Gradient Inversion Attack on Graph Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Graph federated learning is of essential importance for training over large graph datasets while protecting data privacy, where each client stores a subset of local graph data, while the server collects the local gradients and broadcasts only the aggregated gradients. Recent studies reveal that a malicious attacker can steal private image data from gradient exchanging of neural networks during federated learning. However, none of the existing works have studied the vulnerability of graph data and graph neural networks under such attack. To answer this question, the present paper studies the problem of whether private data can be recovered from leaked gradients in both node classification and graph classification tasks and proposes a novel attack named Graph Leakage from Gradients (GLG). Two widely-used GNN frameworks are analyzed, namely GCN and GraphSAGE. The effects of different model settings on recovery are extensively discussed. Through theoretical analysis and empirical validation, it is shown that parts of the graph data can be leaked from the gradients.

1 Introduction

Federated Learning McMahan et al. (2016b) is a distributed learning paradigm that has gained increasing attention. Various organizations in healthcare rely on federated learning to train models while also keeping user data private. Gradient averaging is a widely used mechanism in federated learning, where clients send the gradients of the models instead of the actual private data to a central server. The server then updates the model by taking the average gradient over all the clients. The computation is executed independently on each client and synchronized via exchanging gradients between the server Li et al. (2014); Iandola et al. (2016) and the clients Patarasuk & Yuan (2009). It enables the training of a global model using data distributed at multiple sources without the need to send them to a central location. This scheme is widely used when the training set is large and/or contains private information Jochems et al. (2016); Konečný et al. (2016). For example, multiple hospitals train a model jointly without sharing their patients' medical records Jochems et al. (2017); McMahan et al. (2016a).

Recently, several efforts have demonstrated that gradients can leak input data through gradient inversion attacks. Jeon et al. (2021); Zhu et al. (2019); Geiping et al. (2020); Zhao et al. (2020b); Yin et al. (2021); Zhu & Blaschko (2020); Jin et al. (2021) successfully reconstruct the users' private image data leveraging the gradients. Specifically, Zhao et al. (2020a) proposes that the ground truth labels can be directly inferred from the gradients. Consequently, ensuing works improve the attack techniques on large batches of user data Yin et al. (2021); Fowl et al. (2022b). A few recent works study recovering text data in federated learning. To facilitate the text recovery, Boenisch et al. (2021); Fowl et al. (2022a) recover text data using a strong threat model in which the server is malicious and is able to manipulate the training model's weights. In contrast, Gupta et al. (2022) studies the feasibility of recovering text from large batch sizes by leveraging the memorization capability of Language Models during federated learning. It can be seen that the majority of the works are based on either image or text data.

On another note, Graph Neural Networks (GNNs) have demonstrated state-of-the-art results when applied to structured data. From learning meaningful node representations over large social networks Hamilton et al. (2017), to predicting molecular property Gómez-Bombarelli et al. (2018) Yang et al. (2019), GNNs have been widely adopted across various paradigms. The confluence of Federated Learning and graphs has led to the emergence of Federated Graph Learning He et al. (2021) Scardapane et al. (2020) Zhang et al. (2021).

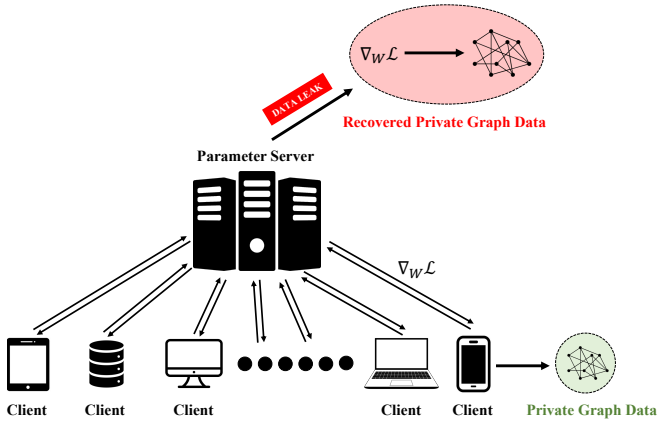


Figure 1: Gradient inversion attack in Federated Graph Learning

Despite the notable success of gradient inversion attacks for general deep neural networks, **none of existing works studied the vulnerability of GNNs towards gradient inversion attacks.** Hence, the present work aims to answer the following unexplored question:

Can graph data be leaked from gradients of GNNs?

A major challenge in recovering graph data is the intertwined nature of the nodal features and the irregular graph structures, making it difficult to separate the information from the two sources. In other words, representations are correlated, and the underlying graph structure influences the correlation. In this work, we first provide a theoretical analysis of the feasibility of recovering graph data based on gradients. Motivated by this observation, we propose a gradient inversion attack called GLG (Graph Leakage From Gradients). We investigate two commonly used GNN frameworks, i.e., Graph Convolutional Networks (GCN) Kipf & Welling (2016), and GraphSAGE Hamilton et al. (2017), and examine their vulnerabilities to gradient-based attacks for both node and graph classification tasks. An illustration of the attack scenario is presented in Figure 1, showcasing the process in which a curious central parameter server attempts to reconstruct the graph data utilizing the gradients of the model obtained from a client. **To our best knowledge, this is the first work that studies gradient-based attacks for graph data.** Our main contributions are summarized as follows:
i) Theoretical analysis highlighting the vulnerabilities of the GNN frameworks towards gradient-based attack.
ii) Developing novel gradient inversion attack mechanisms to recover graph data for both node and graph classification tasks.
iii) Experiments are conducted in a variety of settings with different amounts of prior knowledge.

2 Preliminaries

Given a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} := \{v_1, v_2, \cdots, v_N\}$ represents the node set and $\mathcal{E} \subseteq N \times N$ denotes the edge set. Matrices $\mathbf{X} \in \mathbb{R}^{N \times D}$ and $\mathbf{A} \in \{0,1\}^{N \times N}$ represent the node feature and adjacency matrix respectively, where $\mathbf{A}_{ij} = 1$ if and only if $\{v_i, v_j\} \in \mathcal{E}$. $\tilde{\mathbf{A}} \in \mathbb{R}^{N \times N}$ denotes the normalized adjacency matrix derived from $\mathbf{A}$. In a GCN framework $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\mathbf{D}^{-\frac{1}{2}}$ where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the diagonal degree matrix with i-th diagonal entry d_i denoting the degree of node v_i. While for GraphSAGE with mean aggregation, $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$. Let $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ denote the symmetric normalized laplacian matrix. In a node classification task, the node to be classified is called the target node, denoted as v_n, with $\mathbf{x}_{v_n}$ and $\mathcal{N}_{v_n}$ representing its feature vector and neighboring index set, respectively. Node representations at the l-th layer are denoted by $\mathbf{H}^l \in \mathbb{R}^{N \times F}$ where the i-th row of $\mathbf{H}^l$ i.e. $\mathbf{h}^l_{v_i} \in \mathbb{R}^{1 \times F}$ denotes the representation for the node v_i. Throughout the paper we use the operator $(\cdot)_i$ to denote the i-th row of any matrix or the i-th element of any vector, for example, $(\mathbf{X})_i$ denotes the i-th row of the node feature matrix $\mathbf{X}$.

2.1 GNN Frameworks

To pave the road for ensuing analysis and algorithm design, we first introduce GraphSAGE and Graph Convolutional Network (GCN) frameworks for both node and graph classification tasks below.

2.1.1 Node Classification:

Consider a GraphSAGE layer with weights $\mathbf{W}_1 \in \mathbb{R}^{F\times D}, \mathbf{W}_2 \in \mathbb{R}^{F\times D}$, and bias $\mathbf{b} \in \mathbb{R}^{1\times F}$ Let $\mathbf{h}^l_{v_n}$ and $\mathbf{h}^l_{\text{agg}}$ denote the hidden and aggregated representation of the node v_n at layer l and σ is a non-linear activation function. Then the GraphSAGE layer can be expressed as

$$\mathbf{h}^l_{v_n} = \sigma(\mathbf{h}^{l-1}_{\text{agg}}\mathbf{W}_1^\top + \mathbf{h}^{l-1}_{v_n}\mathbf{W}_2^\top + \mathbf{b}), \tag{1}$$

with $\mathbf{h}^0_{v_n} = \mathbf{x}_{v_n}$ and $\mathbf{h}^0_{\text{agg}} = \mathbf{x}^{\text{agg}}_{v_n}$ denoting the feature vector and aggregated representation of node v_n respectively. For a GraphSAGE layer with mean aggregation the aggregated representation of a node $\mathbf{h}^{l-1}_{\text{agg}}$ can be obtained as

$$\mathbf{h}^{l-1}_{\text{agg}} = \text{mean}_{j\in\mathcal{N}_{v_n}}(\mathbf{h}^{l-1}_{v_j}). \tag{2}$$

Similarly, for a GCN layer with weights $\mathbf{W} \in \mathbb{R}^{F\times D}$ and bias $\mathbf{b} \in \mathbb{R}^{1\times F}$, the representation of a target node v_n can be obtained via

$$\mathbf{h}^l_{v_n} = \sigma(\mathbf{h}^{l-1}_{\text{agg}}\mathbf{W}^\top + \mathbf{b}). \tag{3}$$

Here $\mathbf{h}^{l-1}_{\text{agg}}$ is obtained by aggregating the neighboring nodal feature vectors as

$$\mathbf{h}^{l-1}_{\text{agg}} = \sum_{j\in\mathcal{N}_{v_n}\cup\{n\}} \frac{\mathbf{h}^{l-1}_{v_j}}{\sqrt{d_n d_j}}. \tag{4}$$

For a node classification task using an L layer GraphSAGE or GCN framework let K denote the number of classes. Also, let $\mathbf{h}^L_{v_n} \in \mathbb{R}^{1\times K}$ denote the output of the final layer and $\mathbf{p} = \mathbf{h}^L_{v_n}$. The final Cross-Entropy loss of a node v_n can be written as

$$\mathcal{L}_{v_n} = -\log \frac{e^{\mathbf{p}_k}}{\sum_{j=1}^K e^{\mathbf{p}_j}}, \tag{5}$$

where k is the corresponding ground-truth label and $\mathbf{p}_j$ denotes the jth element of the vector $\mathbf{p}$.

2.1.2 Graph Classification:

Consider an N-node graph with $\mathbf{X}$ and $\tilde{\mathbf{A}}$ as the node feature matrix and the normalized adjacency matrix respectively, let $\mathbf{H}^l \in \mathbb{R}^{N\times F}$ denote the hidden representations of the nodes at layer l. The output of a GraphSAGE layer can be written as

$$\mathbf{H}^l = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{l-1}\mathbf{W}_1^\top + \mathbf{H}^{l-1}\mathbf{W}_2^\top + \overrightarrow{\mathbf{1}}\mathbf{b}), \tag{6}$$

with $\mathbf{H}^0 = \mathbf{X}$. Here $\overrightarrow{\mathbf{1}}$ denotes a column vector of all ones. Similarly, the output of a GCN layer can be written as

$$\mathbf{H}^l = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{l-1}\mathbf{W}^\top + \overrightarrow{\mathbf{1}}\mathbf{b}). \tag{7}$$

Following the final GNN layer, an MLP layer is applied . For an MLP with weights $\mathbf{W} \in \mathbb{R}^{K\times NF}$, the final readout from the MLP layer can be written as

$$\mathbf{h}_{\text{MLP}} = \mathbf{h}_f\mathbf{W}^\top + \mathbf{b}. \tag{8}$$

where $\mathbf{h}_f \in \mathbb{R}^{1\times NF}$ is the flattened representation of $\mathbf{H}^L \in \mathbb{R}^{N\times F}$ which is the hidden representations obtained at the Lth layer. Let $\mathbf{p} = \mathbf{h}_{\text{MLP}} \in \mathbb{R}^{1\times K}$ be the final readout, the Cross-Entropy Loss $\mathcal{L}$ can be computed using equation 5.

2.2 Federated Graph Learning (FGL)

A typical FL framework consists of a server and C clients. In the node-level FGL, each client samples a mini-batch of B target nodes $\{\mathbf{x}_{v_1}, \cdots, \mathbf{x}_{v_B}\}$ from the graph which is used to compute the gradients of the loss function in equation 5 with respect to the model parameters. In the graph-level FGL, at each step t, each client has a set of multiple graphs. Similar to the node-level FGL the cth client samples a minibatch of B graphs $\{\mathcal{G}_{c_1}, \cdots, \mathcal{G}_{c_B}\}$ which are then used to compute the gradients and train the model. The gradients from N clients are then broadcast to the server for updating the weights as

$$\nabla_{\mathbf{W}^t}\mathcal{L} = \frac{1}{B \cdot C}\sum_{c=1}^{C}\sum_{i=1}^{B}\nabla_{\mathbf{W}^t}\mathcal{L}_{c,i}, \tag{9}$$

$$\mathbf{W}^{t+1} = \mathbf{W}^t - \eta\nabla_{\mathbf{W}^t}\mathcal{L} \tag{10}$$

where $\nabla_{\mathbf{W}^t}\mathcal{L}_{c,i}$ denotes the gradient of the loss at the c-th client for the i-th data sample.

3 Gradient Inversion Attack for GNNs

We consider the scenario where there is a curious but honest server, i.e., the server has access to the model gradient with respect to the client's private graph data. Different attack scenarios are extensively discussed where the server may also have access to different amounts of information.

3.1 Threat Model

The attacker is an *honest-but-curious* server that aims to recover the user's graph data given the gradients of a GNN model. We thoroughly investigate attacks targeting both node classification and graph classification tasks. In addition to possessing knowledge of the gradients, the attacker may also have access to varying amounts of information about the user's private graph data. Hence we categorize the attacks based on the task as follows:
Node Attacker 1: has access to the gradients of a node classification model for the target node and recovers the nodal features of the target node and its neighbors.
Node Attacker 2: has access to the gradients of a node classification model for all the nodes in an egonet or subgraph. Specifically, the gradients are obtained for the loss function $\mathcal{L}_v \in \mathbb{R}^{1\times N}$ with respect to the neural networks weights, where each element $\mathcal{L}_{v_i}$ denotes the loss for node v_i. Unlike traditional classification tasks, the adjacency information is also significant for graph data. In this setting, the graph structure could be unknown. The honest-but-curious server would also try to recover the nodal features and/or the graph structure. Based on the input, we categorize the attacker in more detail as follows

- **Node Attacker 2-a**: has access to the gradients of all the nodes in an egonet or subgraph along with the nodal features and recovers the underlying graph structure.
- **Node Attacker 2-b**: has access to the gradients of all the nodes in an egonet or subgraph along with the graph structure and recovers the nodal features.
- **Node Attacker 2-c**: has access only to the gradients of all nodes in an egonet or subgraph and recovers the graph structure as well as the nodal features.

Graph Attacker: has access to the gradients of the graph for a graph classification model. Similar to Node Attacker 2, in this setting the honest-but-curious server would try to recover the nodal features and/or the graph structure. Hence, we categorize the attack in more detail as follows

- **Graph Attacker-a**: has access to the gradients along with the nodal features and recovers the underlying graph structure.
- **Graph Attacker-b**: has access to the gradients along with the graph structure and recovers the nodal features.

- **Graph Attacker-c**: has access to the gradients and recovers both the nodal features and the graph structure.

3.2 Attack Mechanisms

In this section, GLG is introduced to attack a GNN framework and steal private user data using the leaked gradients in a federated graph learning setting. Specifically, given the gradients of the model from a client for the i-th data sample $\nabla_{\mathbf{W}}\mathcal{L}_{c,i}$, the goal is to recover the input graph data sample that was fed to the model. For simplicity, we denote $\nabla_{\mathbf{W}}\mathcal{L}_{c,i}$ as $\nabla_{\mathbf{W}}\mathcal{L}$ for the graph classification task, and as $\nabla_{\mathbf{W}}\mathcal{L}_v$ for the node classification task. In Zhu et al. (2019), the input data to a model is recovered by matching the dummy gradients with the leaked gradients. The dummy gradients are obtained by feeding dummy input data to the model. Given the gradients at a certain time step as $\nabla_{\mathbf{W}}\mathcal{L}$, the input data is recovered by minimizing the following objective function

$$\mathbb{D} = ||\nabla_{\mathbf{W}}\hat{\mathcal{L}} - \nabla_{\mathbf{W}}\mathcal{L}||^2. \tag{11}$$

Here $\nabla_{\mathbf{W}}\hat{\mathcal{L}}$ is the gradient of the loss obtained from the dummy input data. In Geiping et al. (2020) the cosine loss optimization function is used instead of the ℓ_2 loss in equation 11. Specifically, the objective is to minimize the cosine loss between the actual gradients shared by a client and the gradients obtained from dummy input data

$$\mathbb{D} = 1 - \frac{\nabla_{\mathbf{W}}\hat{\mathcal{L}} \cdot \nabla_{\mathbf{W}}\mathcal{L}}{||\nabla_{\mathbf{W}}\hat{\mathcal{L}}||||\nabla_{\mathbf{W}}\mathcal{L}||}. \tag{12}$$

In this work, it is assumed that the input label is known since in a classification task with cross-entropy loss, the labels can be readily inferred from the gradients Zhao et al. (2020a). For further details regarding this please refer to Appendix A.

Feature Smoothness. As shown in equations 2 and 4 Graph Neural Networks (GNNs) typically employ a feature aggregation mechanism wherein node representations are constructed by integrating features from neighboring nodes prior to weight multiplication at each network layer. However, the objective function proposed in equation 12 may be insufficient for simultaneously recovering both nodal features and graph structure. To address this limitation, we leverage intrinsic properties characteristic of real-world graphs that can enhance graph data reconstruction. Many real-world graph structures, such as social networks, exhibit a fundamental property of feature smoothness, wherein connected nodes tend to possess similar characteristics. To formally enforce and quantify this feature smoothness in the reconstructed graph data, we propose the following loss function, which is adapted from the approach introduced by Zhang et al. (2022):

$$\begin{aligned}\mathcal{L}s &= tr(\mathbf{XLX}^\top) \\ &= \sum i, j \in \mathcal{E}(\frac{\mathbf{x}_{v_i}}{\sqrt{d_i}} - \frac{\mathbf{x}_{v_j}}{\sqrt{d_j}})\end{aligned} \tag{13}$$

The loss function captures the similarity between adjacent nodes by minimizing the normalized feature differences across connected nodes.

In addition, the Frobenius norm regularizer is also introduced such that the norm of $\mathbf{A}$ is bounded. Overall, the final objective can be written as

$$\hat{\mathbb{D}} = \mathbb{D} + \alpha\mathcal{L}_s + \beta||\mathbf{A}||_F^2 \tag{14}$$

where $\mathbb{D}$ is defined in equation 12, α and β are hyperparameters. The iterative algorithm for recovering the private data from the gradients in a node classification task and a graph classification task are summarized in algorithms 1, 2 and 3, respectively. Specifically, Node Attacker 1 utilizes algorithm 1 for recovering the nodal features of the target node and its neighbors. Node Attackers 2 (a, b and c) utilizes algorithm 2 for recovering the nodal features and graph structure given the gradients of the loss function for each node in the

egonet/subgraph. By default, algorithm 2 assumes that both nodal features and graph structure are unknown and tries to recover both. However, only the unknown parameter will be optimized when either of the two is known. For instance, Node Attacker 2-a only optimizes for $\mathbf{A}$ in Line 9 of algorithm 2 while skipping Line 8 since the nodal features $\mathbf{X}$ is known. The same holds for Node Attacker 2-b. However, unlike the other two Node Attacker 2-c optimizes for both. Similar logic holds for Graph Attacker (a, b and c) which utilizes algorithm 3. Note that algorithms 2 and 3 might seem similar since they are recovering the same variables. However, a key difference between the two is that algorithm 2 is attacking a node classification model while algorithm 3 attacks a graph classification model. In algorithm 2, the gradients of the loss function are computed individually for each node. This is because each node within the subgraph has an associated label. In contrast, algorithm 3 calculates gradients for the entire graph, as the graph has a single label.

Recovering the Adjacency Matrix: In all the scenarios where the attacker is trying to recover the adjacency matrix, projected gradient descent is applied. Specifically, the gradient descent step in Line 9 of algorithms 2 and 3 entails a projection step, where each entry (i, j) of the adjacency matrix is updated through the entry-wise projection operator defined as

$$\hat{\mathbf{A}}_{ij} = \text{proj}_{[0,1]}(\tilde{\mathbf{A}}_{ij}) = \begin{cases} 1, & \tilde{\mathbf{A}}_{ij} \geq 1 \\ 0, & \tilde{\mathbf{A}}_{ij} \leq 0 \\ \tilde{\mathbf{A}}_{ij}, & \text{otherwise} \end{cases} \tag{15}$$

Finally, to recover the binary adjacency matrix, we consider each entry $\hat{\mathbf{A}}_{ij}$ as the probability of any edge between nodes v_i and v_j and the recovered binary adjacency matrix is obtained by sampling from a Bernoulli distribution with the corresponding probability at the last iteration, see also Line 11 of algorithm 2 and algorithm 3.

Since the recovered adjacency matrix $\hat{\mathbf{A}}$ is obtained via sampling through Bernoulli distribution, the Frobenius regularizer has the effect of reducing the magnitude of entries of $\hat{\mathbf{A}}$. Consequently, this regularizer also contributes to the promotion of sparsity in the ultimately recovered adjacency matrix. This sparsity property is particularly relevant since real-world social networks are sparse.

4 Theoretical Justification

While prior work Geiping et al. (2020) has shown that the input to a fully-connected layer can be recovered from the layer's gradients, the analysis of GNNs is largely under-explored. Below we will extensively study whether similar conclusions apply to GraphSAGE and GCN and what parts of the graph input can be recovered from the gradients.

4.1 Node Attacker

Consider a node classification setting, where the inputs to a GNN layer are the target nodal features and the neighboring nodal features denoted as $\mathbf{x}_v$ and $\{\mathbf{x}_{v_j}\}_{j \in \mathcal{N}_v}$. Through theoretical analysis we will show that parts of the input to a GCN or GraphSAGE layer can be recovered from the gradients analytically without solving an iterative optimization problem.

Proposition 1. *For a GraphSAGE defined in equation 1 or a GCN layer defined in equation 3, Node Attacker 1 can recover the aggregated representations of a target node denoted as $\mathbf{x}_v^{agg}$ (see equation 4 with $\mathbf{h}_{agg}^0 = \mathbf{x}_v^{agg}$) given the gradients of the first layer as $\mathbf{x}_v^{agg} = \nabla_{(\mathbf{W})_i}\mathcal{L}_v / \nabla_{(\mathbf{b})_i}\mathcal{L}_v$ for GCN and as $\mathbf{x}_v^{agg} = \nabla_{(\mathbf{W}_1)_i}\mathcal{L}_v / \nabla_{(\mathbf{b})_i}\mathcal{L}_v$ for GraphSAGE provided that $\nabla_{(\mathbf{b})_i}\mathcal{L}_v \neq 0$.*

Proof. We show the proof for a GCN layer, the proof for GraphSAGE is similar and is deferred to the appendix. For a GCN layer, the aggregated input at the target node can be recovered from the gradients

Algorithm 1 GLG (Node Attacker).

Input: 2-layer GNN model $F(\mathbf{x}_v, \{\mathbf{x}_{v_j}\}_{j\in\mathcal{N}_v}, \mathbf{W})$
$\nabla_{\mathbf{W}}\mathcal{L}_v$: gradients calculated by training data, $\mathbf{y}_v$: True label
Output: private data $\mathbf{x}_v$
//Initialize dummy target and neighboring node features by creating a two level tree with degree as d_{tree} and the label:
Initialize $\hat{\mathbf{x}}_v^1$ and $\{\hat{\mathbf{x}}_{v_j}\}_{j\in\mathcal{N}_v}^1$
for $p \leftarrow 1$ to P **do**
  //Compute dummy gradients:
  $\nabla_{\mathbf{W}}\hat{\mathcal{L}}_v = \partial\mathcal{L}(F(\hat{\mathbf{x}}_v^p, \{\hat{\mathbf{x}}_{v_j}^p\}_{j\in\mathcal{N}_v}, \mathbf{W}), \mathbf{y}_v)\partial\mathbf{W}$
  Compute $\mathbb{D}^p$ as in equation 12
  $\hat{\mathbf{x}}_v^{p+1} = \hat{\mathbf{x}}_v^p - \eta\nabla_{\hat{\mathbf{x}}_v^p}\mathbb{D}^p$,
  for $j \in \mathcal{N}_v$ **do**
    $\hat{\mathbf{x}}_{v_j}^{p+1} = \hat{\mathbf{x}}_{v_j}^p - \eta\nabla_{\hat{\mathbf{x}}_{v_j}^p}\mathbb{D}^p$,
  end for
end for
return $\hat{\mathbf{x}} = \hat{\mathbf{x}}_v^{P+1}$

with respect to the weights of the first layer by writing the following equations

$$\mathbf{h}_v = \sigma(\tilde{\mathbf{h}}_v), \tag{16}$$

$$\tilde{\mathbf{h}}_v = \mathbf{x}_v^{\text{agg}}\mathbf{W}^\top + \mathbf{b}, \tag{17}$$

$$\nabla_{(\mathbf{W})_i}\mathcal{L}_v = \nabla_{(\tilde{\mathbf{h}}_v)_i}\mathcal{L}_v \cdot \nabla_{(\mathbf{W})_i}(\tilde{\mathbf{h}}_v)_i, \tag{18}$$

$$\nabla_{(\mathbf{W})_i}\mathcal{L}_v = \nabla_{(\mathbf{b})_i}\mathcal{L}_v \cdot \mathbf{x}_v^{\text{agg}} \tag{19}$$

where $(\mathbf{W})_i$ denotes the ith row of $\mathbf{W}$, $(\mathbf{b})_i$ and $(\tilde{\mathbf{h}}_v)_i$ denotes the ith element of vectors $\mathbf{b}$ and $\tilde{\mathbf{h}}_v$ respectively. It can be concluded from equation 30 that $\mathbf{x}_v^{\text{agg}}$ can be recovered as $\nabla_{(\mathbf{W})_i}\mathcal{L}_v/\nabla_{(\mathbf{b})_i}\mathcal{L}_v$ as long as $\nabla_{(\mathbf{b})_i}\mathcal{L}_v \neq 0$. □

It can be observed from the above proof that the analytic reconstruction is independent of the loss function or the non-linearity used after the layer and only depends on the gradients of the loss function with respect to the weights of the model.

Proposition 2. *For a GraphSAGE layer defined in equation 1, Node Attacker 1 can recover the target node features given the gradients of the first layer as* $\mathbf{x}_v = \nabla_{(\mathbf{W}_2)_i}\mathcal{L}_v/\nabla_{(\mathbf{b})_i}\mathcal{L}_v$ *as long as* $\nabla_{(\mathbf{b})_i}\mathcal{L}_v \neq 0$.

Proof. See Appendix C. □

In the following, we focus on theoretically analyzing the attack setting of Node Attacker 2, specifically targeting the recovery of nodal features and/or the graph structure. To reiterate, this setting involves leveraging the gradients of the loss function of GNNs associated with each node in the egonet or subgraph.

Proposition 3. *For a GCN layer defined in equations equation 3 and equation 4, Node Attacker 2-a can recover* $\tilde{\mathbf{A}}$ *given the gradients of the first layer as long as* $\mathbf{X}$ *is full row-rank.*

Proof. From Proposition 1 we know that given the gradients for a node we can recover the aggregated representations of that node. Since we know the gradients for each node we can recover the aggregated representation for each node. Let $\mathbf{X}^{\text{agg}} \in \mathbb{R}^{N\times D}$ denote the aggregated node representation matrix where each row i denotes the aggregated node representation of node v_i, equation 4 in the matrix form can be written as the following for $l - 1 = 0$.

$$\mathbf{X}^{\text{agg}} = \tilde{\mathbf{A}}\mathbf{X}. \tag{20}$$

Since $\mathbf{X}^{\text{agg}}$ and $\mathbf{X}$ are known, the adjacency matrix can be then recovered as $\tilde{\mathbf{A}} = \mathbf{X}^{\text{agg}}\mathbf{X}^{\dagger}$. □

Proposition 4. *For a GCN layer defined in equations equation 4 and equation 3, Node Attacker 2-b can recover* $\mathbf{X}$ *given the gradients of the first layer if* $\tilde{\mathbf{A}}$ *is full column-rank.*

Proof. Similar to Proposition 3 once we have $\mathbf{X}^{\text{agg}}$, $\mathbf{X}$ can be obtained as $\mathbf{X} = \mathbf{X}^{\text{agg}}\tilde{\mathbf{A}}^{\dagger}$ □

Algorithm 2 GLG (Node Attacker 2)

Input: GNN model $F(\mathbf{X}, \mathbf{A}, \mathbf{W})$, $\mathbf{W}$: parameter weights; $\nabla_{\mathbf{W}}\mathcal{L}$: gradients calculated by private training data for each node in the graph, $\mathbf{y}$: True label
Output: private training data $\mathbf{X}, \mathbf{A}$
Initialize dummy features and labels $\hat{\mathbf{X}}^1$, $\hat{\mathbf{A}}^1$
for $p \leftarrow 1$ to P **do**
  //Compute dummy gradients:
  $\nabla_{\mathbf{W}}\hat{\mathcal{L}} = \partial\mathcal{L}(F(\hat{\mathbf{X}}^p, \hat{\mathbf{A}}^p, \mathbf{W}), \mathbf{y})/\partial\mathbf{W}$
  Compute $\hat{\mathbb{D}}^p$ as given in equation 14
  $\hat{\mathbf{X}}^{p+1} = \hat{\mathbf{X}}^p - \eta\nabla_{\hat{\mathbf{X}}^p}\hat{\mathbb{D}}^p$,
  $\hat{\mathbf{A}}^{p+1} = \text{proj}_{[0,1]}(\hat{\mathbf{A}}^p - \eta\nabla_{\hat{\mathbf{A}}^p}\hat{\mathbb{D}}^p)$,
end for
return $\hat{\mathbf{X}} = \hat{\mathbf{X}}^{P+1}, \hat{\mathbf{A}} \sim Ber(\mathbf{A}^{P+1})$

Proposition 5. *For a GraphSAGE layer defined in equation 1, Node Attacker 2-c can recover both* $\mathbf{X}$ *and* $\tilde{\mathbf{A}}$ *given only the gradients of the first layer for each node.*

Proof. Proposition 2 states that for a GraphSAGE layer, the target node features can be recovered given the gradients of the first layer for that node. Similarly, if we have access to the gradients for each node in the graph we can recover all the nodal features. Let $\mathbf{X}$ denote this matrix. From Propositions 1 it is known that for a GraphSAGE layer, the aggregated node representations $\mathbf{X}^{\text{agg}}$ can be recovered. With both $\mathbf{X}$ and $\mathbf{X}^{\text{agg}}$ in hand, the adjacency matrix can be recovered as $\tilde{\mathbf{A}} = \mathbf{X}^{\text{agg}}\mathbf{X}^{\dagger}$ similar to Proposition 3. □

The aforementioned proposition reveals big privacy concerns. Even without any prior knowledge of the graph data, the attacker can successfully reconstruct both the nodal features $\mathbf{X}$ and the underlying graph structure $\mathbf{A}$. This result is particularly important as one would anticipate that the intertwined nature of graph data makes it impossible to independently recover $\mathbf{X}$ and $\mathbf{A}$, which is however shown not to be true.

Algorithm 3 Graph Attacker

Input: GNN model $F(\mathbf{X}, \mathbf{A}, \mathbf{W})$, $\mathbf{W}$: parameter weights; $\nabla_{\mathbf{W}}\mathcal{L}$: gradients calculated by private training data, $\mathbf{y}$: True label
Output: private training data $\mathbf{X}, \mathbf{A}$
Initialize dummy features and labels $\hat{\mathbf{X}}^1$, $\hat{\mathbf{A}}^1$
for $p \leftarrow 1$ to P **do**
  //Compute dummy gradients:
  $\nabla_{\mathbf{W}}\hat{\mathcal{L}} = \partial\mathcal{L}(F(\hat{\mathbf{X}}^p, \hat{\mathbf{A}}^p, \mathbf{W}), \mathbf{y})/\partial\mathbf{W}$
  Compute D^p as given in equation 12
  $\hat{\mathbf{X}}^{p+1} = \hat{\mathbf{X}}^p - \eta\nabla_{\hat{\mathbf{X}}^p}\mathbb{D}^p$,
  $\hat{\mathbf{A}}^{p+1} = \text{proj}_{[0,1]}(\hat{\mathbf{A}}^p - \eta\nabla_{\hat{\mathbf{A}}^p}\mathbb{D}^p)$,
end for
return $\hat{\mathbf{X}} = \hat{\mathbf{X}}^{P+1}, \hat{\mathbf{A}} \sim Ber(\mathbf{A}^{P+1})$

Table 1: Target node feature recovery $\mathbf{x}_v$ (RNMSE) for Node Attacker 1.

		Synthetic	FB	GitHub
GraphSAGE ($\times 10^{-3}$)	Mean	2.8±2.5	2.6±1.3	2.8±1.4
	Min	0.75	0.6	0.9
GCN	Mean	0.54±0.35	0.81±0.61	1.27±0.79
	Min	0.23	0.22	0.18

4.2 Graph Attacker

The present subsection provides theoretical analysis for a Graph Attacker-a in a scenario where prior information may be available for the graph data to be attacked. It can be shown that for GraphSAGE, prior knowledge of the nodal feature matrix $\mathbf{X}$ can help recover the graph structure $\mathbf{A}$, making the framework more vulnerable to the gradient-based attacks.

Proposition 6. *For a GraphSAGE framework defined in equation 6 with $\mathbf{H}^0 = \mathbf{X}$, Graph Attacker-a can recover $\tilde{\mathbf{A}}$ from the gradients of the first layer as $\tilde{\mathbf{A}} = \mathbf{X}(\nabla_{\mathbf{W}_2}\mathcal{L})^{\dagger}(\nabla_{\mathbf{W}_1}\mathcal{L})\mathbf{X}^{\dagger}$ if the nodal feature matrix $\mathbf{X}$ is full-row rank and known.*

Proof. See Appendix D. □

Therefore if the attacker knows the nodal feature matrix $\mathbf{X}$ along with the gradients, it can recover the normalized adjacency matrix $\tilde{\mathbf{A}}$ analytically.
Even though it is not possible to analytically demonstrate the recovery for Graph Attacker-b and Graph Attacker-c, we do conduct experiments for all the threat models for both GraphSAGE and GCN.

5 Data & Experiments

The performance of the proposed algorithms is tested over social network datasets and molecular datasets. In this section, the performance of the algorithms is tested over real-world social network datasets as well as synthetic datasets to validate the efficacy of the proposed framework. The results on molecular datasets will be provided in Appendix E.

Evaluation Metrics. Error metrics for recovering the nodal features and the graph structure are evaluated, along with their standard deviation.

- **Nodal Features**: To evaluate the performance of the recovered node features, the Root Normalised Mean Squared Error(RNMSE) is used

$$\text{RNMSE}(\mathbf{x}_v, \hat{\mathbf{x}}_v) = \frac{||\mathbf{x}_v - \hat{\mathbf{x}}_v||}{||\mathbf{x}_v||}. \tag{21}$$

  In order to evaluate the performance of recovering $\mathbf{X}$ in a graph classification setting, we report the mean RNMSE over all the nodal features in $\mathbf{X}$.

- **Graph Structure**: To evaluate the recovery performance of the binary adjacency matrix, we use the following metrics
  - **Accuracy**: defined as $\frac{\sum_{i,j}(\hat{\mathbf{A}}_{ij} - \mathbf{A}_{ij})}{N*N}$.
  - **AUC** (Area Under Curve): The area under the Reciever Operator Characteristic Curve.
  - **AP** (Average Precision): is defined as $\frac{TP}{TP+FP}$ where TP stands for number of True Positives and FP stands for False Positives. True Positives are the correctly identified edges in the actual graph by the attacker and False Positives are the non-edges incorrectly classified as edges by the attacker.

Experimental Settings. Algorithms 1, 2 and 3 solve an optimization problem using gradient descent. The gradient descent step can be replaced with off-the-shelf optimizers such as Adam or L-BFGS. In all the experiments, Adam is used as the optimizer. In an attack setting, allowing multiple restarts to the optimizer (especially L-BFGS) from different starting points can significantly increase the quality of recovery. For a fair comparison, multiple restarts are not used, and all results presented are obtained with a single run of the optimizer. However, in an actual attack setting, an attacker may be able to *greatly improve* the recovery quality by allowing multiple restarts.

In the experiments, a 2-layer GNN model with hidden dimension 100 and a sigmoid activation function is employed. The weights of the model are randomly initialized. For all the experiments, the dummy nodal features are initialized randomly, with each entry sampled from the standard normal distribution i.e., $\mathcal{N}(0,1)$. The dummy adjacency matrix is initialized by randomly setting its entries to 0 or 1. The hyperparameters values for the feature smoothness and the sparsity regularizers are set to $\alpha = 10^{-9}$ and $\beta = 10^{-7}$. These are the values selected by grid search for the choice that leads to the best performance.

- **Node Attacker 1** uses the cosine loss objective given in equation 12 without regularization.
- **Node Attacker 2** uses the objective function is defined in equation 14.
- **Graph Attacker** uses the objective function as defined in equation 14.

All results reported are averaged over 20 runs of the attacks.

Datasets. We consider the following three datasets:

- **GitHub** Rozemberczki et al. (2021): Nodes represent developers on GitHub and edges are mutual follower relationships. The nodal features are extracted based on the location, repositories starred, employer and e-mail address. Binary labels indicate the user's job title, either web developer or machine learning developer. The details regarding the number of nodes and edges in this dataset are presented in Table 2.
- **FacebookPagePage (FB)** Rozemberczki et al. (2019): A page-page graph of verified Facebook sites. Nodes correspond to official Facebook pages, links represent mutual likes between sites. Node features are extracted from the site descriptions. The labels denote the categories of the sites. The details regarding the number of nodes and edges in this dataset are presented in Table 2.
- **Synthetic**: A synthetic dataset containing undirected random graphs with an average degree of 4 and average number of nodes set to 50. The node features are generated by sampling from the standard normal distribution i.e. $\mathcal{N}(0,1)$. Node labels are generated by uniformly sampling an integer between zero and the number of classes specified. Edges are generated by uniformly sampling its two endpoints from the node set. The total number of edges are set to $\frac{d*n}{2}$.

Table 2: Dataset statistics for Node Classification task.

Dataset	**#Nodes**	**#Edges**	**#Labels**
GitHub	37,300	578,006	2
Facebook	22,470	342,004	4

Note that since Node Attacker 2 and Graph Attacker aim at attacking the subgraphs. We randomly sample a node and its 3-hop neighborhood as the subgraph to be attacked. This setting is also consistent with the Federated Graph Learning setting where each client tends to store a subgraph instead of the whole graph.

5.1 Node Attacker 1

In this section, we evaluated the performance of feature recovery of Node Attacker 1 for the node classification task.

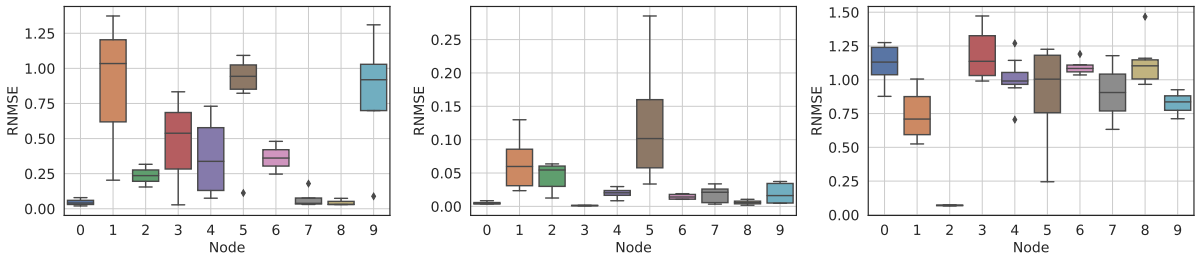
Table 3: Recovery of **A** with Node Attacker 2-a.

Framework	Facebook			GitHub		
	ACC.	AUC	AP	ACC.	AUC	AP
GraphSAGE	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0
GCN	0.97±0.008	0.97±0.02	0.85±0.04	0.97±0.02	0.98±0.001	0.87±0.02

Table 4: Recovery of **X** with Node Attacker 2-b.

	Facebook	GitHub
GraphSAGE ($\times 10^{-4}$)	0.7±0.6	0.9±0.9
GCN	0.07±0.01	0.09±0.02

Target node feature recovery. In this experiment, since no knowledge of the graph data set is available on the attacker side, an initial dummy graph is generated as a 2-layer tree with the target node as the root node and the degree of each node $d_{tree} = 10$. Table 1 lists the RNMSE of recovering the target node features. The results are obtained by running the attack over 20 randomly selected nodes from each data set. It can be observed that for GraphSAGE, the attacker can recover the nodal features with high accuracy. For GCN, the attacker fails to recover the target nodal features. Such results are consistent with the theoretical analysis in Propositions 1 and 2. To further evaluate the performance, we also evaluated the minimum RNMSE obtained in Table 1, which indicates the best recovery performance among the 20 nodes. It can be observed that while the mean RNMSE of the target node features for GCN is large, the minimum RNMSE can be small. This implies that data might also be leaked when GCN is used in pracitice.



(a) Facebook (b) Synthetic (c) Github

Figure 2: One-hop neighbor recovery (RNMSE) for Node Attacker-1 with GraphSAGE.

One-hop Neighborhood Nodal Feature Recovery. To evaluate the performance of the node attacker in recovering the nodal features of one-hop neighbors, a target node is selected randomly from the graph. A dummy graph is initialized as a depth-2 tree with d_{tree} set to the actual number of one-hop neighbors of the target node. Since there's permutation ambiguity in the recovered nodal features, the Hungarian Algorithm is used to find the best match with the original neighboring node features Kuhn (2010). Figure 2 shows the one-hop neighbor recovery error (RNMSE) using GraphSAGE. The results are averaged over 10 randomly selected nodes. For some nodes, the nodal features of neighbors were recovered accurately. For example, nodes 0 and 8 in Figure 2a for Facebook. Further, although the mean RNMSE of the recovered neighbors is high, the minimum RNMSE of the best-recovered neighbors can be low (node 9 in Figure 2a), indicating private data leakage.

5.2 Node Attacker 2

In this section, we evaluate the performance of Node Attacker 2 which has access to the gradient of a node classification model for all nodes in a subgraph.

Table 5: Recovery of **X** (RNMSE) and **A** (AUC and AP) with Node Attacker 2-c.

Framework	Facebook			GitHub		
	X ↓ (RNMSE)	AUC ↑	AP ↑	**X** ↓ (RNMSE)	AUC ↑	AP ↑
GraphSAGE	0.002±0.003	0.99±0.01	0.99±0.0004	0.001±0.002	0.99±0.004	0.99±0.009
GCN	1.39±0.49	0.97±0.03	0.75±0.05	0.58±0.10	0.96±0.01	0.83±0.04
GCN (w/o reg.)	1.19±0.08	0.95±0.001	0.65±0.2	0.81±0.41	0.94±0.02	0.72±0.09

Table 6: Recovery of **A** with Graph Attacker-a.

Framework	Facebook			GitHub		
	ACC.	AUC	AP	ACC.	AUC	AP
GraphSAGE	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0
GCN	0.97±0.03	0.93±0.05	0.97±0.02	0.97±0.02	0.95±0.04	0.97±0.02

Node Attacker 2-a. The recovery performance of the adjacency matrix for this setting is listed in Table 3. Note that the attacker **perfectly** recovers the graph structure for the GraphSAGE model. For GCN, even though Proposition 3 states that we can exactly recover the adjacency matrix, we do observe some error in the recovery. This can be potentially attributed to either the limited number of iterations employed in the attack, hindering its convergence, or the optimization procedure encountering a local minimum. Nonetheless, even though not a perfect recovery, the attacker can recover the graph structure with high accuracy and precision for GCN.

Node Attacker 2-b. Table 4 shows the results for Node Attacker 2-b. The attacker is capable of obtaining almost **perfect** recovery of nodal features for the GraphSAGE model. Similarly, for the GCN model, the RNMSE values, although slightly higher, remain sufficiently low, which indicates data leakage. Note that the small error can again be attributed to the potential algorithmic issues such as local optimal as discussed before.

Node Attacker 2-c. In this case, the attacker doesn't have any prior information about the graph data and tries to recover both the nodal features and the graph structure. The results are shown in Table 5. For the GraphSAGE model, it can be observed that the attacker can recover both the features and the adjacency matrix with high accuracy, in both Facebook and GitHub datasets. Again, such results are consistent with our theoretical results in Proposition 5. Surprisingly, for the GCN framework, the attacker is still able to recover **A** with high AUC and AP scores. Note that, no theoretical guarantee was provided in Section 4 for node attacker 2-c for GCN framework. Such empirical results may be attributed to the introduced feature smoothness and sparsity regularizers in equation 14. To further investigate the effect of regularizers, the performance of the attack on the model was evaluated without the regularizers. These results are also presented in Table 5. It can be observed that there is a clear decline in the performance of recovering the adjacency matrix if the regularizers are not introduced, especially in Average Precision (AP). It suggests that the introduced regularizers can effectively improve the performance of the attack.

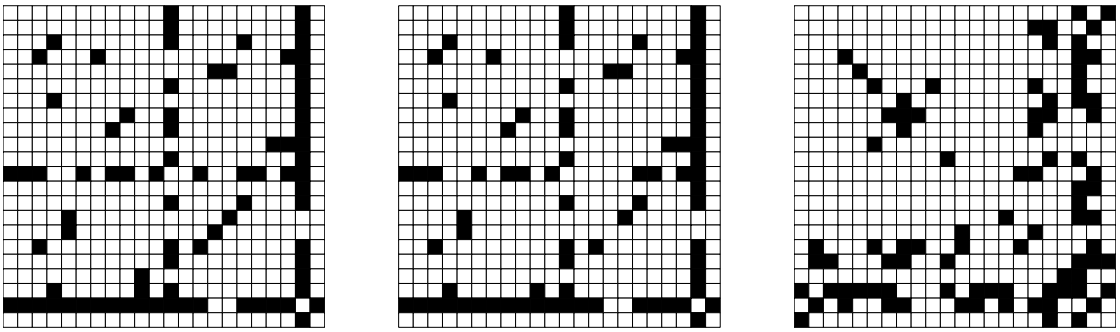
Table 7: Recovery of **X** (RNMSE) Graph Attacker b.

	Facebook	GitHub
GraphSAGE	0.18±0.08	0.15±0.08
GCN	0.61±0.08	0.59±0.08

5.3 Graph Attacker

In this subsection, the performance of different graph attackers are evaluated for attacking nodal features as well as the adjacency matrix of the private subgraph.

Graph Attacker-a. Table 6 lists the recovery performance of the graph structure. It can be observed that the attacker can **perfectly** recover the adjacency matrix from the gradients of a GraphSAGE model in both datasets. Such results are also consistent with the results of Proposition 6. On the other hand, in the case of GCN, despite the absence of theoretical guarantees, the attacker demonstrates high accuracy, precision, and AUC values in identifying the edges. Such results can again be attributed to the feature smoothness and sparsity regularize which help leverage usual properties for social networks. To assess the impact of the regularizer, we again conduct the attack with and without the regularizer as shown in equations equation 12 and equation 14. The resulting recovered adjacency matrices are visualized through heatmaps as depicted in Figures 3b and 3c. Figure 3a illustrates the true graph structure of the private data under attack for comparison. The difference in recovery quality between the two scenarios is evident. When the regularizer is not used, the attack accurately identifies a substantial number of edges. However, this also leads to the erroneous identification of many non-edges as edges, resulting in a lower AP score as also shown in the Figure 3.



(a) (b) (c)

Figure 3: The heatmaps of the recovered adjacency matrix with Graph Attacker-a from the gradient of GCN models. (a) actual graph. (b) recovered graph structure with regularizers (AUC=0.981, AP=1.0). (c) recovered graph structure without regularizers (AUC=0.977, AP=0.86).

Graph Attacker-b. In this setting, the attacker tries to recover the nodal feature matrix given the gradients and the underlying graph structure. The results are listed in Table 7. It can be observed that while the attacker is unable to recover nodal features from a GCN model, it can recover nodel features from a GraphSAGE model with small errors. The results also corroborate that this attacker is different from Node Attacker 2-b where the gradients of the loss for each node are leaked to the attacker. Unlike this scenario, Node Attacker 2-b can successfully recover the nodal features.

Graph Attacker-c. The results for Graph Attacker-c are listed in Table 8. Similar to the results of Node Attacker 2-c, the adjacency matrix can be accurately recovered in this setting for GraphSAGE. Interestingly, in contrast to the previous results, the attack performs better on the GCN than the GraphSAGE framework. Also, in this setting, the attack fails to reconstruct the nodal features for both GraphSAGE and GCN frameworks.

5.4 Attacking batched data

We also extended the experiments for node and graph attackers to the minibatch setting. Specifically, we only evaluate GraphSAGE for both tasks since it provides the best recovery in the stochastic (single data sample) setting as shown in the previous experiments. Although effective attack techniques Wen et al. (2022) have been proposed to recover image data from large batches, here we tested the performance of the attackers by simply optimizing with respect to the dummy input batch. Hence, the performance may improve if attack

Table 8: Recovery of $\mathbf{X}$ (RNMSE) and $\mathbf{A}$ (AUC and AP) with Graph Attacker-c.

Framework	Facebook			GitHub		
	$\mathbf{X}\downarrow$	AUC $\uparrow$	AP $\uparrow$	$\mathbf{X}\downarrow$	AUC $\uparrow$	AP $\uparrow$
GraphSAGE	0.58±0.10	0.89±0.06	0.98±0.01	0.62±0.09	0.92±0.05	0.98±0.02
GCN	0.60±0.09	0.92±0.03	0.97±0.01	0.61±0.03	0.93±0.01	0.98±0.009

Table 9: Target node feature recovery $\mathbf{x}_v$ (RNMSE $\times 10^{-2}$) for Node Attacker 1 in a batched setting.

Dataset		B=5	B=20	B=50
Synth.	Mean	0.07±0.02	0.8±0.3	3.9±1.4
	Min	0.0	0.13	0.4
FB	Mean	0.9±0.4	12±64	17±95
	Min	0.8	28	65
Github	Mean	28±14	126±78	132±83
	Min	0.3	111	107

techniques for batch data are also incorporated. Due to possible permutation ambiguity in a batch setting, the Hungarian algorithm is used to find the best match with the actual input data. The evaluation metric for the algorithm is the RNMSE between all pairs of the dummy batch and the actual batch. These results are averaged over 10 independent runs for each batch size. Experiments are conducted for Node Attacker 1 and Graph Attacker-b.

Node Attacker 1. Table 9 shows the performance of recovering the target node features for batches of size $B = 5, 20$, and 50. It can be observed that the best results are obtained for the Synthetic Dataset with accurate recovery across all batch sizes. When the batch size is $B = 5$, the attack provides accurate recovery across all the 3 datasets.

Graph Attacker-b. In this scenario, the attack attempts to recover $\mathbf{X}$ for the whole batch with $\mathbf{A}$ being known. To simplify the matching after running the attack we only consider graphs with the same number of nodes. This is done by generating Erdos-Renyi (ER) graphs with the number of nodes $n = 50$. For a ER graph with n nodes and edge probability p, the average degree of a node is $d = (n-1)p$. We conduct experiments by varying d with different batch sizes. The nodal features are generated by sampling from the standard normal distribution i.e. $\mathcal{N}(0, 1)$. Table 10 lists the performance of recovering $\mathbf{X}$ with varying batch sizes and degrees. As expected, the quality of recovery declines with increasing batch size, with the best results being achieved at $B = 5$. Also, the performance remains roughly the same across all degrees.

5.5 Sensitivity Analysis for Hyperparameters

In our attack, numerous hyperparameters come into play, including but not limited to the regularization parameters α and β, the graph size, and the network width. In our analysis, we examined a subset of these hyperparameters in different settings and subsequently present the corresponding outcomes below.

Table 10: Recovery performance of $\mathbf{X}$ (RNMSE $\times 10^{-2}$) for Graph Attacker-b for batched graph dataset.

d		B=5	B=20	B=50
4	Mean	23 ± 31	82 ± 27	119±1
	Min	0.04	9	86
8	Mean	25±33	84±34	121±9
	Min	0.04	9	87
12	Mean	22±30	89±36	121±9
	Min	0.04	9	87

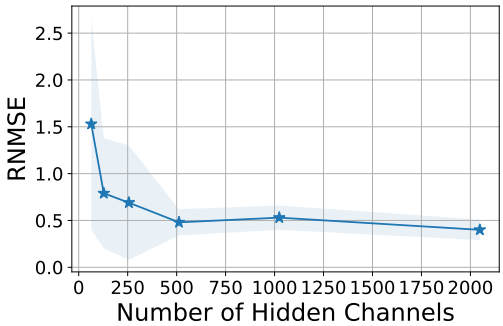
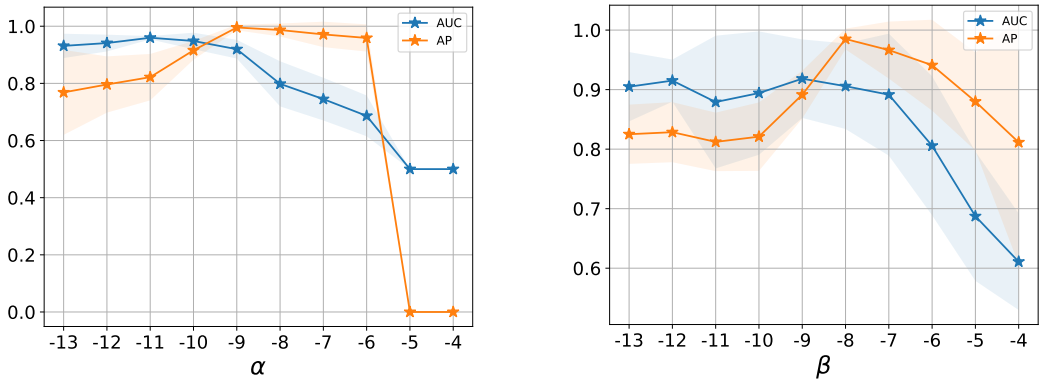


Figure 4: Variation in RNMSE of the recovered target features vs. the number of hidden channels for a GCN framework on the Github dataset.

Network Width. All previous experiments were conducted with the number of hidden channels set to 100. In the setting of Node Attacker 1, the attack could *successfully* recover the target nodal features for a GraphSAGE framework with high accuracy. However, for a GCN framework, the attack only managed to recover the aggregated nodal features. Consequently, in this section, we explore the impact of varying the number of hidden channels in the 2-layer GCN framework on the quality of recovery. Figure 4 illustrates the results of target node feature recovery for the Github Dataset. Notably, as the number of hidden channels increases, there is a clear decrease in the RNMSE values, suggesting that wider networks are more susceptible to the attack.



(a) (b)

Figure 5: Variation of AUC and AP vs. α and β for Graph Attacker-a.

Regularization Hyperparameters. To investigate the influence of α and β in equation 14, we examine the effects on the recovery performance of Graph Attacker-a for the GCN framework. More specifically, we maintain one hyperparameter at a fixed value while altering the other, and depict the AUC and AP values. Figure 5a illustrates the AUC and AP values for different α values in a log scale with base 10. The values on the x-axis are also in the log scale with base 10. In Figure 5a, α is varied while keeping $\beta = 10^{-7}$. Similarly, in Figure 5b, where β is varied, α is fixed at 10^{-9}. It is evident from the figures that higher values of hyperparameters severely degrade the performance of the attack whereas extremely low values give suboptimal results. This clearly demonstrates the effectiveness of the feature smoothness and sparsity regularizers.

6 Conclusion & Future Work

In this work, we presented the **first** attempt to study the gradient inversion attack for Graph Neural Networks (GNNs) in a federated graph learning setting and explored what information can be leaked about the graph

data from the gradient. We studied both the node classification and the graph classification tasks, along with two widely used GNN structures, GraphSAGE and GCN. Through both theoretical and empirical results, we highlighted and analyzed the vulnerabilities of different GNN frameworks across a wide variety of gradient inversion attack settings.

Future work. In the future, it would be interesting to study the extension of the attack mechanisms to other GNN frameworks, such as GIN (Graph Isomorphism Networks) Xu et al. (2019) and GAT (Graph Attention Networks) Veličković et al. (2018). Experiments and theoretical analysis regarding the vulnerability of other GNN frameworks hence is a promising direction. For example, are certain GNN frameworks more robust to gradient inversion attacks than others? Also, are there any other special properties that can be utilized for graph datasets other than social networks? For instance, it is worth exploiting the fact that the degrees of nodes in molecular datasets exhibit an upper limit. Finally, recent studies such as Wang et al. (2023) have provided theoretical bounds on the recovery quality of minibatch data. Hence, an intriguing question would be whether similar guarantees can be established for graph data.

Furthermore, it is crucial to have appropriate defense mechanisms in place to prevent data leakage. It is commonly believed that employing larger batch sizes could be an effective defense against the gradient inversion attack, but our experiments show that even in batches of size 50, the private data can be recovered with high accuracy. Therefore, solely relying on larger batch sizes is insufficient as a defense strategy against gradient inversion attacks. Recent studies such as Wen et al. (2022); Wang et al. (2023) have provided provable guarantees regarding the quality of recovered private data from training batches. Based on the various defense strategies introduced in Gupta et al. (2022), some potential defense mechanisms can remain effective in graph setting: e.g., noisy gradients Abadi et al. (2016), gradient pruning Zhu et al. (2019), encoding inputs Huang et al. (2020). However, more specific defensive mechanisms from graph perspectives remain undeveloped.

References

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*, CCS '16, pp. 308–318, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341394.

Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the curious abandon honesty: Federated learning is not private. *arXiv preprint arXiv:2112.02918*, 2021.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

Liam Fowl, Jonas Geiping, Steven Reich, Yuxin Wen, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. Decepticons: Corrupted transformers breach privacy in federated learning for language models. *arXiv preprint arXiv:2201.12675*, 2022a.

Liam H Fowl, Jonas Geiping, Wojciech Czaja, Micah Goldblum, and Tom Goldstein. Robbing the fed: Directly obtaining private data in federated learning with modified models. In *International Conference on Learning Representations*, 2022b.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients - how easy is it to break privacy in federated learning? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 16937–16947. Curran Associates, Inc., 2020.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and

Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Samyak Gupta, Yangsibo Huang, Zexuan Zhong, Tianyu Gao, Kai Li, and Danqi Chen. Recovering private text in federated learning of language models. *arXiv preprint arXiv:2205.08514*, 2022.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Yu Rong, Peilin Zhao, Junzhou Huang, Murali Annavaram, and Salman Avestimehr. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *CoRR*, abs/2104.07145, 2021.

Yangsibo Huang, Zhao Song, Kai Li, and Sanjeev Arora. InstaHide: Instance-hiding schemes for private distributed learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4507–4518. PMLR, 13–18 Jul 2020.

Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, and Kurt Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2592–2600, 2016.

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.

Xiao Jin, Pin-Yu Chen, Chia-Yi Hsu, Chia-Mu Yu, and Tianyi Chen. Cafe: Catastrophic data leakage in vertical federated learning. *Advances in Neural Information Processing Systems*, 34:994–1006, 2021.

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.

Arthur Jochems, Timo M Deist, Issam El Naqa, Marc Kessler, Chuck Mayo, Jackson Reeves, Shruti Jolly, Martha Matuszak, Randall Ten Haken, Johan van Soest, et al. Developing and validating a survival prediction model for nsclc patients through distributed learning across 3 countries. *International Journal of Radiation Oncology* Biology* Physics*, 99(2):344–352, 2017.

Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1):312–320, 2005.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52, 2010.

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 583–598, Broomfield, CO, 2014. USENIX Association. ISBN 978-1-931971-16-4.

H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016a.

H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016b.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs, 2020.

Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pp. 287–297. Springer, 2008.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *CoRR*, abs/1909.13021, 2019.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021.

Simone Scardapane, Indro Spinelli, and Paolo Di Lorenzo. Distributed training of graph convolutional networks. *IEEE Transactions on Signal and Information Processing over Networks*, 7:87–100, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Zihan Wang, Jason Lee, and Qi Lei. Reconstructing training data from model gradient, provably. In *International Conference on Artificial Intelligence and Statistics*, pp. 6595–6612. PMLR, 2023.

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

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019.

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*, pp. 16337–16346, 2021.

Huanding Zhang, Tao Shen, Fei Wu, Mingyang Yin, Hongxia Yang, and Chao Wu. Federated graph learning–a position paper. *arXiv preprint arXiv:2105.11099*, 2021.

Zaixi Zhang, Qi Liu, Zhenya Huang, Hao Wang, Chee-Kong Lee, and Enhong Chen. Model inversion attacks against graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–13, 2022. doi: 10.1109/TKDE.2022.3207915.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *CoRR*, abs/2001.02610, 2020a.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020b.

Junyi Zhu and Matthew Blaschko. R-gap: Recursive gradient attack on privacy. *arXiv preprint arXiv:2010.07733*, 2020.

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 (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

A Extracting ground-truth labels

In a classification task, it is possible to extract the ground-truth labels from the gradients as shown in Zhao et al. (2020a). Similary, it is possible to extract the ground-truth labels in a graph/node classification task. We demonstrate how to extract the ground-truth labels when cross-entropy loss is used as defined in equation 5. The gradients of the loss with respect to $\mathbf{p}_i$ is given by

$$g_i = \frac{\partial \mathcal{L}}{\partial \mathbf{p}_i} = \begin{cases} -1 + \left(\frac{e^{\mathbf{p}_i}}{\sum_j e^{\mathbf{p}_j}}\right) & \text{if } i = k, \\ \left(\frac{e^{\mathbf{p}_i}}{\sum_j e^{\mathbf{p}_j}}\right) & \text{else.} \end{cases} \tag{22}$$

Since $\left(\frac{e^{\mathbf{p}_i}}{\sum_j e^{\mathbf{p}_j}}\right) \in (0,1)$ therefore $g_i \in (-1,0)$ when $i = k$ and $g_i \in (0,1)$ otherwise. Now consider the node classification task for a GCN layer as defined in equation 3. For the final layer of the model with output as $\mathbf{h}_v^L = \mathbf{p}$ we can write the gradients of the loss as follows

$$\nabla_{(\mathbf{W})_i}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial (\mathbf{W})_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{p}_i}\frac{\partial \mathbf{p}_i}{\partial (\mathbf{W})_i} \tag{23}$$

$$= g_i \cdot \frac{\partial \left(\sigma\left(\mathbf{h}_v^{\text{agg}}(\mathbf{W})_i^\top + \mathbf{b}_i\right)\right)}{\partial (\mathbf{W})_i} \tag{24}$$

$$= g_i \cdot \lambda_i \cdot \mathbf{h}_v^{\text{agg}}. \tag{25}$$

As $\mathbf{h}_v^{\text{agg}}$ is independent of the logit index i, and $\lambda_i \geq 0$ because of the sigmoid non-linearity, the ground-truth label k can be identified by just checking the signs of $\nabla_{(\mathbf{W})_i}\mathcal{L}$ for all i. Formally, the ground-truth label k is the one for which the following condition holds

$$k = i \text{ s.t. } \nabla_{(\mathbf{W})_i}\mathcal{L} \cdot \nabla_{(\mathbf{W})_j}\mathcal{L} \leq 0, \forall i \neq j. \tag{26}$$

B Proof of Proposition 1 for GraphSAGE

For GraphSAGE layer defined in equation 1 the gradient equations for the first layer can be written as follows

$$\mathbf{h}_v = \sigma(\tilde{\mathbf{h}}_v), \tag{27}$$

$$\tilde{\mathbf{h}}_v = \mathbf{x}_v^{\text{agg}}\mathbf{W}_1^\top + \mathbf{x}_v\mathbf{W}_2^\top + \mathbf{b}, \tag{28}$$

$$\nabla_{(\mathbf{W}_1)_i}\mathcal{L}_v = \nabla_{(\tilde{\mathbf{h}}_v)_i}\mathcal{L}_v \cdot \nabla_{(\mathbf{W}_1)_i}(\tilde{\mathbf{h}}_v)_i, \tag{29}$$

$$\nabla_{(\mathbf{W}_1)_i}\mathcal{L}_v = \nabla_{(\mathbf{b})_i}\mathcal{L}_v \cdot \mathbf{x}_v^{\text{agg}} \tag{30}$$

C Proof of Proposition 2

For a GraphSAGE layer as shown in equation 1 the target nodal features can be recovered as the following

$$\mathbf{h}_v = \sigma(\tilde{\mathbf{h}}_v), \tag{31}$$

$$\tilde{\mathbf{h}}_v = \mathbf{x}_v^{\text{agg}}\mathbf{W}_1^\top + \mathbf{x}_v\mathbf{W}_2^\top + \mathbf{b}, \tag{32}$$

$$\nabla_{(\mathbf{W}_2)_i}\mathcal{L}_v = \nabla_{(\tilde{\mathbf{h}}_v)_i}\mathcal{L}_v \cdot \nabla_{(\mathbf{W}_2)_i}(\tilde{\mathbf{h}}_v)_i, \tag{33}$$

$$\nabla_{(\mathbf{W}_2)_i}\mathcal{L}_v = \nabla_{(\mathbf{b})_i}\mathcal{L}_v \cdot \mathbf{x}_v. \tag{34}$$

The target nodal features can henceforth be recovered as $\mathbf{x}_v = \nabla_{(\mathbf{W}_2)_i}\mathcal{L}_v / \nabla_{(\mathbf{b})_i}\mathcal{L}_v$ as long as $\nabla_{(\mathbf{b})_i}\mathcal{L}_v \neq 0$.

D Proof of Proposition 6

Proof. For a GraphSAGE framework as shown in equation 6, let $\tilde{\mathbf{H}}$ be the hidden node representation matrix before applying the non-linearity as shown below

$$\mathbf{H} = \sigma(\tilde{\mathbf{H}}), \tag{35}$$

$$\tilde{\mathbf{H}} = \tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_1^\top + \mathbf{X}\mathbf{W}_2^\top + \overrightarrow{\mathbf{1}}\mathbf{b}. \tag{36}$$

The derivatives of the loss function with respect to the weights $\mathbf{W}_1$,$\mathbf{W}_2$, and bias $\mathbf{b}$ of the first convolutional layer can be written as

$$\frac{\partial \mathcal{L}}{\partial (\mathbf{W}_1)_i} = \sum_{k=1}^{N} \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{H}}_{ki}} \frac{\partial \tilde{\mathbf{H}}_{ki}}{\partial (\mathbf{W}_1)_i} = \sum_{k=1}^{N} \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{H}}_{ki}} (\tilde{\mathbf{A}}\mathbf{X})_k, \tag{37}$$

$$\frac{\partial \mathcal{L}}{\partial (\mathbf{W}_2)_i} = \sum_{k=1}^{N} \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{H}}_{ki}} (\mathbf{X})_k, \tag{38}$$

$$\frac{\partial \mathcal{L}}{\partial (\mathbf{b})_i} = \sum_{k=1}^{N} \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{H}}_{ki}}, \tag{39}$$

where ∂ denotes the partial derivative operator. Equations equation 37-equation 39 can be written in matrix form as the following

$$(\tilde{\mathbf{A}}\mathbf{X})^\top \nabla_{\tilde{\mathbf{H}}} \mathcal{L} = \nabla_{\mathbf{W}_1} \mathcal{L}^\top, \tag{40}$$

$$\mathbf{X}^\top \nabla_{\tilde{\mathbf{H}}} \mathcal{L} = \nabla_{\mathbf{W}_2} \mathcal{L}^\top, \tag{41}$$

$$\nabla_{\tilde{\mathbf{H}}} \mathcal{L}^\top \cdot \overrightarrow{\mathbf{1}} = \nabla_{\mathbf{b}} \mathcal{L}^\top. \tag{42}$$

It can be observed from equation 40-equation 41 that there are two unknowns $\tilde{\mathbf{A}}$ and $\nabla_{\tilde{\mathbf{H}}}\mathcal{L}$. However $\tilde{\mathbf{A}}$ and $\nabla_{\tilde{\mathbf{H}}}\mathcal{L}$ can be computed directly from the above equations. Given equation 41 and the fact that $\mathbf{X}$ is full-row rank, $\nabla_{\tilde{\mathbf{H}}}\mathcal{L}$ can be obtained as follows

$$\nabla_{\tilde{\mathbf{H}}} \mathcal{L} = (\mathbf{X}^\top)^\dagger \nabla_{\mathbf{W}_2} \mathcal{L}^\top. \tag{43}$$

Plugging the value of $\nabla_{\tilde{\mathbf{H}}}\mathcal{L}$ in equation 40, $\tilde{\mathbf{A}}$ can be calculated as

$$\tilde{\mathbf{A}} = \mathbf{X}(\nabla_{\mathbf{W}_2} \mathcal{L})^\dagger (\nabla_{\mathbf{W}_1} \mathcal{L}) \mathbf{X}^\dagger \tag{44}$$

. □

E Additional Graph Classification results on TUDatasets

All attack settings considered in previous experiments in Section 5 are defined within the context of social networks. In order to assess the efficacy of the attack on alternative datasets, such as molecular datasets which are commonly employed in graph classification tasks, we extend our analysis to the TUDataset Morris et al. (2020).

For the graph classification task, we conduct the attack on the following three datasets from the TUDataset repository Morris et al. (2020).

- **MUTAG** Debnath et al. (1991): It consists of 188 chemical compounds from two classes. The vertices represent atoms and edges represent chemical bonds. Node features denote the atom type represented by one-hot encoding. Labels represent their mutagenic effect on a specific gram-negative bacterium.

- **COIL-RAG** Riesen & Bunke (2008): Dataset in computer vision where images of objects are represented as region adjacency graphs. Each node in the graph is a specific part in the original image. Graph labels denote the object type.
- **FRANKENSTEIN** Kazius et al. (2005): It contains graphs representing chemical molecules, where the vertices are chemical atoms and edges represent the bonding. Node features denote the atom type. Binary labels represent mutagenecity.

The average number of nodes in the graphs of these datasets is significantly smaller than social networks (see Table 11). However, unlike social networks, these graphs might not necessarily exhibit properties like feature smoothness and sparsity. Hence, for all the attacks we use the objective function as defined in equation 12. The results in all attack settings are again consistent with the theoretical analysis.

The results are obtained by randomly selecting 20 graphs from the datasets. The selected graphs are fixed for different parameter selections for fair comparison. In this case since the graphs are small we can directly use the Mean Absolute Error (MAE) of the recovered adjacency matrix to measure the performance of recovering the underlying graph structure. Specifically, the MAE is given as follows

$$\text{MAE}(\mathbf{A}, \hat{\mathbf{A}}) = \frac{2 * |\text{tri}_{\text{lower}}(\mathbf{A}) - \text{tri}_{\text{lower}}(\hat{\mathbf{A}})|}{N(N+1)} \tag{45}$$

Also, instead of directly sampling edges from the probabilistic adjacency matrix as given in Section 3.2, the MAE is also evaluated with and without thresholding the recovered adjacency matrix. We apply min-max normalization to the resulting $\hat{\mathbf{A}}$ before thresholding. After applying the threshold, $\hat{\mathbf{A}}_\tau := th(\hat{\mathbf{A}}, \tau)$ is obtained, where $th(\mathbf{Z}, \tau)$ is a entry-wise operator that returns 0 if $z_{ij} < \tau$, and returns 1 otherwise.

Table 11: Dataset statistics (Graph Classification). Edge density is calculated based on (#Avg.Edges/#Edges in the fully connected graph).

Dataset	**#Avg. Nodes**	**#Avg. Edge**	**#Edge Density**	**#Labels**
MUTAG	17.93	19.79	0.1304	2
COIL-RAG	3.01	3.02	0.9983	100
FRANK.	16.90	17.88	0.1331	2

Graph Attacker-a. Table 12 shows the MAE of recovering $\mathbf{A}$ with thresholding (for different values of τ) and without (N/A in the table). The attack achieves the best results on the FRANKENSTEIN dataset while using GraphSAGE. Each node in the FRANKENSTEIN dataset has $D = 780$ features, whereas MUTAG and COIL-RAG have only $D = 7$ and and $D = 64$ features respectively. Moreover, the average number of nodes (N) per graph in FRANKENSTEIN, MUTAG and COIL-RAG are 16.9, 17.9 and 3.01 respectively. This implies that $\mathbf{X} \in \mathbb{R}^{N \times D}$ is full-row rank with high probablity in COIL-RAG and FRANKENSTEIN. Meanwhile, the node features in the MUTAG dataset consists of only binary values. It can be observed from the experimental results that the best and worst results are obtained on FRANKENSTEIN and MUTAG respectively. This is also consistent with Proposition 6 which requires $\mathbf{X}$ to be full row-rank for successfully recovering $\tilde{\mathbf{A}}$. The recovery performance improves for COIL-RAG with GraphSAGE at $\tau = 0.8$. The FRANKENSTEIN dataset is recovered accurately even with the GCN framework.

Table 12: Recovery of $\mathbf{A}$ (MAE) for Graph Attacker-a

(a) GraphSAGE

τ	MUTAG	COIL-RAG	FRANK.($\times 10^{-4}$)
N/A	0.25±0.10	0.02±0.04	0.08±0.3
0.2	0.27±0.11	0.03±0.07	0.0±0.0
0.4	0.24±0.11	0.03±0.06	0.0±0.0
0.6	0.23±0.11	0.02 ± 0.04	0.0±0.0
0.8	0.23±0.10	0.003±0.13	0.0±0.0

(b) GCN

τ	MUTAG	COIL-RAG	FRANK. ($\times 10^{-2}$)
N/A	0.20±0.04	0.22±0.11	4.58±2.59
0.2	0.27±0.07	0.16±0.09	4.74±3.3
0.4	0.19±0.06	0.16±0.09	4.20±2.15
0.6	0.15±0.04	0.16±0.09	4.08±1.96
0.8	0.14±0.01	0.26±0.267	4.22±2.04

Table 13: Recovery of $\mathbf{X}$ (RNMSE) for Graph Attacker-b.

	MUTAG	COIL-RAG	FRANK.
GraphSAGE($\times 10^{-2}$)	0.07±0.06	6.91±7.1	21.95±16.05
GCN	8.80±1.27	1.51±0.17	2.68±0.30

Table 14: Recovery of $\mathbf{A}$ (MAE) in for Graph Attacker-c with different thresholds.

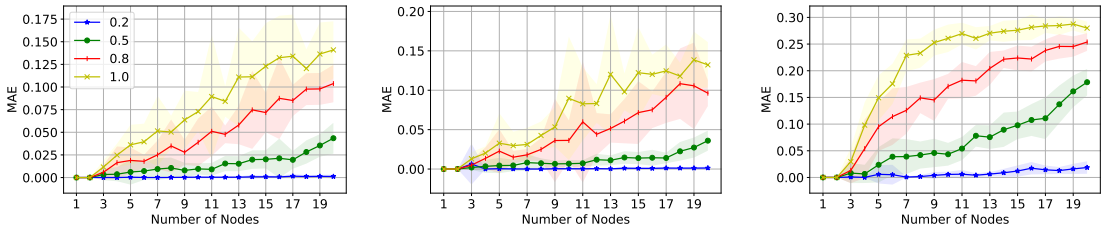
(a) GraphSAGE

τ	MUTAG	COIL-RAG	FRANK.
N/A	0.37±0.10	0.18±0.11	0.41±0.17
0.2	0.47 ±0.13	0.20±0.11	0.64±0.25
0.4	0.43±0.13	0.15±0.14	0.50±0.24
0.5	0.39±0.13	0.14±0.17	0.38±0.20
0.6	0.32±0.08	0.11±0.18	0.29±0.15
0.8	0.26±0.06	0.19±0.18	0.19±0.10

(b) GCN

τ	MUTAG	COIL-RAG	FRANK.
N/A	0.48±0.05	0.20±0.12	0.41±0.08
0.2	0.68 ±0.06	0.15±0.08	0.64±0.08
0.4	0.55 ±0.07	0.14±0.08	0.46±0.12
0.5	0.47 ±0.07	0.14±0.08	0.38±0.11
0.6	0.40±0.07	0.14±0.08	0.32±0.10
0.8	0.30±0.05	0.32±0.34	0.19±0.08

Graph Attacker-b. Table 13 lists the RNMSE of recovering $\mathbf{X}$. It can be observed that the attack is able to recover $\mathbf{X}$ with high accuracy for GraphSAGE across all three datasets. For the GCN framework, it can be observed that the algorithm fails to recover $\mathbf{X}$.



(a) Entries of $\hat{\mathbf{X}}^1$ and $\hat{\mathbf{A}}^1$ set to 1 (b) Entries of $\hat{\mathbf{X}}^1$ and $\hat{\mathbf{A}}^1$ set to 0.5 (c) Entries of $\hat{\mathbf{X}}^1$ and $\hat{\mathbf{A}}^1$ sampled from $\mathcal{N}(0,1)$

Figure 6: Recovery of different graph structures (MAE) for Graph Attacker-c using GraphSAGE.

Graph Attacker-c. In this scenario, since the attacker has no knowledge of the graph data, we try different settings and observe their effects on the recovery. To improve the attack for Graph Attacker-c where we have no prior knowledge of the graph we utilize various initialization and thresholding strategies. Specifically, since the optimization process given in the algorithms can be greatly improved by utilizing different starting points, we initialize the dummy node feature and adjacency matrices with different values. In the case of thresholding, instead of sampling the edges given the probabilistic adjacency matrix $\hat{\mathbf{A}}^{P+1}$, we use various threshold values to sample edges.

The effect of initialization: Previous studies have established that the quality of recovery heavily depends on the initialization Gupta et al. (2022); Geiping et al. (2020). The closer the initialization is to the original data, the better the recovery. An initialization value of 0.1 implies that all the entries of $\hat{\mathbf{X}}$ and $\hat{\mathbf{A}}$ are initialized to 0.1 and similarly $\mathcal{N}(0,1)$ implies that all entries are initialized from the standard normal distribution. Since the values of $\mathbf{X}$ and $\mathbf{A}$ lie in the range $[0,1]$ for all three datasets, we tested the effect of different initialization points in the same range. Specifically, we tested the initialization values as $\{0, 0.1, 0.5, 1, \mathcal{N}(0,1)\}$. The results of the attack with different initialization strategies is shown in Table 16. The leftmost column gives the initialization values of the dummy adjacency and node feature matrices. For the adjacency matrix, the results are listed with and without thresholding. Similar to Scenario 2, we apply

Table 15: Recovery of $\mathbf{X}$ (RNMSE) and $\mathbf{A}$ (MAE) for Graph Attacker-c.

Dataset	GraphSAGE			GCN		
	$\hat{\mathbf{X}}$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$	$\hat{\mathbf{X}}$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$
MUTAG	0.0027±0.0078	0.22±0.09	0.21±0.11	1.07±0.20	0.24±0.02	0.14±0.04
COIL-RAG	0.005 ± 0.0015	0.01±0.03	0.001±0.006	0.74±0.13	0.05±0.10	0.03±0.10
FRANK.	0.01±0.006	0.004 ± 0.002	0.00±0.00	0.63±0.18	0.07±0.02	0.007±0.01

min-max normalization (before thresholding) and a threshold value of 0.5 is chosen. A more compact version of the results showing the best performance over all initialization strategies is shown in Table 15. It can be observed that for GraphSAGE, the attack can accurately recover both $\mathbf{X}$ and $\mathbf{A}$ in the MUTAG and FRANKENSTEIN datasets. While using GCN, even though $\mathbf{X}$ cannot be recovered, $\mathbf{A}$ can be recovered accurately.

For GraphSAGE in Table 16a, it can be observed that even without any knowledge of the graph data, the attack can accurately recover both $\mathbf{X}$ and $\mathbf{A}$ just by utilising a good initialization strategy. For example, in Table 16a for the FRANKENSTEIN dataset, the attack gives a high error for both $\mathbf{A}$ and $\mathbf{X}$ while using $\mathcal{N}(0,1)$ as the initialization. However, the MAE drops by orders of magnitude when we initialize $\mathbf{X}$ and $\mathbf{A}$ to a value of 0.2. Even for the GCN framework in Table 16b, while the data cannot be recovered in most of the cases, it can be observed that a good initialization point can leak $\mathbf{A}$ in COIL-RAG and FRANKENSTEIN datasets.

The effect of threshold: To explore the effect of thresholding on the adjacency matrix, we examined the MAE across all three datasets using different threshold values. Table 14 lists the MAE of recovering $\mathbf{A}$ at different thresholds. It can be observed that unlike Scenario 2, the performance of the algorithm stays roughly the same for both GraphSAGE and GCN. Also, the FRANKENSTEIN dataset is no longer vulnerable to the attack unlike Scenario 2. Few of the threshold values reduce the MAE by a considerable amount. For example, $\tau = 0.8$ works well for MUTAG and FRANKENSTEIN in the case of GraphSAGE, and for MUTAG in the case of GCN.

Table 16: Recovery of $\mathbf{X}$ (RNMSE) and $\mathbf{A}$ (MAE) for Graph Attacker-c .

(a) GraphSAGE

init	MUTAG			COIL-RAG ($\times 10^{-2}$)			FRANK. ($\times 10^{-2}$)		
	$\hat{\mathbf{X}}(\times 10^{-2})$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$	$\hat{\mathbf{X}}$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$	$\hat{\mathbf{X}}$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$
0.0	4±1	0.37±0.10	0.47±0.13	31±8	1±9	4±8	29±14	27±15	21±15
0.1	0.9±0.9	0.23±0.08	0.26 ± 0.14	22±1	9±8	3±7	4±5	1±2	0.6±1
0.2	0.8±2	0.22±0.09	0.21±0.11	9±5	5±4	0.5±2	1±0.6	0.4±0.2	0.0±0.0
0.5	0.3±0.4	0.23±0.10	0.22±0.12	0.6±1	1±3	0.1±0.6	1±0.9	0.4±0.3	0.0±0.0
1.0	0.2±0.7	0.23±0.10	0.22±0.11	0.5±1	1±3	0.1±0.6	1±4	0.4 ±0.8	0.06±0.2
$\mathcal{N}(0,1)$	6 ± 3	0.37 ± 0.10	0.39 ± 0.13	26 ± 12	18 ± 11	14 ± 17	66 ± 43	41 ± 17	38 ± 20

(b) GCN

init	MUTAG			COIL-RAG			FRANK.		
	$\hat{\mathbf{X}}$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$	$\hat{\mathbf{X}}$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$	$\hat{\mathbf{X}}$	$\hat{\mathbf{A}}$	$\hat{\mathbf{A}}_{0.5}$
0.0	1.38±0.31	0.25±0.05	0.21±0.07	1.96±2.2	0.05±0.10	0.03±0.10	1.11±0.27	0.11±0.03	0.03±0.04
0.1	1.06 ± 0.15	0.24±0.02	0.14±0.04	1.54±0.1.48	0.43±0.44	0.42±0.47	0.66±0.15	0.07±0.02	0.007±0.01
0.2	1.07±0.20	0.25 ± 0.02	0.14±0.04	1.12±0.71	0.50±0.40	0.47±0.43	0.63±0.18	0.08±0.02	0.007±0.01
0.5	1.47±0.43	0.33±0.06	0.23±0.09	0.75±0.14	0.17±0.33	0.17±0.35	0.72±0.25	0.14±0.05	0.05±0.07
1.0	1.15±0.33	0.34±0.04	0.26±0.06	0.74±0.13	0.76±0.31	0.78±0.32	0.81±0.23	0.19±0.05	0.13±0.06
$\mathcal{N}(0,1)$	3.26±0.36	0.48 ± 0.05	0.47 ± 0.07	7.98 ± 1.50	0.20 ± 0.12	0.14 ± 0.08	4.54 ± 0.44	0.41 ± 0.08	0.388 ± 0.11

Vulnerable graph structures: In order to identify vulnerable graph structures, we tested the performance of the attack in attacking graphs with varying number of nodes and edges for the GraphSAGE framework. Specifically, the performance was tested on Erdos-Renyi graphs (with varying edge probabilities). Figure 6 shows the MAE of recovering $\mathbf{A}$ for different graphs. The number of features for all graphs is set to $D = 10$, and their values are sampled from the standard normal distribution. Each subfigure represents different initialization strategies for $\hat{\mathbf{X}}$ and $\hat{\mathbf{A}}$. For each graph with a specific number of nodes, the MAE is averaged over 20 runs of the attack. As the number of nodes in a graph increases, the quality of recovery decreases. More importantly, it is evident that the vulnerability to the attack increases with decreasing sparsity. Fully connected graphs (ER graphs with edge probability 1.0) are the least vulnerable to the attack.