\}_{i=1}^{N_{out}}$ are the nonlinear activation functions. We let B be the batch size, $\mathbf{z} \in \mathbb{R}^{B \times d}$ and $\mathbf{a} \in \mathbb{R}^{B \times d}$ represent the pre-activation and post-activation of this MIMO layer, respectively. Similarly, $\left\{ \left(\mathbf{z}_i^{(in)} \in \mathbb{R}^{B \times d_i^{(in)}}, \mathbf{a}_i^{(in)} \in \mathbb{R}^{B \times d_i^{(in)}} \right) \right\}_{i=1}^{N_{in}}$ and $\left\{ \left(\mathbf{z}_j^{(out)} \in \mathbb{R}^{B \times d_j^{(out)}}, \mathbf{a}_j^{(out)} \in \mathbb{R}^{B \times d_j^{(out)}} \right) \right\}_{j=1}^{N_{out}}$ represent the pre-activation and post-activation pairs for the input layers and the output layers, respectively. In addition, $\left\{ \mathbf{W}_i^{(in)} \in \mathbb{R}^{d \times d_i^{(in)}} \right\}_{i=1}^{N_{in}}$ and $\left\{ \mathbf{W}_j^{(out)} \in \mathbb{R}^{d_j^{(out)} \times d} \right\}_{j=1}^{N_{out}}$ refer to the weights connecting this layer and its adjacent layers. We replace notation \mathbf{W} with \mathbf{g} to represent the gradient of the loss function \mathcal{L} w.r.t its weights, e.g., $\mathbf{g}_i^{(in)} = \nabla_{W_i^{(in)}} \mathcal{L}$, $\mathbf{g}_j^{(out)} = \nabla_{W_j^{(out)}} \mathcal{L}$. We omit the bias term for notation simplicity, since the bias terms can be incorporated as part of the weight matrices.

We have the following equations by back-propagation:

$$\mathbf{g}_j^{(out)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_j^{(out)}} \otimes \mathbf{a}^T, \quad \mathbf{g}_i^{(in)} = \left(\left(\sum_{j=1}^{N_{out}} \mathbf{W}_j^{(out)T} \otimes \frac{\partial \mathcal{L}}{\partial \mathbf{z}_j^{(out)}} \right) \odot \sigma'(\mathbf{z}) \right) \otimes \mathbf{a}_i^{(in)T} \quad (2)$$

Here we define operator \otimes as tensor multiplication and operator \odot as broadcast row-wise product. In addition, \mathbf{z} , \mathbf{a} are broadcast as a tensor of shape $B \times d \times 1$, similar broadcast mechanisms are applied to $\mathbf{z}_j^{(out)}$ and $\mathbf{z}_i^{(in)}$; $\mathbf{W}_j^{(out)}$ is broadcast as a tensor of shape $1 \times d_j^{(out)} \times d$, and the transpose operator $(\cdot)^T$ switches the second and the third dimensions of a 3-d tensor. Based on Equation (2), we cancel out $\partial \mathcal{L} / \partial \mathbf{z}_j^{(out)}$ and approximate the input of the layer as follows:

$$\mathbf{a}_i^{(in)T} \simeq \left(\left(\sum_{j=1}^{N_{out}} \mathbf{W}_j^{(out)T} \otimes \mathbf{g}_j^{(out)} \otimes (\mathbf{a}^T)^+ \right) \odot \sigma'(\mathbf{z}) \right)^+ \otimes \mathbf{g}_i^{(in)} \quad (3)$$

Here we use $(\cdot)^+$ to represent the Moore–Penrose inverse of a matrix. For a third-order tensor, $(\cdot)^+$ calculate the Moore–Penrose inverse of each of its subspace via the first dimension. Equation (3) establishes the formulation wherein we leverage the gradients, the parameters and the output activation to estimate the input data of a neuron. For a neural network of general architecture, we can estimate the input of each layer following back-propagation and ultimately obtain the reconstructed input data.

Mini-Batch Training. In mini-batch training, $\mathbf{g}_i^{(in)}$ and $\mathbf{g}_j^{(out)}$ obtained by Equation (2) contains the gradient information for all data instances in the mini-batch. In practice, the leaked the gradient is their average over the mini-batch, that is $\mathbf{g}_i^{(in)} \leftarrow \mathbb{E}_b \mathbf{g}_i^{(in)}[b, :, :], \mathbf{g}_j^{(out)} \leftarrow \mathbb{E}_b \mathbf{g}_j^{(out)}[b, :, :]$. Since the leaked gradient is the average over the mini-batch. When reconstructing the input data, we broadcast the leaked gradient in the dimension of batch size in Equation (3).

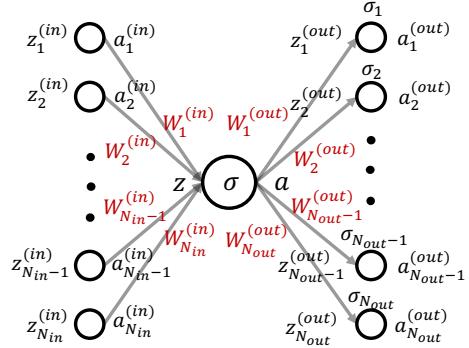


Figure 2: An MIMO layer.

Generality. Our analysis is generic and can be applied to general neural network architectures as long as they support back-propagation. For multi-layer perceptrons (MLP) and vanilla convolutional neural networks (CNN) like LeNet, we have $N_{in} = N_{out} = 1$ for all layers; for residual networks (ResNet), we have $N_{out} > 1$ for layers which receive the inputs from both the preceding layer and the shortcut connections. Our framework is also compatible with more complicated architectures like attention mechanism in transformers [26]. We defer detailed derivation for these popular architectures in Appendix B.2.

3.2 Modularized Input Data Reconstruction

For models with large amount of parameters, it would be computationally expensive to infer the input data by recursively using Equation (3). In this context, we formulate the large model as a composition of several modules and apply input data reconstruction on the module level instead of the layer level. We re-consider the multi-input multi-output (MIMO) layer as in Section 3.1 with input and output connections followed by functions $\{f_i^{(in)}\}_{i=1}^{N_{in}}, \{f_j^{(out)}\}_{j=1}^{N_{out}}$, respectively:

$$\mathbf{z} = \sum_{i=1}^{N_{in}} f_i^{(in)}(\mathbf{W}_i^{(in)} \mathbf{a}_i^{(in)}), \mathbf{z}_j^{(out)} = f_j^{(out)}(\mathbf{W}_j^{(out)} \mathbf{a}), j = 1, \dots, N_{out}. \quad (4)$$

$\mathbf{a}_i^{(in)}$ and \mathbf{a} are calculated in the same way as in Equation (1). We follow the derivation as in Section 3.1 and obtain the following equation for modularized input data reconstruction:

$$\mathbf{a}_i^{(in)T} = \left(\left(\sum_{j=1}^{N_{out}} \mathbf{W}_j^{(out)T} \otimes \mathbf{g}_j^{(out)} \otimes (\mathbf{a}^T)^+ \right) \odot \sigma'(\mathbf{z}) \otimes f_i'^{(in)}(\mathbf{W}_i^{(in)} \mathbf{a}_i^{(in)}) \right)^+ \otimes \mathbf{g}_i^{(in)} \quad (5)$$

Equation (5) demonstrates modularized input data reconstruction. It establishes a high-level formulation to estimate input data for large models. However, we need the gradient information from the input module $f_i^{(in)}$ to estimate $f_i'^{(in)}(\mathbf{W}_i^{(in)} \mathbf{a}_i^{(in)})$, which will be elaborated in the next section.

4 GIT: Gradient Inverse Transcript

Exact-GIT. Based on Equation (3) and the analyses in Section 3.1, the input value $\mathbf{a}_i^{(in)}$ of a general MIMO layer can be estimated from the activation \mathbf{a} , the gradient $\mathbf{g}_i^{(in)}$ of the input weight and output weights $\{\mathbf{W}_j^{(out)}\}_{j=1}^{N_{out}}$. Therefore, we can recursively utilize Equation (3) to reconstruct the input data by a generative model with all unknown variables, such as the weights, as its trainable parameters. We use the mean square error between the true input data and its estimation as the loss objective function. Once trained, the generative model can subsequently reconstruct the training data batch using the leaked gradients as input during inference.

The detailed pseudo-code for the training and the inference phase is shown as Algorithm 1. The key innovation of our method is that we adaptively adjust the architectures of the generative models by Equation (3) based on the leaked model and map the leaked gradients to the estimated input data, so we name it *gradient inverse transcript (GIT)*. We further name our method *Exact-GIT* when we strictly follow Equation (3), i.e. using model weights as parameters for the generative models, for all layers to reconstruct the input data. We present some specific example architectures in Appendix B.2. The results of the Exact-GIT implementation are presented in Appendix B.1.

Coarse-GIT. Exact-GIT enjoys good interpretability but is computationally expensive for large models. Moreover, the Moore-Penrose inverse in Equation (3) would introduce numerical instability issues for large-scale tensors in practice. To tackle these issues, compared with Equation (3), we can also model such estimation in a more coarse-grained manner and name the corresponding method *Coarse-GIT*. Specifically, we utilize a shallow multi-layer perceptron (MLP) m_θ , parameterized by θ , to approximate the right-hand-side of Equation (3). The inputs of this shallow MLP are all the known variables on the right-hand-side of Equation (3), including the leaked gradients and the output activation. Therefore, like Equation (3), Coarse-GIT recursively estimates each layer's input by $\mathbf{a}_i^{(in)} = m_\theta(\{\mathbf{g}_j^{(out)}\}_{j=1}^{N_{out}}, \mathbf{g}_i^{(in)}, \mathbf{a})$. The generative model comprises multiple shallow MLPs, with orders based on back-propagation and collectively trained to minimize the difference between the estimated input and the corresponding ground truth.

Coarse-GIT also supports modularized input data reconstruction as discussed in Section 3.2, which is more computationally affordable. It employs a shallow MLP m_θ to estimate the right-hand-side of Equation (5): $\mathbf{a}_i^{(in)} =$

 Algorithm 1: Training and Inference of GIT

```

1: Training input: Training set of the generative model, i.e., input-gradient pairs  $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{g}^{(i)})\}_{i=1}^N$ . Epoch budget  $E$ . Batch size  $B$ . Learning rate  $\eta$ .
2: Initialization: Construct the generative model  $M$  parameterized by  $\Theta$  based on the architecture of the leaked model. Popular architectures are discussed as examples in Appendix B.2.
3: for the epoch index from 1 to  $E$  do
4:   for the batch index from 1 to  $N/B$  do
5:     Sample one mini-batch  $\{(\mathbf{x}^{(b_i)}, \mathbf{g}^{(b_i)})\}_{i=1}^B$ 
6:     if use Exact-GIT then
7:       Estimate the input  $\{\hat{\mathbf{x}}^{(b_i)}\}_{i=1}^B = M(\Theta, \{\mathbf{g}^{(b_i)}\}_{i=1}^B)$  by recursively using Equation (3).
8:     else
9:       Estimate the input  $\{\hat{\mathbf{x}}^{(b_i)}\}_{i=1}^B = M(\Theta, \{\mathbf{g}^{(b_i)}\}_{i=1}^B)$  by recursively using Equation (3) or Equation (5) with right hand side replaced by a shallow MLP discussed in Section 4.
10:    end if
11:    Calculate the loss  $\mathcal{L}^{(gen)} = \frac{1}{2B} \sum_{i=1}^B \|\hat{\mathbf{x}}^{b_i} - \mathbf{x}^{b_i}\|$  and update  $\Theta \leftarrow \Theta - \eta \nabla_{\Theta} \mathcal{L}^{(gen)}$ 
12:  end for
13: end for
14: Training output: GIT generator  $M$  with learned parameters  $\Theta$ .
15:
16: Inference input: GIT generator  $M$  with parameters  $\Theta$ . Leaked gradients  $\{\mathbf{g}^{(i)}\}_{i=1}^{N'}$ .
17: Inference output: Input data estimation  $\{\hat{\mathbf{x}}^{(i)}\}_{i=1}^{N'} = M(\Theta, \{\mathbf{g}^{(i)}\}_{i=1}^{N'})$ 

```

$m_{\theta} \left(\{\mathbf{g}_j^{(out)}\}_{j=1}^{N_{out}}, \mathbf{g}_i^{f'(in)}, \mathbf{g}_i^{(in)}, \mathbf{a} \right)$ where $\mathbf{g}_i^{f'(in)}$ represents the leaked gradients for the parameters in the input module $f_i^{(in)}$. Compared with layerwise reconstruction, modularized reconstruction only considers the high-level topologies of the leaked model, making it suitable for large models.

Bootstrap. For both Exact-GIT and Coarse-GIT, we need to estimate the output logits, i.e., last layer’s output, to start the recursive estimation. The average output logits over the mini-batch can be analytically estimated if the last layer has a bias term [10]. Otherwise, we use the leaked gradients for the weight of the last layers to estimate the output logits by a shallow MLP. The ablation studies are presented in Appendix D.

5 Experiments

We comprehensively assess our methods on various datasets, including CIFAR-10 [27], ImageNet [28] and facial datasets (Facial Expression Recognition (FER) from kaggle, Japanese Female Facial Expression (Jaffe) [29]). Correspondingly, we employ various model architectures, including LeNet [30], ResNet [31] and ViT [32] to comprehensively demonstrate the effectiveness of our methods. Since the generative models are trained by minimizing the mean square error (MSE) between the ground-truth and the estimated inputs, in addition to MSE, we also use peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), learned perceptual image patch similarity (LPIPS) as metrics to quantitatively and comprehensively evaluate the performance of training data reconstruction. PSNR, SSIM and LPIPS reflect more perceptual and structural differences than MSE.

5.1 Comparison with Baselines

We compare our method with baselines under two different settings: (1) we directly employ generative models to map the leaked gradient to the reconstructed input data; (2) we first employ generative models to obtain the input data estimation as priors and then refine the estimation by optimization-based methods. Unless specified, we use 10000 random samples and their gradients to train the generative models. More implementation details are deferred to Appendix C.

5.1.1 Direct Inference by Generative Models

As shown in Table 2, we first compare GIT with other generative models in direct inference. Specifically, we compare GIT with Learning to Invert (LTI) [17], which employs an MLP with approximately the same number of parameters as the generators. In addition, we include the performance of optimization-based methods for reference, such as Deep

Leakage from Gradients (DLG) [6] and Inverting Gradients (IG) [9]. The computational overhead for optimization-based methods and generative methods are fundamentally different. Optimization-based methods necessitate a complete optimization process for each batch data recovery, whereas generative methods need to train a generative model capable of retrieving data from the corresponding leaked gradients. We run both types of methods until their convergence and report the training and inference time for comparison.

The results in Table 2 include different tasks and network architectures. To save memory consumption and guarantee numerical stability, we adopt Coarse-GIT for all architectures and specifically modularized reconstruction for ViT. The GIT implementation details for specific architectures are deferred to Appendix C. The results indicate that GIT outperforms in most cases and metrics than baselines, including both optimization-based methods and generative methods.

Visual inspection of reconstructed ImageNet samples by GIT as shown in Appendix D.5 reveals that images with large uniform color regions tend to be recovered more accurately, while those containing complex structures or multiple objects exhibit inferior reconstruction quality. This is consistent with the observations in Table 2 that GIT always performs the best in term of MSE but may underperform in term of LPIPS which focuses more on the image structure. Therefore, instead of directly employing GIT for inference, we further utilize it as an image prior to guide optimization-based methods toward more perceptually accurate results.

Table 2: Quantitative comparison for different datasets and models in terms of different metrics. Dashed lines separate generative-based methods with optimization-based ones. The training time represents the time cost for training the generative model. The inference time represents the average time to reconstruct one input data instances from the leakage gradients during inference.

Dataset	Leaked Model	Method	MSE↓	PSNR↑	LPIPS↓	SSIM↑	Training Time (s)	Inference Time (s)
CIFAR10	LeNet	DLG	0.073	11.32	0.2380	0.0847	/	1660
		IG	0.082	11.27	0.3916	0.1193	/	1899
		LTI	0.015	19.17	0.2202	0.5304	8549	0.0030
		GIT	0.010	20.38	0.2663	0.5533	8071	0.0025
	ResNet	DLG	0.084	10.93	0.3813	0.0667	/	7474
		IG	0.080	10.75	0.2489	0.0739	/	6875
		LTI	0.035	15.32	0.4400	0.2888	5212	0.0020
		GIT	0.032	15.53	0.3957	0.3188	4019	0.0013
ImageNet	ResNet	DLG	0.147	9.25	0.8754	0.1324	/	3974
		IG	0.161	9.17	0.8802	0.1283	/	4103
		LTI	0.043	14.25	0.9017	0.3418	10200	0.0007
		GIT	0.039	14.42	0.8513	0.3507	13011	0.0008
	ViT	DLG	0.172	7.57	0.9513	0.1217	/	3734
		IG	0.175	7.64	0.9427	0.1210	/	3025
		LTI	0.046	13.37	0.9223	0.2117	9738	0.0029
		GIT	0.034	15.25	0.8365	0.3774	6717	0.0025

5.1.2 Optimization-Based Data Reconstruction Using Generative Models as Priors

Optimization-based methods are shown highly sensitive to the initialization of dummy inputs [8]. Therefore, recent methods, such as Gradient Inversion with Generative Image Prior (GIAS) [14], propose to utilize generative models to generate image priors as initialization of optimization-based methods. In this context, we can employ GIT to generate informative priors, which are subsequently refined through iterative optimization-based methods like IG. For generative methods [12, 14, 15, 16, 19, 17], we select GIAS [14] and LTI [17] as a representative baseline for comparison. To ensure fairness, all generative models are trained from scratch without relying on any pretraining, and are followed by the same optimization-based method.

As shown in Table 3, using GIT to generate priors and refine the reconstructed image by IG (GIT+IG) have the best performance in almost all cases and all metrics. In addition, GIT+IG always has better performance than LTI+IG, indicating GIT based on adaptive architectures can provide better priors than LTI based on fixed architectures.

Furthermore, GIT+IG, as a hybrid approach, not only converges faster but also outperforms both GIT and IG when used individually, demonstrating its superior effectiveness. We visualize the convergence curves of IG with and without the image prior in Appendix D.4, highlighting the benefits of incorporating generative priors into the optimization process.

Table 3: Quantitative comparison for different datasets and models in terms of different metrics. The performance of IG is used as references. The training time represents the time cost for training the generative model. The inference time represents the average time to reconstruct one input data instances from the leakage gradients during inference.

Dataset	Leaked Model	Method	MSE \downarrow	PSNR \uparrow	LPIPS \downarrow	SSIM \uparrow	Training Time (s)	Inference Time (s)
CIFAR10	LeNet	IG	0.082	11.27	0.3916	0.1193	/	1899
		GIAS+IG	0.009	21.45	0.0328	0.8925	10025	242
		LTI+IG	0.002	30.86	0.0025	0.9356	8549	158
		GIT+IG	0.001	31.25	0.0009	0.9551	8071	161
	ResNet	IG	0.080	10.75	0.2489	0.0739	/	6875
		GIAS+IG	0.019	18.96	0.2437	0.6125	10892	187
		LTI+IG	0.009	20.48	0.0092	0.8266	5212	1651
		GIT+IG	0.002	31.34	0.0041	0.9218	4019	1655
ImageNet	ResNet	IG	0.161	9.17	0.8802	0.1283	/	4103
		GIAS+IG	0.037	14.32	0.8218	0.3765	27453	3209
		LTI+IG	0.029	15.38	0.7434	0.4129	10200	2065
		GIT+IG	0.021	16.78	0.6995	0.4758	13011	1998
	ViT	IG	0.175	7.64	0.9427	0.1210	/	3025
		GIAS+IG	0.039	14.09	0.7572	0.5239	36950	3997
		LTI+IG	0.029	15.96	0.7250	0.4231	6138	2950
		GIT+IG	0.019	17.21	0.6730	0.5025	6717	2987

5.2 Reconstruction Under Challenging Situations

In this section, we investigate the robustness of reconstruction methods under different challenging situations, including inaccurate leaked gradients and the substantial distributional shift between the public data and the training data. In such situations, optimization-based methods are not applicable or do not have competitive performance. Therefore, we mainly compare the results from the direct inference by generative models. More details are deferred to Appendix C.

5.2.1 Inaccurate Gradients

Prior works [17] have shown degraded performance of optimization-based methods when the leaked gradients are inaccurate. Unlike optimization-based methods which observe significant performance degradation in the case of inaccurate gradients, Appendix D.1 shows that generative methods primarily utilize the gradient elements with large absolute values to generate outputs, indicating robustness in such challenging cases.

In Table 4, we consider the leaked gradients perturbed by isotropic Gaussian noise with different standard deviation (std). We compare different generative methods and include optimization-based methods like IG for reference. The results confirm the vulnerability of optimization-based methods against gradient perturbations. Among generative methods, GIT performs the best in all cases and all metrics, showing minimal susceptibility to inaccurate gradients.

5.2.2 Distribution Shift

As shown in Figure 1, GIT is trained on the public dataset injected by the attacker, and aims to reconstruct the local dataset. There may be a distributional shift between these two datasets, which could influence the effectiveness of the generative attack model.

We consider two possible scenarios of distribution shifts: (1) the public and local datasets come from different subsets of the same dataset, with distribution differences arising from overlapping but distinct classes; (2) the public datasets are subsets of huge but more general datasets, such as FER, while the local datasets held by individual clients are more specific ones, such as Jaffe.

Table 4: Comparison of metrics under gradient perturbation with varying noise variance. The batch size is fixed at 1, and the leaked model is LeNet with 5 layers.

Method	std of noise	MSE \downarrow	PSNR \uparrow	LPIPS \downarrow	SSIM \uparrow
IG	None	0.082	11.27	0.3916	0.1193
	0.01	0.105	9.79	0.4098	0.1172
	0.1	0.162	9.18	0.4320	0.1126
LTI	None	0.015	19.17	0.2202	0.5304
	0.01	0.015	19.16	0.2205	0.5300
	0.1	0.015	19.16	0.2199	0.5287
GIAS	None	0.012	19.21	0.2350	0.5398
	0.01	0.012	19.18	0.3010	0.5219
	0.1	0.013	18.87	0.3113	0.5189
GIT	None	0.010	20.38	0.2663	0.5533
	0.01	0.010	20.36	0.2675	0.5520
	0.1	0.010	20.37	0.2669	0.5522

Our experiment in Table 5 investigate both scenarios above. For CIFAR-10 and ImageNet, the public data and the local data share 6 classes and the rest classes are distinct. For facial dataset, the public data and the local data have different resolutions and significant distributional shifts. The results in Table 5 indicate that GIT demonstrates the strongest generalization ability across distribution shifts and achieves the best performance on the local dataset. GIT learns an implicit representation of the leaked model’s parameters by its adaptive architecture, which is more agnostic to the data distribution. By contrast, GIAS learns a latent space that captures the distribution characteristics of the public dataset, which requires the public and local datasets to share highly similar features to perform well.

Table 5: Comparison of the metrics under distributional shift. For each dataset, we select a federated learning model architecture that is well-suited for its classification task: LeNet is used for CIFAR-10, ResNet for ImageNet, and Vision Transformer (ViT) for facial datasets. The "classes" in the public data and local data represent categories sampled to form datasets. In the case of facial data, we conduct experiment where both the public data and local data come from FER, which serves as a comparison.

Dataset	Public Data	Local Data	Method	MSE \downarrow	PSNR \uparrow	LPIPS \downarrow	SSIM \uparrow
CIFAR10	classes 1-8	classes 3-10	GIAS	0.065	11.87	0.3670	0.3092
			LTI	0.029	15.38	0.3028	0.3790
			GIT	0.020	17.44	0.2155	0.4150
ImageNet100	classes 1-53	classes 48-100	GIAS	0.061	12.15	0.9518	0.3126
			LTI	0.049	13.10	0.9274	0.3150
			GIT	0.043	13.67	0.9044	0.3224
Facial Data	FER	FER	GIAS	0.020	17.73	0.4174	0.4051
			LTI	0.020	17.60	0.4420	0.3949
			GIT	0.018	17.93	0.3405	0.4228
		Jaffe	GIAS	0.042	13.77	0.5128	0.2826
			LTI	0.033	15.21	0.4625	0.3187
			GIT	0.030	15.54	0.3461	0.3244

5.2.3 Discrepancies in Model Parameters

In federated learning, gradient sharing may be asynchronous [9], leading to slight discrepancies in model parameters across different nodes. Such inconsistencies can affect the performance of both optimization-based and generative reconstruction methods.

To create discrepancies in model parameters, we train each node with different volume of local dataset for several epochs, then we use generative models trained on input-gradient pairs from one node, i.e., the node under attack, to reconstruct the input data from another node, which is not necessarily under attack and has parameter discrepancies. In this context, larger local data and more training epochs lead to larger parameter discrepancies, making the input data reconstruction task more challenging.

Our experimental results under different settings are summarized in Table 6. We can clearly observe that GIT achieves the best performance under significant parameter discrepancies and is still capable of reconstructing high-quality training data, demonstrating its robustness to variations in model parameters across nodes. We also find that increasing the number of local datasets sometimes leads to reduced performance degradation. This may be because a larger number of local datasets increases the likelihood of including samples from classes that are easier to reconstruct, thus introducing a degree of randomness that favors recovery.

Table 6: Comparison of the MSE for GIT with varying parameter discrepancies. The parameter discrepancies is quantified by volume of local dataset & number of locally trained epochs. The leaked model for CIFAR10 is LeNet, and for ImageNet is ResNet.

Dataset	Volume of Local Dataset	Method	Number of Locally Trained Epochs		
			0	10	20
CIFAR10	500	IG	0.082	0.089	0.096
		LTI	0.015	0.019	0.023
		GIT	0.010	0.013	0.017
	1000	IG	0.082	0.097	0.102
		LTI	0.015	0.026	0.030
		GIT	0.010	0.017	0.020
	10000	IG	0.082	0.096	0.107
		LTI	0.015	0.032	0.036
		GIT	0.010	0.029	0.034
ImageNet	500	IG	0.161	0.162	0.162
		LTI	0.043	0.049	0.053
		GIT	0.039	0.043	0.044
	1000	IG	0.161	0.162	0.163
		LTI	0.043	0.049	0.053
		GIT	0.039	0.043	0.043
	10000	IG	0.161	0.164	0.164
		LTI	0.043	0.048	0.051
		GIT	0.039	0.040	0.040

5.3 Ablation Studies

Training data volume refers to the size of the auxiliary dataset sampled by the attacker from public data. A larger training dataset, akin to performing data augmentation, can enhance the generalization ability of the generative model. However, it also incurs higher computational costs, requiring more time and resources to train the model. Moreover, in practice, acquiring a large volume of data with a distribution similar to the local dataset can be challenging. Given this tradeoff, it is essential to evaluate the performance of the generative approach under different training data volumes. In this section, we evaluate the performance of GIT using varying amounts of training data: 1,000, 2,000, 5,000 and 10,000 samples. Table 7 presents the impact of training data volume on the performance of the generative approach. It demonstrates that even with only 1,000 input-gradient pairs, GIT is capable of reconstructing reasonable images, indicating that effective recovery is achievable with a limited amount of training data. As the

Table 7: MSE comparison with varying volumes of training data. The dataset is CIFAR10 and the leaked model is LeNet.

Data Volume	LTI	GIAS	GIT
1000	0.039	0.049	0.027
2000	0.033	0.045	0.021
5000	0.027	0.036	0.015
10000	0.015	0.027	0.010

generative model achieves near-perfect performance with larger training sets, we can conclude that increased data volume helps mitigate overfitting and consequently improves model performance. Moreover, GIT consistently outperforms both LTI and GIAS across all settings.

More analyses and ablation studies are deferred to Appendix D.

6 Conclusions

This work introduces *Generative Gradient Inversion Transcript (GIT)*, a novel method for reconstructing training data in federated learning by exploiting gradient leakage. We propose a reconstruction framework based on generative models, leveraging an adaptive structure inspired by the inverse of backpropagation. Our attack model realistically assumes that an attacker can inject data into a compromised client, enabling the reconstruction of local datasets from both compromised and non-compromised clients. This setup aligns with practical adversarial scenarios in federated learning. GIT offers a significant efficiency advantage, being more cost-effective in inference than optimization-based methods, and can be seamlessly deployed after offline training. Compared to existing methods, GIT achieves superior performance in most cases. Furthermore, GIT-generated outputs can serve as priors for optimization-based gradient matching approaches, further enhancing attack effectiveness. GIT demonstrates strong generalization and robustness under challenging conditions, including inaccurate gradients, distributional shifts, and discrepancies in model parameters across clients.

References

- [1] Arthur Jochems, Timo M Deist, Johan Van Soest, Michael Eble, Paul Bulens, Philippe Coucke, Wim Dries, Philippe Lambin, and Andre Dekker. Distributed learning: developing a predictive model based on data from multiple hospitals without data leaving the hospital—a real life proof of concept. *Radiotherapy and Oncology*, 121(3):459–467, 2016.
- [2] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.
- [3] Wensi Yang, Yuhang Zhang, Kejiang Ye, Li Li, and Cheng-Zhong Xu. Ffd: A federated learning based method for credit card fraud detection. In *Big Data–BigData 2019: 8th International Congress, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 8*, pages 18–32. Springer, 2019.
- [4] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 7232–7241. Curran Associates, Inc., 2021.
- [5] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning: Revisited and enhanced. In *Applications and Techniques in Information Security: 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6–7, 2017, Proceedings*, pages 100–110. Springer, 2017.
- [6] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [7] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.
- [8] Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. A framework for evaluating gradient leakage attacks in federated learning. *arXiv preprint arXiv:2004.10397*, 2020.
- [9] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients—how easy is it to break privacy in federated learning? *Advances in neural information processing systems*, 33:16937–16947, 2020.
- [10] Junyi Zhu and Matthew Blaschko. R-gap: Recursive gradient attack on privacy. *arXiv preprint arXiv:2010.07733*, 2020.
- [11] Yijue Wang, Jieren Deng, Dan Guo, Chenghong Wang, Xianrui Meng, Hang Liu, Caiwen Ding, and Sanguthevar Rajasekaran. Sapag: A self-adaptive privacy attack from gradients. *arXiv preprint arXiv:2009.06228*, 2020.
- [12] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16337–16346, 2021.

- [13] Zihan Wang, Jason Lee, and Qi Lei. Reconstructing training data from model gradient, provably. In *International Conference on Artificial Intelligence and Statistics*, pages 6595–6612. PMLR, 2023.
- [14] Jinwoo Jeon, Kangwook Lee, Sewoong Oh, Jungseul Ok, et al. Gradient inversion with generative image prior. *Advances in neural information processing systems*, 34:29898–29908, 2021.
- [15] Zhuohang Li, Jiaxin Zhang, Luyang Liu, and Jian Liu. Auditing privacy defenses in federated learning via generative gradient leakage. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10132–10142, 2022.
- [16] Hao Fang, Bin Chen, Xuan Wang, Zhi Wang, and Shu-Tao Xia. Gifd: A generative gradient inversion method with feature domain optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4967–4976, 2023.
- [17] Ruihan Wu, Xiangyu Chen, Chuan Guo, and Kilian Q Weinberger. Learning to invert: Simple adaptive attacks for gradient inversion in federated learning. In *Uncertainty in Artificial Intelligence*, pages 2293–2303. PMLR, 2023.
- [18] Huancheng Chen and Haris Vikalo. Recovering labels from local updates in federated learning. *arXiv preprint arXiv:2405.00955*, 2024.
- [19] Liwen Wu, Zhizhi Liu, Bin Pu, Kang Wei, Hangcheng Cao, and Shaowen Yao. Dggi: Deep generative gradient inversion with diffusion model. *Information Fusion*, 113:102620, 2025.
- [20] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [21] 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*, pages 308–318, 2016.
- [22] Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. *Advances in Neural Information Processing Systems*, 35:22911–22924, 2022.
- [23] Kailang Ma, Yu Sun, Jian Cui, Dawei Li, Zhenyu Guan, and Jianwei Liu. Instance-wise batch label restoration via gradients in federated learning. In *The Eleventh International Conference on Learning Representations*, 2023.
- [24] Dimitar I Dimitrov, Maximilian Baader, Mark Müller, and Martin Vechev. Spear: Exact gradient inversion of batches in federated learning. *Advances in Neural Information Processing Systems*, 37:106768–106799, 2024.
- [25] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [27] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [29] Michael Lyons, Miyuki Kamachi, and Jiro Gyoba. The japanese female facial expression (jaffe) dataset. (*No Title*), 1998.
- [30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [32] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

A Notation

\mathcal{L}	Loss objective function of the leaked model
$\mathcal{L}^{(gen)}$	Loss objective function of GIT
$\sigma(\cdot)$	The elementwise activation function
$\sigma'(\cdot)$	The derivative of $\sigma(\cdot)$
z	Pre-activation of an MIMO layer
a	Post-activation of an MIMO layer
\mathbf{W}	Weight matrix in a neural network
\mathbf{g}	Gradient of the loss objective function w.r.t \mathbf{W}
B	Batch size in mini-batch training
b	Batch index in mini-batch training
\mathbb{E}	Empirical average over the samples in batch b
d	Number of hidden nodes of an MIMO layer
N_{in}	Number of input layers of an MIMO layer
N_{out}	Number of output layers of an MIMO layer
N	Number of input-gradient pairs for training GIT
N'	Number of input-gradient pairs for testing
$(\cdot)^+$	Moore-Penrose inverse of a matrix; or Moore-Penrose of each of (\cdot) 's subspace via the first dimension when (\cdot) is a third order tensor
$(\cdot)^T$	Transpose of a matrix; or transpose of the second and the third dimension when (\cdot) is a third order tensor
\otimes	Tensor Multiplication
\odot	Broadcast row-wise product
$f(\cdot)^{(in)}$	A module that approximates the input mapping of an MIMO layer
$f(\cdot)^{(out)}$	A module that approximates the output mapping of an MIMO layer
$f'(\cdot)$	The derivative of module $f(\cdot)$
\mathcal{D}	Input-gradient pairs
E	Epoch budget
η	Learning rate
$(\cdot)^{(i)}$	The i -th sample in the dataset
M	The generative model GIT
Θ	Trainable parameters of GIT
m_θ	A shallow MLP parameterized by θ to approximate recursive reconstruction in Coarse-GIT
ϑ	Trainable parameters in Module-GIT
\mathbf{x}	Input data of the leaked model
$\hat{\mathbf{x}}$	The estimated input data by GIT

B Methodology Details

B.1 Exact-GIT

Activation Function. The Exact-GIT method in Algorithm 1 requires iteratively applying Equation (3). Equation (3) involves the derivative of the activation function $\sigma'(z)$, which can be estimated by a . Although function σ may not be an injective function, we demonstrate in Table 8 below that we can uniquely identify $\sigma'(z)$ given a for the most popular activation functions used in practice. In Exact-GIT, the weights of the generative attack model represent the estimated

Table 8: Mappings from a to $\sigma'_i(z)$ for popular activation functions. Operations are elementwise.

Name	ReLU	Leaky ReLU	Sigmoid	Tanh
$a = \sigma(z)$	$\max(0, z)$	$\max(kz, z)$	$\frac{1}{1+e^{-z}}$	$\frac{e^z - e^{-z}}{e^{2z} + e^{-z}}$
$\sigma'(z)$	$\begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{if } a = 0 \end{cases}$	$\begin{cases} 1 & \text{if } a > 0 \\ k & \text{if } a \leq 0 \end{cases}$	$a_i(1 - a)$	$1 - a^2$

weights of the leaked model. Therefore, we can compare the difference between their weights to investigate to which degree the generative attack models recover the gradient-to-input inversion. In this context, we run Exact-GIT based on Algorithm 1 and plot its convergence curve as in Figure 3. Figure 3 illustrates the l_2 distance curve between the generative model's weights and the leaked model's weights, alongside the MSE between the reconstructed inputs and the ground truth inputs. As shown in Figure 3, when Exact-GIT converges, its weights align closely with the ground truth weights. This convergence highlights the effectiveness of exact-GIT in extracting weight information from the leaked model.

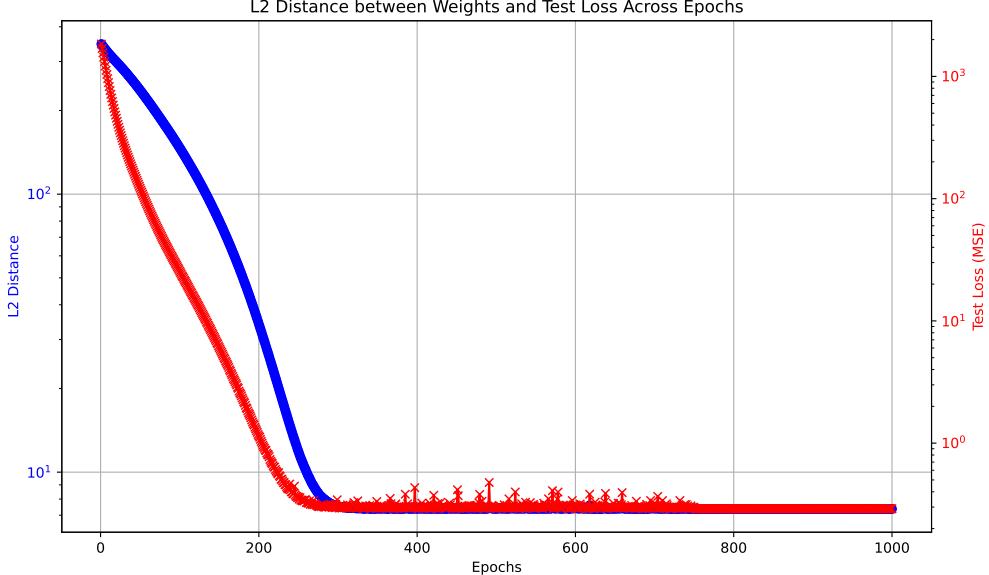


Figure 3: The red curve represents convergence curve of l_2 distance between weights of the generative model and the leaked model. The blue curve represents the convergence curve of MSE between reconstructed input and the ground truth input. The experiment is conducted on CIFAR-10 using Exact-GIT.

B.2 Coarse-GIT for Different Architectures

Section 3 demonstrates a generic framework for any neural network architectures as long as they support back-propagation. In this section, we provide more technical details for specific neural network architecture that we use in the experiments, including feed forward networks, residual networks (ResNet) and vision transformer (ViT). We believe the details in this section will provide more insights for practitioners to understand how GIT is adapted to different neural architectures. Due to the scale of the architectures discussed in this section, we employ Coarse-GIT for all of them.

While related to the notation we use in Section 3, we use specific notations in this section for better readability. We provide the exact definition for each of these notations.

B.2.1 Feed Forward Neural Networks

Feed forward neural networks, including multi-layer perceptron (MLP) and convolutional neural networks (CNN), can be formulated as follows:

$$\begin{aligned}\mathcal{L}_\theta(\mathbf{x}, y) &= \ell(\mathbf{z}_N, y) = \ell(\mathbf{W}_N \mathbf{a}_{N-1}, y) \\ \mathbf{a}_i &= \sigma_i(\mathbf{z}_i), \quad \mathbf{z}_i = \mathbf{W}_i \mathbf{a}_{i-1}, \quad i = 1, 2, \dots, N-1\end{aligned}\tag{6}$$

We denote the number of hidden nodes for the i -th layer as $\{d_i\}_{i=1}^{N-1}$. The input data batch $\mathbf{a}_0 = \mathbf{x} \in \mathbb{R}^{B \times d_0}$, where B is the batch size. $\theta = \{\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}\}_{i=1}^N$ refer to the parameters of N linear layers, including convolutional layers and fully connected layers. $\{\sigma_i\}_{i=1}^{N-1}$ are the nonlinear activation functions of different layers. In this context, $\{\mathbf{z}_i \in \mathbb{R}^{B \times d_i}\}_{i=1}^{N-1}$ and $\{\mathbf{a}_i \in \mathbb{R}^{B \times d_i}\}_{i=1}^{N-1}$ represent the pre-activation and post-activation of intermediate layers, respectively. $\mathbf{z}_N = \mathbf{W}_N \mathbf{a}_{N-1}$ is the output logit, and ℓ is the function calculating the classification error, such as the softmax cross-entropy function. We use $\mathbf{g}_i = \nabla_{\mathbf{W}_i} \mathcal{L}_\theta(\mathbf{x}, y)$ to represent the gradient of each weight matrix.

In this context, similar to Equation (2) and Equation (3) in Section 3, we derive the back-propagation and then the iterative input layer approximation for feed forward neural network defined in Equation (6) as follows:

$$\mathbf{g}_i = \prod_{j=i}^{N-1} (\mathbf{W}_{j+1}^T \odot \sigma'_j(\mathbf{z}_j)) \otimes \frac{\partial \mathcal{L}}{\partial \mathbf{z}_N} \otimes \mathbf{a}_{i-1}^T \tag{7}$$

$$\mathbf{a}_{i-1}^T \simeq \mathbf{a}_i^T \otimes \mathbf{g}_{i+1}^T \otimes (\mathbf{W}_{i+1}^T \odot \sigma'_i(\mathbf{z}_i))^+ \otimes \mathbf{g}_i \tag{8}$$

\otimes and \odot have the same definition as in Section 3. As we can see, Equation (7) can be considered as a specific case of Equation (3) where $N_{in} = N_{out} = 1$. Furthermore, we employ Coarse-GIT in the experiments. Specifically, we use an MLP model f parameterized by ϑ to estimate \mathbf{a}_{i-1} from \mathbf{a}_i , \mathbf{g}_{i+1} and \mathbf{g}_i . We apply Equation (7) recursively and utilize it to reconstruct the input data.

$$\mathbf{a}_{i-1} = f_\vartheta(\mathbf{a}_i, \mathbf{g}_{i+1}, \mathbf{g}_i) \tag{9}$$

B.2.2 Residual Networks

The key feature for residual networks (ResNet) [31] is the skip connections, resulting in $N_{out} > 1$ for layers that combine inputs from both the previous layer and shortcut connections.

Without the loss of generality, we generally follow the notation of feed forward neural network defined in (6) except that there is a single shortcut connection linking the k -th layer to l -th layer ($k < l$). Specifically, the shortcut connection links the post-activation \mathbf{a}_k to the pre-activation \mathbf{z}_l with a weight parameter $\mathbf{S} \in \mathbb{R}^{d_k \times d_l}$. Therefore, $\{\mathbf{z}_i\}_{i=1}^N$ and $\{\mathbf{a}_i\}_{i=1}^N$ are calculated in the same manner except that $\mathbf{z}_l = \mathbf{W}_l \mathbf{a}_{l-1} + \mathbf{S} \mathbf{a}_k$. Based on the back propagation, \mathbf{g}_i is calculated in the same way as in Equation (7) when $i > k$. When $i \leq k$, \mathbf{g}_i is calculated as follows:

$$\mathbf{g}_i = \prod_{j=i}^{k-1} M_j \otimes \left(\prod_{j=k}^{l-1} M_j + \mathbf{S} \odot \sigma'_k(\mathbf{z}_k) \right) \otimes \prod_{j=l}^{N-1} \left(\mathbf{W}_{j+1}^T \odot \sigma'_j(\mathbf{z}_j) \right) \otimes \frac{\partial \mathcal{L}}{\partial \mathbf{z}_N} \otimes \mathbf{a}_{i-1}^T \tag{10}$$

Following a similar analysis to feed forward neural networks, we can derive an approximation of \mathbf{a}_{i-1} using \mathbf{a}_i . The approximation is the same as (8) except for the case $i = k$. This is because we calculate \mathbf{a}_i using \mathbf{a}_{i-1} in the same manner except for the case $i = k$, where the k -th layer is connected to not only the immediate preceding layer but also the l -th layer via shortcut connection. Therefore, \mathbf{a}_{k-1} is approximated in a different way from (8) as follows.

$$\mathbf{a}_{k-1} \simeq (\mathbf{W}_{k+1}^T \odot \sigma'_k(\mathbf{z}_k)) \otimes \mathbf{g}_{k+1} \otimes (\mathbf{a}_k^T)^+ + (\mathbf{S} \odot \sigma'_k(\mathbf{z}_k)) \otimes \mathbf{g}_l \otimes (\mathbf{a}_{l-1})^+ \otimes \mathbf{g}_k \tag{11}$$

Compared with (8), the estimation in (11) incorporates not only \mathbf{g}_k and \mathbf{g}_{k+1} but also \mathbf{g}_l to estimate \mathbf{a}_{k-1}^T , which is consistent with the case of $N_{out} = 2$ in the analysis in Section 3. Since \mathbf{a}_k is connected to \mathbf{z}_l via skip connection, gradients can flow directly from the k -th layer to the l -th layer in back propagation. The insight provided by the approximation in Equation (11) indicates that the reconstruction sequence follows the same path as the gradient flow during backpropagation.

We use Coarse-GIT in the experiment for ResNet, similar to Equation (9), we employ an MLP model f parameterized by ϑ and reconstruct \mathbf{a}_{k-1} by:

$$\mathbf{a}_{k-1} = f_\vartheta(\mathbf{a}_k, \mathbf{g}_{k+1}, \mathbf{g}_k, \mathbf{g}_l) \tag{12}$$

When estimating the input from the leaked gradients, we apply Equation (12) when there is a shortcut connection and Equation (9) otherwise.

B.2.3 Vision Transformer (ViT)

In the case of vision transformer (ViT) [32], we apply modularized input data reconstruction and represent the each self-attention module as follows:

$$\mathbf{z} = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right) \mathbf{V}, \quad \mathbf{Q} = \mathbf{a}_i^{(in)} \mathbf{W}^Q, \quad \mathbf{K} = \mathbf{a}_i^{(in)} \mathbf{W}^K, \quad \mathbf{V} = \mathbf{a}_i^{(in)} \mathbf{W}^V \quad (13)$$

where \mathbf{W}^Q , \mathbf{W}^K and \mathbf{W}^V represent the mapping weights to the tuple of query, key and value. In multi-head attention (MHA), we concatenate the outputs of several self-attention modules and transform them by an affine operation. Without the loss of generality, we focus on single layer attention. Furthermore, we reorganize Equation (13) to fit the formulation of Equation (4):

$$\begin{aligned} \mathbf{z} &= f_i^{(in)}([\mathbf{Q}, \mathbf{K}, \mathbf{V}]) := \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right) \mathbf{V} \\ [\mathbf{Q}, \mathbf{K}, \mathbf{V}] &= \mathbf{a}_i^{(in)} \mathbf{W}_i^{(in)} := \mathbf{a}_i^{(in)} [\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V] \end{aligned} \quad (14)$$

Equation (13) identifies the concrete definitions of $f^{(in)}$ and $\mathbf{W}_i^{(in)}$ for self-attention modules in the framework by Equation (4) so that we can plug these definitions employ Equation (5) to reconstruct the input of the attention layer by the leaked gradients.

Due to the large amount of parameters in ViT, we use Coarse-GIT for input reconstruction. If we use \mathbf{g}^Q , \mathbf{g}^K , \mathbf{g}^V to represent the leaked gradients of \mathbf{W}^Q , \mathbf{W}^K and \mathbf{W}^V , respectively, then we employ an MLP module f parameterized by ϑ to reconstruct $\mathbf{a}_i^{(in)}$ in a self-attention module.

$$\mathbf{a}_i^{(in)} = f_\vartheta(\mathbf{g}^Q, \mathbf{g}^K, \mathbf{g}^V, \{\mathbf{g}_j^{(out)}\}_{j=1}^{N_{out}}, \mathbf{z}) \quad (15)$$

where $\{\mathbf{g}_j^{(out)}\}_{j=1}^{N_{out}}$ are the leaked gradients of output matrices as defined for a general MIMO layer in Section 3. The gradient inversion of fully-connected layers and residual structure in the ViT follows the same formulation as described in previous sections. Altogether, we can iteratively employ these formulas to reconstruct the input estimation of each layer, starting from the last layer and progressing to the first layer of the ViT model, eventually obtaining the input data estimation.

C Experiment Configurations

Universal Settings We employ various architectures for the leaked model. For LeNet, we use a 5-layer configuration with kernel size 2 and same padding. For ResNet, we adopt a 15-layer variant with kernel size 3, consisting of 4 blocks, each containing 2 convolutional layers and 1 skip connection. For ViT, we connect four 4-head attention blocks following the patch embedding layer.

For generative methods, we use 10000 batches of input-gradient pairs from the public dataset to train the generative model. During reconstruction, we use 10000 batches of gradients from the local dataset to recover the corresponding local data. For iterative optimization-based methods, we perform reconstruction for each batch of local data by starting from dummy inputs and applying iterative optimization individually.

For Coarse-GIT and Module-GIT, we use m_θ and f_ϑ with 3000 neurons in each hidden layer. For LTI, we employ a generative model consisting of three hidden layers, each with 3000 neurons, as described in [17].

Specific Settings In our experiments described in Section 5.1.1, the inference time for generative methods is computed as the average over reconstructing 10000 local data batches, whereas for optimization-based methods, it is calculated based on the average over 10 local data batches, since each reconstruction is significantly more time-consuming.

In our experiments described in Section 5.1.2, the inference time of generative+optimization-based hybrid methods is computed as the sum of their individual inference times. For all methods, inference time is calculated as the average over 10 local data batches.

D More Experimental Analyses and Ablation Studies

D.1 Reconstruction with Clipped Gradients

The prune rate γ represents the proportion of gradient directions with small absolute values that are pruned (Pruning is applied by a mask with 0 and 1 values, therefore the dimension of gradients is not changed). As shown in Table 9,

gradient pruning has minimal impact on GIT’s performance but significantly degrades the performance of DLG. The results indicate that even when pruning 90% of the gradient components with smaller absolute values, generative approaches remain largely unaffected, relying only on the top 10% of the largest gradient values for training. This suggests that generative approaches primarily capture the dominant gradient components with large absolute values during training, unlike optimization-based methods, which require a finer alignment with the full gradient information.

It can be deduced from Table 9 that generative approaches are less effective than optimization-based methods in recovering fine-grained image details. However, they demonstrate greater robustness against inaccurate gradients and gradient pruning while also being more efficient. Furthermore, generative methods train significantly faster, as gradient matching requires extensive computation to precisely align finer gradient details, leading to higher time complexity.

Table 9: Comparison of the MSE under gradient pruning with varying pune rate. The dataset is CIFAR10 and the leaked model is LeNet.

Prune rate γ	DLG	LTI	GIAS	GIT
0	0.073	0.015	0.027	0.010
0.9	0.098	0.016	0.033	0.010
0.99	0.116	0.021	0.049	0.016
0.999	0.187	0.049	0.070	0.040

D.2 Effect of Noise on the Performance of Optimization-Based Reconstruction Methods

Under inaccurate gradients, generative approaches demonstrate robust performance, as shown in Table 4. However, IG fails to recover meaningful data with as little as 0.01 noise applied. This highlights the significant impact of noise on gradient matching methods like DLG. In the contrast, IG fails to recover meaningful data with as little as 0.01 noise applied. This highlights the significant impact of noise on optimization-based methods like IG.

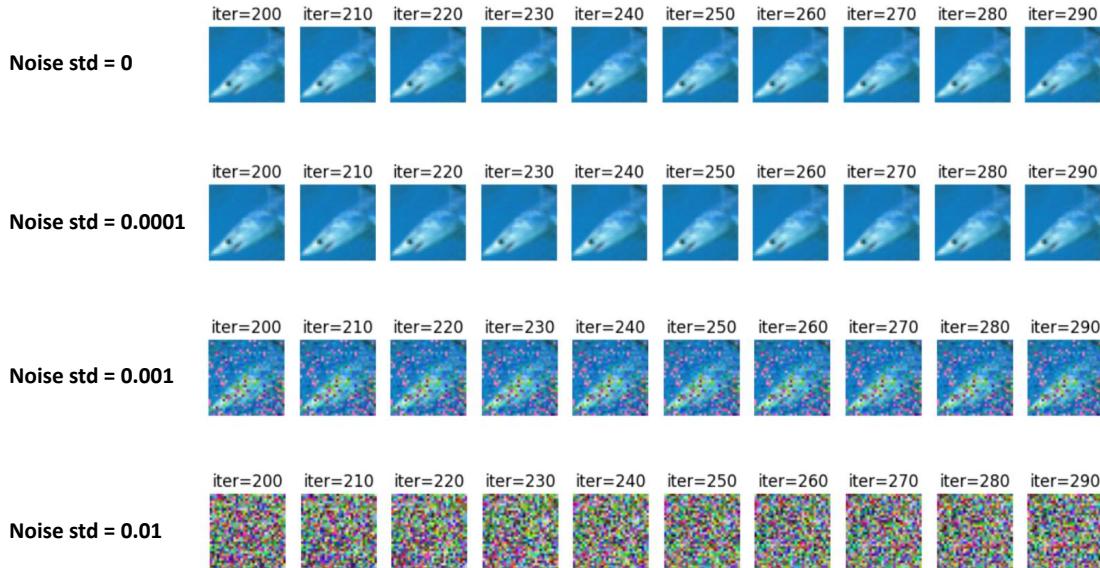


Figure 4: The figure illustrates the reconstructed images for IG when the leaked model is LeNet and the dataset is CIFAR-10. Varying levels of noise are applied to the gradients. The results depict IG’s reconstructions between the 200th and 300th optimization iterations.

D.3 Reconstruction without gradient of last layer's bias

When the last layer of the neural network has a bias term \mathbf{b}_N , i.e., $\mathbf{a}_N = \mathbf{W}_N \mathbf{a}_{N-1} + \mathbf{b}_N$, following the idea of [23], we have $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_N} = \frac{\partial \mathcal{L}}{\partial \mathbf{b}_N}$. That is to say, we can directly utilize the gradient of the bias term in the last year as $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_N}$. When the last layer of the neural network does not have a bias term, we cannot directly obtain $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_N}$. Therefore, we employ ablation study using the leaked gradients for the weight of the last layers to estimate the output logits by a shallow MLP.

Table 10: Quantitative comparison for GIT with and w/o gradient of last layer's bias.

Dataset	Leaked Model	Method	MSE \downarrow	PSNR \uparrow	LPIPS \downarrow	SSIM \uparrow
CIFAR10	LeNet	with bias	0.010	20.38	0.2663	0.5533
		w/o bias	0.012	19.35	0.2879	0.5035
Imagenet	Resnet	with bias	0.039	14.42	0.8513	0.3507
		w/o bias	0.039	14.30	0.9017	0.3120

D.4 Optimization-based methods with and w/o Generated Image Prior

As shown in the Figure 5, the blue curve represents the convergence of the hybrid method, while the red curve illustrates IG without an image prior. It is evident that the hybrid method not only converges faster but also achieves superior performance. The fluctuations in the blue convergence curve are due to the small learning rate set for the optimizer, which causes oscillations when the loss falls below 1.

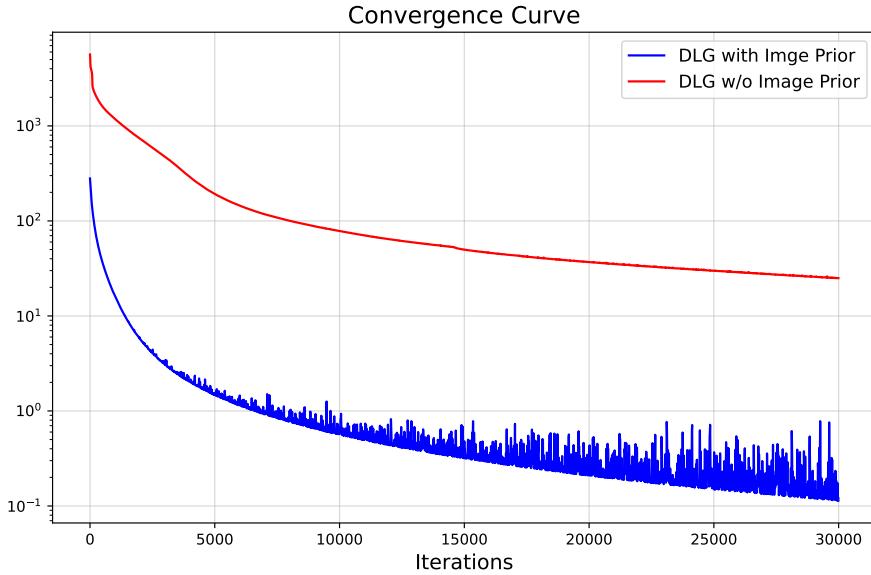


Figure 5: The convergence curve of DLG with and without an image prior. The leaked model is ResNet. The vertical axis indicates the distance between the dummy gradients and the corresponding ground-truth gradients.

D.5 Visual Results

D.5.1 Visual Results on CIFAR-10 and Tiny ImageNet

Figure 6 illustrates reconstructed CIFAR-10 and Tiny ImageNet direct using GIT for reconstruction or using GIT as generated image prior. These results show that directly using GIT for reconstruction can achieve reasonable recovery but tends to lose some high-frequency details. In contrast, using GIT as an image prior—specifically for initializing optimization-based methods—helps preserve high-frequency information and achieves reconstruction quality beyond what optimization-based methods alone can attain.

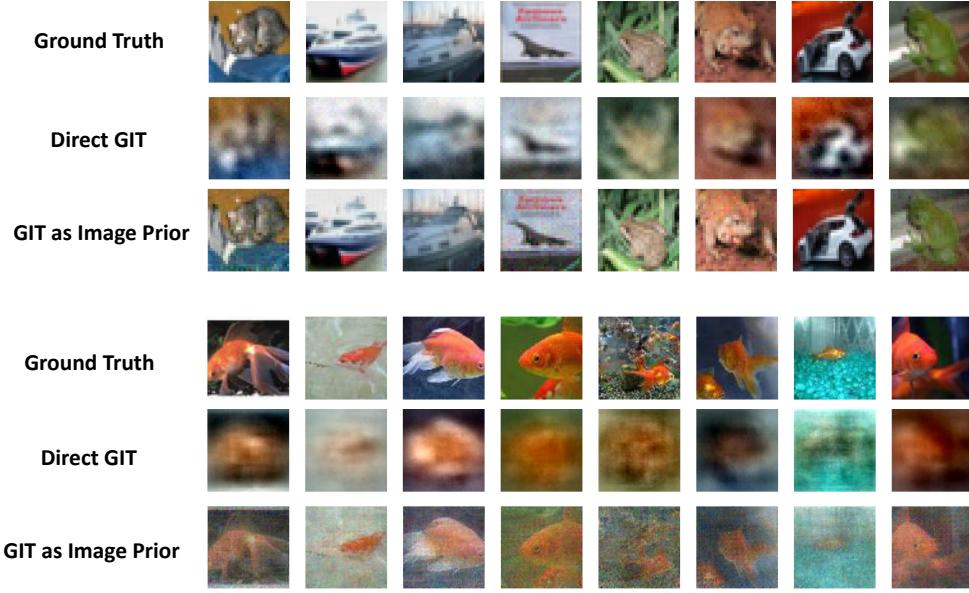


Figure 6: The figure illustrates the **ground truth input images**, the **direct reconstructed images by GIT** and the **reconstructed images using IG initialized with GIT-generated prior**, from top to the bottom respectively. The top three rows correspond to the CIFAR-10 dataset with the leaked model being LeNet, while the bottom three rows correspond to the TinyImageNet dataset with the leaked model being ResNet.

D.5.2 Visual Results for GIT on Large Resolution

Reconstruction at high resolution tends to be more challenging, especially for images containing complex objects. To illustrate the characteristics of both easy and hard-to-recover examples, we present the first 8 and the best 100 reconstructions. Odd-numbered rows show the ground-truth images, while even-numbered rows display the corresponding reconstructions obtained directly using GIT.



Figure 7: The first 8 reconstructed images (ImageNet, ResNet). Odd-numbered rows show the ground-truth images, while even-numbered rows display the corresponding reconstructions obtained directly using GIT.

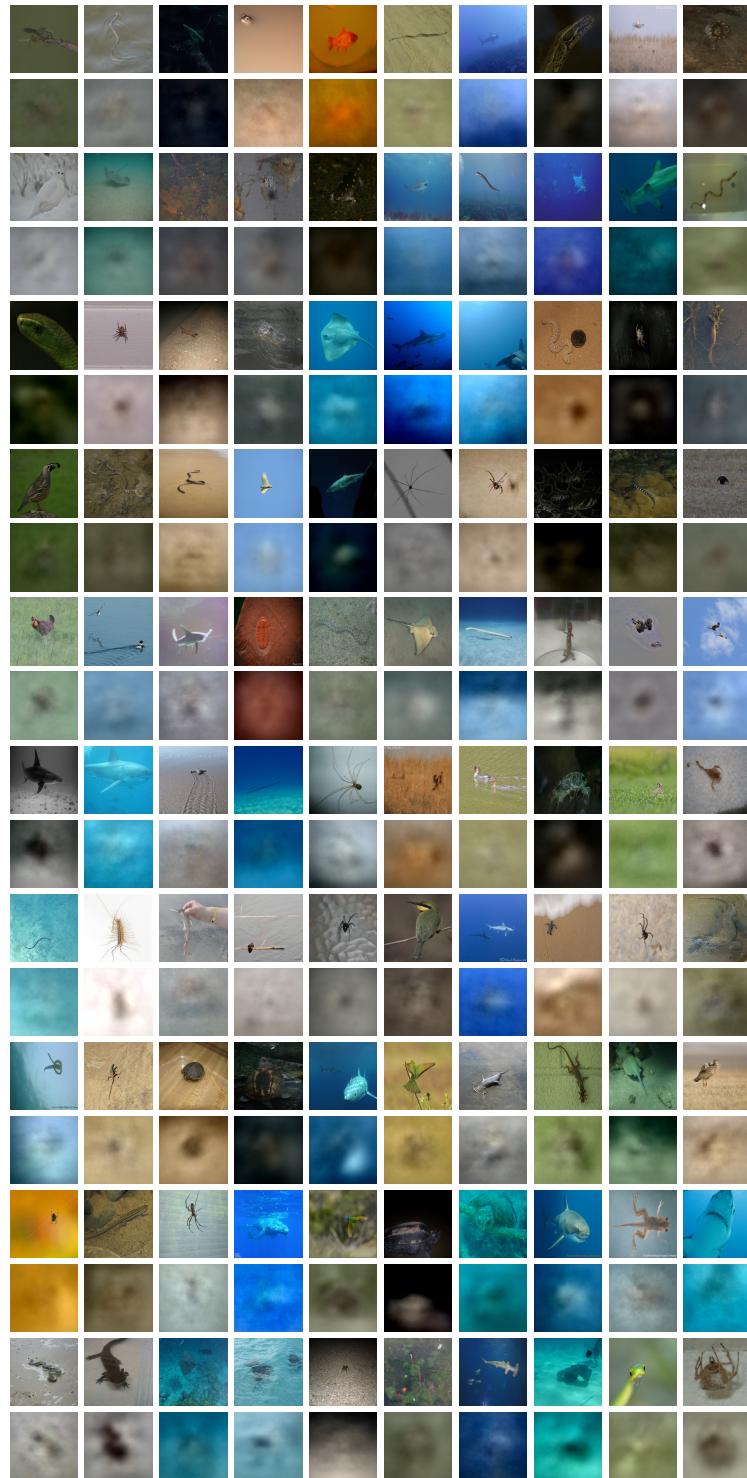


Figure 8: The best 100 reconstructed images with lowest MSE (ImageNet, ResNet). Odd-numbered rows show the ground-truth images, while even-numbered rows display the corresponding reconstructions obtained directly using GIT.