GIF: A General Graph Unlearning Strategy via Influence Function

Jiancan Wu*
University of Science and Technology of China, China
wujcan@gmail.com

Yi Yang*
University of Science and Technology of China, China
yanggnay@mail.ustc.edu.cn

Yuchun Qian
University of Science and Technology of China, China
daytoy_qyc@mail.ustc.edu.cn

Yongduo Sui
University of Science and Technology of China, China
syd2019@mail.ustc.edu.cn

Xiang Wang†
University of Science and Technology of China, China
xiangwang1223@gmail.com

Xiangnan He†
University of Science and Technology of China, China
xiangnanhe@gmail.com

ABSTRACT

With the greater emphasis on privacy and security in our society, the problem of graph unlearning — revoking the influence of specific data on the trained GNN model, is drawing increasing attention. However, ranging from machine unlearning to recently emerged graph unlearning methods, existing efforts either resort to retraining paradigm, or perform approximate erasure that fails to consider the inter-dependency between connected neighbors or imposes constraints on GNN structure, therefore hard to achieve satisfying performance-complexity trade-offs.

In this work, we explore the influence function tailored for graph unlearning, so as to improve the unlearning efficacy and efficiency for graph unlearning. We first present a unified problem formulation of diverse graph unlearning tasks *w.r.t.* node, edge, and feature. Then, we recognize the crux to the inability of traditional influence function for graph unlearning, and devise Graph Influence Function (GIF), a model-agnostic unlearning method that can efficiently and accurately estimate parameter changes in response to a ϵ-mass perturbation in deleted data. The idea is to supplement the objective of the traditional influence function with an additional loss term of the influenced neighbors due to the structural dependency. Further deductions on the closed-form solution of parameter changes provide a better understanding of the unlearning mechanism. We conduct extensive experiments on four representative GNN models and three benchmark datasets to justify the superiority of GIF for diverse graph unlearning tasks in terms of unlearning efficacy, model utility, and unlearning efficiency. Our implementations are available at https://github.com/wujcan/GIF-torch/.

KEYWORDS

Graph Unlearning; Influence Functions, Graph Neural Networks; Model Explainability

*These authors contributed equally to this work.
†Corresponding authors.

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583521

ACM Reference Format:
Jiancan Wu, Yi Yang, Yuchun Qian, Yongduo Sui, Xiang Wang, and Xiangnan He. 2023. GIF: A General Graph Unlearning Strategy via Influence Function . In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583521

1 INTRODUCTION

Machine unlearning [4] is attracting increasing attention in both academia and industry. At its core is removing the influence of target data from the deployed model, as if it did not exist. It is of great need in many critical scenarios, such as (1) the enforcement of laws concerning data protection or user's right to be forgotten [21, 26, 28], and (2) the demands for system provider to revoke the negative effect of backdoor poisoned data [29, 43], wrongly annotated data [25], or out-of-date data [33]. As an emerging topic, machine unlearning in the graph field, termed **graph unlearning**, remains largely unexplored, but is the focus of our work. Specifically, graph unlearning focuses on ruling out the influence of graph-structured data (*e.g.,* nodes, edges, and their features) from the graph neural networks (GNNs) [16, 19, 22, 32, 35, 36, 38]. Clearly, the nature of GNNs — the entanglement between the graph structure and model architecture — is the main obstacle of unlearning. For example, given a node of a graph as the unlearning target, we need to not only remove its own influence, but also offset its underlying effect on neighbors multi-hop away.

Such a structural influence makes the current approaches fall short in graph unlearning. Next we elaborate on the inherent limitations of these approaches:

- A straightforward unlearning solution is retraining the model from scratch, which only uses the remaining data. However, it can be resource-consuming when facing large-scale graphs like social networks and financial transaction networks.
- Considering the unlearning efficiency, some follow-on studies [1, 3, 5, 6, 9] first divide the whole training data into multiple disjoint shards, and then train one sub-model for each shard. When the unlearning request arrives, only the sub-model of shards that comprise the unlearned data needs to be retrained. Then, aggregating the predictions from all the sub-models can get the final prediction. Although this exact way guarantees the removal of all information associated with the unlearned data, splitting data into shards will inevitably destroy the connections in samples, especially for graph data, hence hurting the model performance. GraphEditor [9] is a very recent work that supports

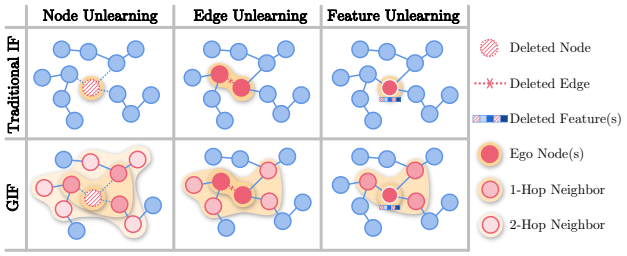

Figure 1: The differences between the traditional influence function method and our GIF. The traditional IF only computes the parameter change if the loss of nodes that are directly affected by the unlearning request (as shown in the colored region) is upweighted by a small infinitesimal amount, while our GIF considers both the directly affected node(s) and the influenced neighborhoods as shown in the bottom subfigures, hence, it can unlearn the data more completely.

exact graph unlearning free from shard model retraining, but is restricted to linear GNN structure under Ridge regression formulation.

- Another line [8, 13, 14, 23, 24, 30] resorts to gradient analysis techniques instead to approximate the unlearning process, so as to avoid retraining sub-models from scratch. Among them, influence function [20] is a promising proxy to estimate the parameter changes caused by a sample removal, which is on up-weighting the individual loss *w.r.t.* the target sample, and then reduces its influence accordingly. However, it only considers the individual sample, leaving the effect of sample interaction and coalition untouched. As a result, it hardly approaches the structural influence of graph, thus generalizing poorly on graph unlearning. To the best of our knowledge, no effort has been made to tailor the influence function for graph unlearning.

To fill this research gap, we explore the graph-oriented influence function in this work. Specifically, we first present a unified problem formulation of graph unlearning *w.r.t.* three essential gradients: node, edge, and feature, thus framing the corresponding tasks of node unlearning, edge unlearning, and feature unlearning. Then, inspecting the conventional influence function, we reveal why it incurs incomplete data removal for GNN models. Taking unlearning on an ego node as an example, it will not only affect the prediction of the node, but also exert influence on the K-hop neighboring nodes, due to the message passing scheme of GNN. Similar deficiencies are observed for edge and feature unlearning tasks. Realizing this, we devise a Graph Influence Function (**GIF**) to consider such structural influence of node/edge/feature on its neighbors. It performs the accurate unlearning answers of simple graph convolutions, and sheds new light on how different unlearning tasks for various GNNs influence the model performance.

We summarize our contributions as follows,

(1) We propose a more general and powerful graph unlearning algorithm **GIF** tailor-made for GNN models, which allows us to estimate the model prediction in advance efficiently and protect privacy.

Table 1: Summary of Notations

Notations	Description
$\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$	Input graph
$\mathcal{D}_0 = \{z_1, z_2, \ldots, z_k\}$	Training set
$f_{\mathcal{G}}, f_{\mathcal{G}\setminus\Delta\mathcal{G}}$	GNN trained on graph $\mathcal{G}$ and $\mathcal{G} \setminus \Delta\mathcal{G}$ from scratch
z_i, z_{test}	Training and test node
$f_{\mathcal{G}}(z_i; \mathcal{G} \setminus \Delta\mathcal{G})$	Prediction of $z_i \in \mathcal{G} \setminus \Delta\mathcal{G}$ using $f_{\mathcal{G}}$
$l\left(f_{\mathcal{G}}(z_i), y_i\right)$	Original loss
$\Delta l\left(f(z_i), y_i\right)$	Influenced loss
$l\left(\hat{f}_{\mathcal{G}}(z_i), y_i\right)$	Final loss

(2) We propose more comprehensive evaluation criteria and major tasks for graph unlearning. Extensive experiments show the effectiveness of GIF.

(3) To our best knowledge, our GIF is one of the first attempts to interpret the black box of graph unlearning process.

2 PRELIMINARY

Throughout this paper, we define the lower-case letters in bold (*e.g.,* $\mathbf{x}$) as vectors. The blackboard bold typefaces (*e.g.,* $\mathbb{R}$) denote the spaces, while the calligraphic font for uppercase letters (*e.g.,* $\mathcal{V}$) denote the sets. The notions frequently used in this paper are summarized in Table 1.

2.1 Graph Unlearning Formulation

In this work, we focus on the problem of node classification. Consider a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$ with $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges. Each node $v_i \in \mathcal{V}$ is associated with an F-dimensional feature vector $\mathbf{x}_i \in \mathcal{X}$. The training set $\mathcal{D}_0$ contains N samples $\{z_1, z_2, ..., z_N\}$, each of which is annotated with a label $y \in \mathcal{Y}$. Given a graph $\mathcal{G}$, the goal of node classification task is to train a GNN model that predicts the label of a node $v \in \mathcal{V}$.

After the arrival of unlearning request $\Delta\mathcal{G} = \{\mathcal{V}^{(rm)}, \mathcal{E}^{(rm)}, \mathcal{X}^{(rm)}\}$, the goal of graph unlearning is to find a mechanism $\mathcal{M}$ that takes $\mathcal{D}_0$, $f_{\mathcal{G}}$ and $\Delta\mathcal{G}$ as input, then outputs a new model $\hat{f}$ parameterized by $\hat{\theta}$ that minimizes the discrepancy $D(\hat{f}, f_{\mathcal{G}\setminus\Delta\mathcal{G}})$, where $f_{\mathcal{G}\setminus\Delta\mathcal{G}}$ is the GNN model retrained from scratch on the remaining graph $\mathcal{G} \setminus \Delta\mathcal{G}$ using the objective similar to Equation (4). The graph unlearning mechanism should satisfy the following criteria:

- **Removal Guarantee**. The primary need of unlearning mechanism is to remove the information of deleted data completely from the trained model, including the deleted data itself and its influences on other samples.
- **Comparable Model Utility**. It is a practical requirement that the unlearning mechanism only brings in a small utility gap in comparison to retraining from scratch.
- **Reduced Unlearning Time**. The unlearning mechanism should be time-efficient as compared to model retraining.
- **Model Agnostic**. The strategy for unlearning can be applied to any GNN model with various architectures, including but not limited to linear GNNs or non-linear GNNs.

Furthermore, we can categorize the task of graph unlearning into fine-grained tasks based on the type of request $\Delta\mathcal{G}$. In this work, we consider the following three types of graph unlearning tasks, while leaving the others in future work:

- **Node Unlearning**: $\Delta\mathcal{G} = \{\mathcal{V}^{(rm)}, \varnothing, \varnothing\}$.
- **Edge Unlearning**: $\Delta\mathcal{G} = \{\varnothing, \mathcal{E}^{(rm)}, \varnothing\}$.
- **Feature Unlearning**: $\Delta\mathcal{G} = \{\varnothing, \varnothing, \mathcal{X}^{(rm)}\}$.

2.2 Abstract Paradigm of GNNs

The core of GNNs is to apply the neighborhood aggregation on $\mathcal{G}$, recursively integrate the vectorized information from neighboring nodes, and update the representations of ego nodes. Thereafter, a classifier is used to generate the final predictions.

Neighborhood Aggregation. The aggregation scheme consists of the following two crucial components:
(1) Representation aggregation layers. After k layers, a node's representation is able to capture the structural information within its k-hop neighbors. The k-th layer is formulated as:

$$\mathbf{a}_i^{(k)} = f_{\text{aggregate}}\left(\mathbf{z}_j^{(k-1)} | j \in \mathcal{N}_i\right),$$
$$\mathbf{z}_i^{(k)} = f_{\text{combine}}\left(\mathbf{z}_i^{(k-1)}, \mathbf{a}_i^{(k)}\right), \tag{1}$$

where $\mathbf{a}_i^{(k)}$ denotes the aggregation of vectorized information from node i's neighborhood $\mathcal{N}_i$ and $\mathbf{z}_i^{(k)}$ is the representation of node i after k aggregation layers, which integrates $\mathbf{a}_i^{(k)}$ with its previous representation $\mathbf{z}_i^{(k-1)}$. Specially, $\mathbf{z}_i^{(0)} = \mathbf{x}_i$. The designs for aggregation function $f_{\text{aggregate}}(\cdot)$ and combination function $f_{\text{combine}}(\cdot)$ vary in different studies [15, 19, 32, 39].
(2) Readout layer. Having obtained the representations at different layers, the readout function generates the final representations for each node:

$$\mathbf{z}_i = f_{\text{readout}}\left(\{\mathbf{z}_i^{(k)} | k = [0, \cdots, K}\}\right), \tag{2}$$

which can be simply set as the last-layer representation [31, 41], concatenation [34], or weighted summation [16] over the representations of all layers.

Prediction and Optimization. After that, a classifier (or prediction layer) is built upon the final representations of nodes to predict their classes. A classical solution is an MLP network with *Softmax* activation,

$$\hat{y}_i = f_{\mathcal{G}}(z_i) = \text{Softmax}\left(\mathbf{z}_i W^{(pred)}\right), \tag{3}$$

where $f_{\mathcal{G}}$ is a GNN model built on $\mathcal{G}$ parameterized by $\theta_0, W^{(pred)} \in \theta_0$ is the parameters of the classifier, $\hat{y}_i$ is the predicted probability of sample z_i. The optimal model parameter θ_0 is obtained by minimizing the following objective:

$$\theta_0 = \arg\min_{\theta} \mathcal{L}_0, \quad \mathcal{L}_0 = \sum_{z_i \in \mathcal{D}_0} l\left(f_{\mathcal{G}}(z_i), y_i\right), \tag{4}$$

where l is the loss function defined on the prediction of each sample $f_{\mathcal{G}}(z_i)$ and its corresponding label y_i, $\mathcal{L}_0$ is the total loss for all training samples.

2.3 Traditional Influence Functions

The concept of influence function comes from robust statistics, which measures how the model parameters change when we upweight a sample by an infinitesimally-small amount.

Proposition 1. *(Traditional Influence Functions): Assuming we get a node unlearning request $\Delta\mathcal{G} = \{\mathcal{V}^{(rm)}, \varnothing, \varnothing\}$ for the deployed model $f_{\mathcal{G}}$ learned under Equation* (4)*, then the estimated parameter change using traditional influence function is $\hat{\theta} - \theta_0 \approx H_{\theta_0}^{-1} \nabla_{\theta_0} \mathcal{L}_{\Delta\mathcal{G}}$, where θ_0 and $\hat{\theta}$ are the model parameters of the learned model before and after unlearning, respectively, H_{θ_0} is the Hessian matrix of $\mathcal{L}_0$* w.r.t. θ_0, $\mathcal{L}_{\Delta\mathcal{G}} = \sum_{z_i \in \Delta\mathcal{D}} l\left(f_{\mathcal{G}}(z_i), y_i\right)$

Following the traditional influence function algorithms [10], the loss defined on original training set for node classification tasks can be described as

$$\mathcal{L}_0 = \sum_{z_i \in \mathcal{D}_0} l\left(f_{\mathcal{G}}(z_i), y_i\right). \tag{5}$$

Assume $\mathcal{L}_0$ is twice-differentiable and strictly convex[1]. Thus the hessian matrix H_{θ_0} is positive-definite and invertible.

$$H_{\theta_0} = \sum_{z_i \in \mathcal{D}_0} \nabla_{\theta_0}^2 l\left(f_{\mathcal{G}}(z_i), y_i\right), \tag{6}$$

then, the final loss can be written as

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}, \quad \mathcal{L} = \sum_{z_i \in \mathcal{D}_0 \setminus \Delta\mathcal{D}} l\left(f_{\mathcal{G}\setminus\Delta\mathcal{G}}(z_i), y_i\right). \tag{7}$$

For a small perturbation $\epsilon\mathcal{L}_{\Delta\mathcal{G}}$(ϵ is scalar), the parameter change θ_ϵ can be expressed as

$$\theta_\epsilon = \arg\min_{\theta}(\mathcal{L}_0 + \epsilon\mathcal{L}_{\Delta\mathcal{G}}). \tag{8}$$

Note that θ_0 and θ_ϵ are the minimums of equations (5) and (8) respectively, we have the first-order optimality conditions:

$$0 = \nabla_{\theta_\epsilon}\mathcal{L}_0 + \epsilon\nabla_{\theta_\epsilon}\mathcal{L}_{\Delta\mathcal{G}}, \quad 0 = \nabla_{\theta_0}\mathcal{L}_0. \tag{9}$$

Given that $\lim_{\epsilon\to 0}\theta_\epsilon = \theta_0$, we keep one order Taylor expansion at the point of θ_0. Denote $\Delta\theta = \theta_\epsilon - \theta_0$, we have

$$0 \approx \Delta\theta\left(\nabla_{\theta_0}^2\mathcal{L}_0 + \epsilon\nabla_{\theta_0}^2\mathcal{L}_{\Delta\mathcal{G}}\right) + \left(\epsilon\nabla_{\theta_0}\mathcal{L}_{\Delta\mathcal{G}} + \nabla_{\theta_0}\mathcal{L}_0\right). \tag{10}$$

Since $\Delta\mathcal{G}$ is a tiny subset of $\mathcal{G}$, we can neglect the term $\nabla_{\theta_0}^2\mathcal{L}_{\Delta\mathcal{G}}$ here. With a further assumption that

$$\text{When } \epsilon = -1, \quad \mathcal{L} = \mathcal{L}_0 + \epsilon\mathcal{L}_{\Delta\mathcal{G}}, \tag{11}$$

we finally get the estimated parameter change

$$\hat{\theta} - \theta_0 = \theta_{\epsilon=-1} - \theta_0 \approx H_{\theta_0}^{-1}\nabla_{\theta_0}\mathcal{L}_{\Delta\mathcal{G}}. \tag{12}$$

3 GIF FOR GRAPH UNLEARNING

3.1 Graph Influence Functions

The situation where Equation (11) holds is that for any sample $z_i \in \mathcal{D} \setminus \Delta\mathcal{D}$, its prediction from $f_{\mathcal{G}}$ and $f_{\mathcal{G}\setminus\Delta\mathcal{G}}$ are identical. We can formulate such an assumption as follows:

$$\sum_{z_i \in \mathcal{D}_0 \setminus \Delta\mathcal{D}} l\left(f_{\mathcal{G}\setminus\Delta\mathcal{G}}(z_i), y_i\right) = \sum_{z_i \in \mathcal{D}_0} l\left(f_{\mathcal{G}}(z_i), y_i\right) - \sum_{z_i \in \Delta\mathcal{D}_0} l\left(f_{\mathcal{G}}(z_i), y_i\right). \tag{13}$$

[1]We provide a discussion of the non-convex and non-differential cases in Appendix A.

This suggests that the perturbation in one sample will not affect the state of other samples. Although it could be reasonable in the domain of text or image data, while in graph data, nodes are connected via edges by nature and rely on each other. The neighborhood aggregation mechanism of GNNs further enhances the similarity between linked nodes. Therefore, the removal of $\Delta\mathcal{G}$ will inevitably influence the state of its multi-hop neighbors. With these in mind, we next derive the graph-oriented influence function for graph unlearning in detail. Following the idea of data perturbation in traditional influence function [20], we correct Equation (8) by taking graph dependencies into consideration.

$$\hat{\theta}_\epsilon = \arg\min_\theta(\mathcal{L}_0 + \epsilon\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}), \quad \Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})} = \sum_{z_i\in\mathcal{D}} \hat{l}\,(z_i, y_i), \tag{14}$$

$$\hat{l}\,(z_i, y_i) = \begin{cases} l(f_{\mathcal{G}}(z_i), y_i), & z_i \in \Delta\mathcal{G} \\ l(f_{\mathcal{G}}(z_i), y_i) - l(f_{\mathcal{G}\setminus\Delta\mathcal{G}}(z_i), y_i), & z_i \text{ is influenced by } \Delta\mathcal{G} \\ 0, & \text{other nodes} \end{cases} \tag{15}$$

Utilizing the minimum property of $\hat{\theta}_\epsilon$ and denoting $\Delta\hat{\theta}_\epsilon = \hat{\theta}_\epsilon - \theta_0$, we similarly make a one-order expansion and deduce the answer.

$$0 \approx \Delta\hat{\theta}_\epsilon(\nabla^2_{\theta_0}\mathcal{L}_0 + \epsilon\nabla^2_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}) + (\epsilon\nabla_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})} + \nabla_{\theta_0}\mathcal{L}_0). \tag{16}$$

We thus have the following theorem:

Theorem 2. *(Graph-oriented Influence Functions) A general GIF algorithm for unlearning tasks has a closed-form expression:*

$$\hat{\theta} - \theta_0 \approx H^{-1}_{\theta_0}\nabla_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}. \tag{17}$$

A remaining question is how large the influenced region is. To answer this question, we first define the k-hop neighbors of node z_i and edge e_i (denoted as $\mathcal{N}_k(z_i)$ and $\mathcal{N}_k(e_i)$, respectively) following the idea of shortest path distance (SPD). Specifically,

$$\mathcal{N}_k(z_i) = \{z_j | 1 \leqslant \mathrm{SPD}(z_j, z_i) \leqslant k\}, \tag{18}$$

$$\mathcal{N}_k(e_i) = \mathcal{N}_k(z'_1) \bigcup \mathcal{N}_k(z'_2) \bigcup \{z'_1, z'_2\}, \tag{19}$$

where z'_1 and z'_2 are the two endpoints of edge e_i. Then we have the following Lemma.

Lemma 3. *Let A and D denote the adjacency matrix and the corresponding degree matrix of graph $\mathcal{G}$. Suppose the normalized propagation matrix $\hat{A}$ is defined as $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. Then the influenced region* w.r.t. *unlearning request $\Delta\mathcal{G}$ for different types of graph unlearning task is:*

- *For Node Unlearning request $\Delta\mathcal{G} = \{\mathcal{V}^{(rm)}, \varnothing, \varnothing\}$, the influenced region is $\mathcal{N}_k(\mathcal{V}^{(rm)}) = \bigcup_{e_i\in\mathcal{V}^{(rm)}} \mathcal{N}_{k+1}(e_i)$;*
- *For Edge Unlearning request $\Delta\mathcal{G} = \{\varnothing, \mathcal{E}^{(rm)}, \varnothing\}$, the influenced region is $\mathcal{N}_k(\mathcal{E}^{(rm)}) = \bigcup_{z_i\in\mathcal{E}^{(rm)}} \mathcal{N}_k(z_i)$;*
- *For Feature Unlearning request $\Delta\mathcal{G} = \{\varnothing, \varnothing, \mathcal{X}^{(rm)}\}$, the influenced region is $\mathcal{N}_k(\mathcal{X}^{(rm)}) = \bigcup_{z_i\sim\mathcal{X}^{(rm)}} \mathcal{N}_k(z_i)$, where $z_i \sim \mathcal{X}^{(rm)}$ indicates that the feature of node z_i is revoked.*

Combining the above Lemma, the formula can be further simplified to facilitate the estimation.

Corollary 4. *Denote the prediction of z_i in the remaining graph $\mathcal{G}\setminus\Delta\mathcal{G}$ using $f_{\mathcal{G}}$ as $f_{\mathcal{G}}(z_i; \mathcal{G}\setminus\Delta\mathcal{G})$ and the parameter change as $\Delta\theta = H^{-1}_{\theta_0}\nabla_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}$ when $\epsilon = -1$. Then the estimated parameter change by GIF for different unlearning tasks are as follows:*

- *For Node Unlearning tasks,*

$$\Delta\theta = H^{-1}_{\theta_0} \sum_{z_i\in\mathcal{N}_k(\mathcal{V}^{(rm)})\cup\mathcal{V}^{(rm)}} \nabla_{\theta_0} l\left(f_{\mathcal{G}}(z_i), y_i\right) - H^{-1}_{\theta_0} \sum_{z_i\in\mathcal{N}_k(\mathcal{V}^{(rm)})} \nabla_{\theta_0} l\left(f_{\mathcal{G}}(z_i; \mathcal{G}\setminus\Delta\mathcal{G}), y_i\right). \tag{20}$$

- *For Edge Unlearning tasks,*

$$\Delta\theta = H^{-1}_{\theta_0} \sum_{z_i\in\mathcal{N}_k(\mathcal{E}^{(rm)})} \nabla_{\theta_0} l\left(f_{\mathcal{G}}(z_i), y_i\right) - H^{-1}_{\theta_0} \sum_{z_i\in\mathcal{N}_k(\mathcal{E}^{(rm)})} \nabla_{\theta_0} l\left(f_{\mathcal{G}}(z_i; \mathcal{G}\setminus\Delta\mathcal{G}), y_i\right). \tag{21}$$

- *For Feature Unlearning tasks,*

$$\Delta\theta = H^{-1}_{\theta_0} \sum_{z_i\sim N_k(\mathcal{X}^{(rm)})\cup\mathcal{X}^{(rm)}} \nabla_{\theta_0} l\left(f_{\mathcal{G}}(z_i), y_i\right) - H^{-1}_{\theta_0} \sum_{z_i\sim N_k(\mathcal{X}^{(rm)})\cup\mathcal{X}^{(rm)}} \nabla_{\theta_0} l\left(f_{\mathcal{G}}(z_i; \mathcal{G}\setminus\Delta\mathcal{G}), y_i\right). \tag{22}$$

3.2 Efficient Estimation

Directly calculating the inverse matrix of Hessian matrix then computing Equation (17) requires the complexity of $O(|\theta|^3 + n|\theta|^2)$ and the memory of $O(|\theta|^2)$, which is prohibitively expensive in practice. Following the stochastic estimation method [10], we can reduce the complexity and memory to $O(n|\theta|)$ and $O(|\theta|)$, respectively.

Theorem 5. *Suppose Hessian matrix H is positive definite. When the spectral radius of $(I - H)$ is less than 1, $H_0^{-1} = I$ and $H_n^{-1} = (I - H)H_{n-1}^{-1} + I$ for $n = 1, 2, 3, \cdots$, then $\lim_{n\to\infty} H_n^{-1} = H^{-1}$.*

Since $\nabla_{\theta_0}\mathcal{L}_0$ is frequently computed when estimating the parameter change in Equation (20) - (22), which is however unchanged throughout the whole iteration process, we can storage it at the end of the model training once for all. Denote $H_t^{-1} = \sum_{i=0}^{t}(I - H)^i$, which represents the first t terms in the Taylor expansion of H^{-1}. Thus we can obtain the recursive equation $H_t^{-1} = I + (I - H)H_{t-1}^{-1}$. Let $v = \nabla_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}$, $H_t^{-1}v$ be the estimation of $H^{-1}_{\theta_0}\nabla_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}$. By iterating the equation until convergence, we have,

$$[H_t^{-1}v] = v + [H_{t-1}^{-1}v] - H_{\theta_0}[H_{t-1}^{-1}v], \quad [H_0^{-1}v] = v. \tag{23}$$

According to Theorem 5, Equation (23) will naturally iterate to converge when the spectral radius of $(I - H)$ is less than one. We take $H_t^{-1}\nabla_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}$ as the estimation of $H^{-1}_{\theta_0}\nabla_{\theta_0}\mathcal{L}(Z, \theta)$.

For more general Hessian matrixes, we add an extra scaling coefficient λ to guarantee the convergence condition. Specifically, let $\lambda H_t^{-1}\nabla_{\theta_0}\Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})}$ be the estimation based on the recursive results, then Equation (23) can be modified into:

$$[H_j^{-1}v] = v + [H_{j-1}^{-1}v] - \lambda H_{\theta_0}[H_{j-1}^{-1}v], \quad [H_0^{-1}v] = v. \tag{24}$$

We also discuss the selection of hyperparameters λ in Appendix D.

Proposition 6. *The time complexity of each iteration defined in Equation* (24) *is* $O(|\theta|)$.

Relying on the fast Hessian-vector products (HVPs) approach [27], $H_{\theta_0}^{-1}[H_{j-1}^{-1}v]$ can be exactly computed in only $O(|\theta|)$ time and $O(|\theta|)$ memory. All computations can be easily implemented by Autograd Engine [2]. According to our practical attempts, the number of iterations $t \ll n$ in practice. The computational complexity can be reduced to $O(n|\theta|)$ with $O(|\theta|)$ in memory.

3.3 Understanding the Mechanism of Unlearning

To better understand the underlying mechanism of graph unlearning, we deduce the closed-form solution for one-layer graph convolution networks under a node unlearning request. Denote P and P' as the GCN model's prediction trained on the original graph and the remaining graph. And the optimal model parameters are obtained by minimizing the following objective:

$$P = \sigma(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}XW), \quad W_0 = \arg\min_W \mathcal{L}_0. \tag{25}$$

We choose Softmax as the activation function σ and Cross entropy as measurement. For the original graph, we denote $H = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X$. h_{z_i} is the corresponding column of H for training data z_i. $p_{z_i,j}$ is the predicted probability of node z_i on the j-th label. $\hat{p}_{z_i,j} = 1 - p_{z_i,j}$ is the error probability when the j-th label is the accurate label for node z_i otherwise equals $p_{z_i,j}$. Similarly, we can define H', h'_{z_i}, $p'_{z_i,j}$, and $\hat{p}'_{z_i,j}$ for the remaining graph.

We hope to give a closed-form solution of $\hat{W} - W_0$,

$$P' = \sigma(D'^{-\frac{1}{2}}A'D'^{-\frac{1}{2}}X'W), \quad \hat{W} = \arg\min_W \mathcal{L}. \tag{26}$$

Theorem 7. *For one-layer GCN, the closed-form solution of* $\hat{W} - W_0 = (w_1, w_2, .., w_c)$, $\quad w_i = D_i^{-1}(E_i^{(rm)} + E_i^{(nei)}) \in R^{F\times 1}$, *where*

$$D_j = \sum_{z_i \in \mathcal{G}} p_{z_i,j}(1 - p_{z_i,j}) h_{z_i}^T h_{z_i},$$

$$E_j^{(rm)} = \sum_{z_i \in \mathcal{V}^{rm}} \hat{p}_{z_i,j} h_{z_i}^T, \quad E_j^{(nei)} = \sum_{z_i \in \mathcal{N}_K(\mathcal{V}^{(rm)})} (\hat{p}_{z_i,j} h_{z_i}^T - \hat{p}'_{z_i,j} h'^T_{z_i}).$$

Detailed derivations are included in Appendix C.

Generally, there are three key factors influencing the accuracy of parameter change estimation:

(1) D_j represents the resistance of overall training point.

(2) $E_j^{(rm)}$ represents the the influence caused by nodes from $\Delta\mathcal{G}$.

(3) $E_j^{(nei)}$ indicates the reflects the influence of affected neighbor nodes on parameter changes, which is jointly determined by model's accuracy and structural information in the original graph and remaining graph.

4 EXPERIMENTS

To justify the superiority of GIF for diverse graph unlearning tasks, we conduct extensive experiments and answer the following research questions:

[2] https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

Table 2: Statistics of the datasets.

Dataset	Type	#Nodes	#Edges	#Features	#Classes
Cora	Citation	2,708	5,429	1,433	7
Citeseer	Citation	3,327	4,732	3,703	6
CS	Coauthor	18,333	163,788	6,805	15

- **RQ1**: How does GIF perform in terms of model utility and running time as compared with the existing unlearning approaches?
- **RQ2**: Can GIF achieve removal guarantees of the unlearned data on the trained GNN model?
- **RQ3**: How does GIF perform under different unlearning settings?

4.1 Experimental Setup

Datasets. We conduct experiments on three public graph datasets with different sizes, including Cora [19], Citeseer [19], and CS [6]. These datasets are the benchmark dataset for evaluating the performance of GNN models for node classification task. Cora and Citeseer are citation datasets, where nodes represent the publications and edges indicate citation relationship between two publications, while the node features are bag-of-words representations that indicate the presence of keywords. CS is a coauthor dataset, where nodes are authors who are connected by an edge if they collaborate on a paper, the features represent keywords of the paper. For each dataset, we randomly split it into two subgraphs — a training subgraph that consists of 90% nodes for model training and a test subgraph containing the rest nodes for evaluation. The statistics of all three datasets are summarized in Table 2.

GNN Models. We compare GIF with four widely-used GNN models, including GCN [19], GAT [32], GIN [39], and SGC [35].

Evaluation Metrics. We evaluate the performance of GIF in terms of the following three criteria:

- `Unlearning Efficiency`: We record the running time to reflect the unlearning efficiency across different unlearning algorithms.
- `Model Utility`: As in GraphEraser [6], we use F1 score — the harmonic average of precision and recall — to measure the utility.
- `Unlearning Efficacy`: Due to the nonconvex nature of graph unlearning, it is intractable to measure the unlearning efficacy through the lens of model parameters. Instead, we propose an indirect evaluation strategy that measures model utility in the task of forgetting adversarial data. See section 4.3 for more details.

Baselines. We compare the proposed GIF with the following unlearning approaches:

- `Retrain`. This is the most straightforward solution that retrains GNN model from scratch using only the remaining data. It can achieve good model utility but falls short in unlearning efficiency.
- `GraphEraser` [6]. This is an efficient retraining method. It first divides the whole training data into multiple disjoint shards and then trains one sub-model for each shard. When the unlearning request arrives, it only needs to retrain the sub-model of shards that comprise the unlearned data. The final prediction is obtained by aggregating the predictions from all the sub-models. GraphEraser has two partition strategies for shard splitting, namely balanced LPA and balanced embedding k-means. We consider both of them as baselines.

Table 3: Comparison of F1 scores and running time (RT) for different graph unlearning methods for edge unlearning with 5% edges deleted from the original graph. 'LPA' and 'Kmeans' stand for GraphEraser [6] using balanced label propagation and balanced embedding k-means algorithm for community detection, respectively. The bold indicates the best result for each GNN model on each dataset.

Model		Dataset					
Backbone	Strategy	Cora		Citeseer		CS	
		F1 score	RT (second)	F1 score	RT (second)	F1 score	RT (second)
GCN	Retrain	0.8210±0.0055	6.33	0.7318±0.0096	7.52	0.9126±0.0055	55.95
	LPA	0.6790±0.0001	1.35	0.5556±0.0001	1.69	0.7732±0.0001	3.04
	Kmeans	0.5535±0.0001	1.89	0.5045±0.0001	1.56	0.7754±0.0001	9.97
	GIF	**0.8218±0.0066**	**0.16**	**0.6925±0.0060**	**0.13**	**0.9137±0.0016**	**0.26**
GAT	Retrain	0.8804±0.0060	15.72	0.7643±0.0049	19.50	0.9305±0.0011	110.39
	LPA	0.3432±0.0001	3.04	0.6997±0.0001	3.92	0.7650±0.0001	6.43
	Kmeans	0.6900±0.0001	3.19	0.7628±0.0001	3.41	0.8794±0.0001	17.36
	GIF	**0.8649±0.0072**	**0.86**	**0.7663±0.0072**	**0.59**	**0.9325±0.0015**	**1.02**
SGC	Retrain	0.8236±0.0142	6.63	0.7132±0.0091	7.16	0.9165±0.0045	58.12
	LPA	0.3247±0.0001	2.03	0.3934±0.0001	1.70	0.5267±0.0001	3.08
	Kmeans	0.3690±0.0001	1.41	0.3874±0.0001	1.56	0.6532±0.0001	10.59
	GIF	**0.8129±0.0110**	**0.12**	**0.6892±0.0082**	**0.12**	**0.9164±0.0051**	**0.24**
GIN	Retrain	0.8051±0.0144	8.48	0.7294±0.0207	9.94	0.8822±0.0074	70.38
	LPA	0.6605±0.0001	2.65	0.6096±0.0001	2.21	0.6510±0.0001	3.82
	Kmeans	0.7491±0.0001	2.53	0.6517±0.0001	2.11	0.8336±0.0001	10.29
	GIF	**0.8059±0.0234**	**0.48**	**0.7315±0.0185**	**0.48**	**0.8884±0.0083**	**0.57**

- `Influence Function` [20]. It can be regarded as a simplified version of our GIF, which only considers the individual sample, ignoring the interaction within samples.

It's worth mentioning that we do not include some recent methods like GraphEditor [9] (not published yet) and [14] since they are tailored for linear models.

Implementation Details. We conduct experiments on a single GPU server with Intel Xeon CPU and Nvidia RTX 3090 GPUs. In this paper, we focus on three types of graph unlearning tasks. For node unlearning tasks, we randomly delete the nodes in the training graph with an unlearning ratio ρ, together with the connected edges. For edge unlearning tasks, each edge on the training graph can be revoked with a probability ρ. For feature unlearning tasks, the feature of each node is dropped with a probability ρ, while keeping the topology structure unchanged. All the GNN models are implemented with the PyTorch Geometric library. Following the settings of GraphEraser [6], for each GNN model, we fix the number of layers to 2, and train each model for 100 epochs. For GraphEraser, as suggested by the paper [6], we partition the three datasets, Cora, Citeseer, and Physics, with two proposed methods, BLPA and BKEM, into 20, 20, and 50 shards respectively. For the unique hyperparameters of GIF, we set the number of iterations to 100, while tuning the scaling coefficient λ in the range of $\{10^1, 10^2, 10^3, 10^4, \cdots\}$. All the experiments are run 10 times. We report the average value and standard deviation.

4.2 Evaluation of Utility and Efficiency (RQ1)

To answer **RQ1**, we implement different graph unlearning methods on four representative GNN models and test them on three datasets. Table 3 shows the experimental result for edge unlearning with ratio $\rho = 0.05$. We have the following **Obs**ervations:

- **Obs1: LPA and Kmeans have better unlearning efficiency than Retrain while failing to guarantee the model utility.** Firstly, we can clearly observe that Retrain can achieve excellent performance in terms of the model utility. However, it suffers from worse efficiency with a large running time. Specifically, on four GNN models, Retrain on the CS dataset is approximately 5 to 20 times the running time of other baseline methods. Meanwhile, other baseline methods also cannot guarantee the performance of the model utility. For example, when applying LPA to different GNN models on Cora dataset, the performance drops by approximately 17.9%∼60.5%, compared with Retrain. For Kmeans, it will drop by around 15.0%∼28.72% on the Citeseer dataset with four different GNN models. These results further illustrate that existing efforts cannot achieve a better trade-off between the unlearning efficiency and the model utility.
- **Obs2: GIF consistently outperforms all baselines and can achieve a better trade-off between unlearning efficiency and model utility.** For the performance of node classification tasks, GIF can completely outperform LPA and Kmeans, and even keep a large gap. For example, on the Cora dataset, GIF outperforms LPA by around 21.0%∼150.4% over 4 different GNN models, and outperforms Kmeans by 7.58%∼120.3%. In addition, GIF greatly closes the gap with Retrain. For instance, GIF achieves a comparable performance level with Retrain on all datasets over 4 different GNN models, and it even achieves better performance than Retrain with GCN models on Cora and CS datasets. These results demonstrate that GIF can effectively improve the performance of the model utility. In terms of unlearning efficiency, GIFs can effectively reduce running time. Compared with LPA and Kmeans, GIF can shorten the running time by about 15∼40 times. Moreover, compared with Retrain, GIF can even shorten the running time by hundreds of times, such as GAT on the CS dataset. These results fully demonstrate that GIF can guarantee

both performance and efficiency, and consistently outperform existing baseline methods. GIF effectively closes the performance gap with Retrain and achieves a better trade-off between unlearning efficiency and the model utility.

4.3 Evaluation of Unlearning Efficacy (RQ2)

The paramount goal of graph unlearning is to eliminate the influence of target data on the deployed model. However, it is insufficient to measure the degree of data removal simply based on the model utility. Given that IF-based methods, including GIF and traditional IF, generate the new model by estimating parameter changes in response to a small mass of perturbation, a tricky unlearning mechanism may deliberately suppress the amount of parameter change, thus achieving satisfactory model utility. To accurately assess the completeness of data removal, we propose a novel evaluation method inspired by adversarial attacks. Specifically, we first add a certain number of edges to the training graph, satisfying that each newly-added edge links two nodes from different classes. The adversarial noise will mislead the representation learning, thus reducing the performance. Thereafter, we use the adversarial edges as unlearning requests and evaluate the utility of the estimated model by GIF and IF. Intuitively, larger utility gain indicated higher unlearning efficacy. Figure 2 shows the F1 scores of the different models produced by GIF and IF under different attack ratios on Cora. We have the following **Obs**ervations:

- **Obs3: Both GIF and IF can reduce the negative impact of adversarial edges, while GIF consistently outperforms IF.** The green curves show the performance of directly training on the corrupted graphs. We can clearly find that as the attack ratio increases, the performance shows a significant downward trend. For example, on GCN model, the performance drops from 0.819 to 0.760 as the attack ratio increases from 0.5 to 0.9; on SGC model, the performance drops from 0.817 to 0.785 as the attack ratio increases from 0.7 to 0.9. These results illustrate that adversarial edges can greatly degrade performance. The results of IF are shown as blue curves. We can observe that the trend of performance degradation can be significantly alleviated. For example, on GCN model, compared with training on the corrupted graphs, when the attack ratio is increased from 0.5 to 0.9, the performance is relatively increased by 5%∼10%. On SGC, there are also relative improvements of 5%∼10% on performance when the attack ratio is increased from 0.7 to 0.9. These results demonstrate that IF can effectively improve the unlearning efficacy. Finally, we plot the experimental results of GIF with the red curves. We observe that GIF consistently outperforms other methods and maintains a large gap. Compared with IF, the performance is improved by about 2.1%∼4.7% on the GCN model and is improved by about 1%∼5% on SGC model. In addition, except for the GIN model, the performance of GIF even improves as the ratio goes up. These results further demonstrate that GIF can effectively improve the unlearning efficacy by considering the information of graph structure.

4.4 Hyperparameter Studies (RQ3)

We then move on to studying the impact of hyperparameters. We implement two unlearning tasks with different unlearning ratios ρ on two representative GNN models, GCN and SGC, to compare the performance of the GIF with the retrained model. We also study the impact of scaling coefficient λ in Appendix D. Figure 3 shows the experimental result of the node unlearning and feature unlearning with ratio ρ range from 0.1 to 0.5 on Cora and Citeseer datasets. We have the following **Obs**ervations:

- **Obs4: GIF consistently achieves comparable performance to Retrain on both node and feature unlearning tasks.** The black dashed lines represent the performance without unlearning, which is an upper bound on performance. From the results, we can clearly observe that GIF can achieve comparable performance to Retrain on both node unlearning and feature unlearning tasks, with different models or datasets. Specifically, for the node unlearning tasks, the performance of both methods gradually decreases as the ratio increases, in contrast to feature unlearning tasks. Nonetheless, GIF can achieve a similar performance compared to Retrain, and even surpass Retrain on the Cora dataset. These results illustrate that GIF can efficiently handle node-level tasks and can replace Retrain-based methods. For feature unlearning tasks, we can observe that the performance drops less compared to node unlearning tasks. Since node unlearning tasks need to remove the local structure of the entire graph, including node features and neighbor relationships. While feature unlearning tasks just need to remove some node features, which will lead to less impact on performance. From the results, we can find that GIF can also achieve comparable performance to retraining on different datasets or GNN models. These experimental results and analyses further demonstrate that GIF can effectively handle both node-level and feature-level tasks, and can replace retraining-based methods to achieve outstanding performance.

5 RELATED WORK

Machine unlearning aims to eliminate the influence of a subset of the training data from the trained model out of privacy protection and model security. Ever since Cao & Yang [4] first introduced the concept, several methods are proposed to address the unlearning tasks, which can be classified into two branches: exact approaches and approximate approaches.

Exact approaches. Exact unlearning methods aim to create models that perform identically to the model trained without the deleted data, or in other words, retraining from scratch, which is the most straightforward way but computationally demanding. Many prior efforts designed models, say Ginart *et al.* [12] described unlearning approaches for k-means clustering while Karasuyama *et al.* [18] for support vector machines. Among these, the *SISA*(sharded, isolated, sliced, and aggregated) approach [3] partitions the data and separately trains a set of constituent models, which are afterwards aggregated to form a whole model. During the procedure of unlearning, only the affected submodel is retrained smaller fragments of data, thus greatly enhance the unlearning efficiency. Follow-up work GraphEraser [6] extends the shards-based idea to graph-structured data, which offers partition methods to preserve the structural information and also designs a weighted aggregation for inference. But inevitably, GraphEraser could impair the performance of the unlearned model to some extent, which are shown in our paper.

Approximate approaches. Approximate machine unlearning methods aim to facilitate the efficiency through degrading the performance requirements, in other words, to achieve excellent trade-offs

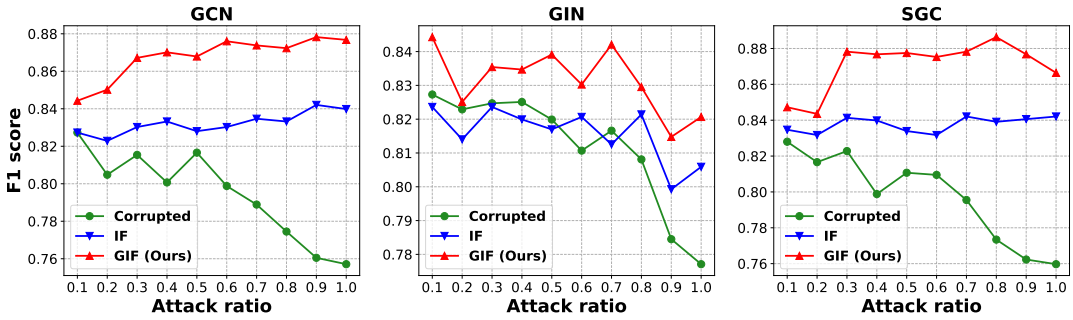

Figure 2: Comparison of unlearning efficacy over three GNN models.

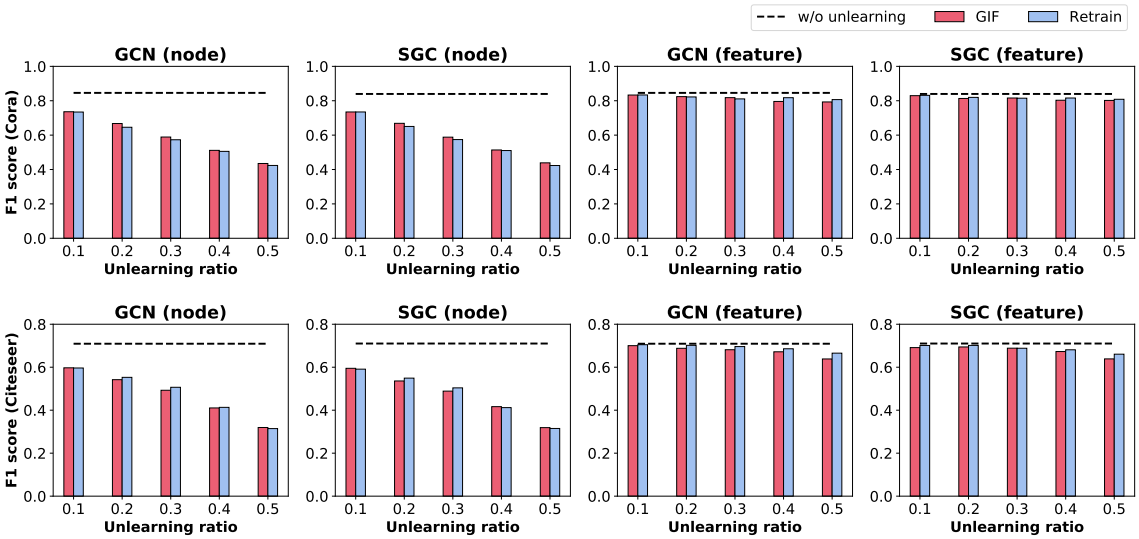

Figure 3: Impact of unlearning ratio ρ on node unlearning tasks and feature unlearning tasks.

between the efficiency and effectiveness. Recently, influence function [20] is proposed to measure the impact of a training point on the trained model. Adapting the influence function in the unlearning tasks, Guo *et al.* [14] introduced a probabilistic definition of unlearning motivated by differential privacy [11], and proposed to unlearn by removing the influence of the deleted data on the model parameters. Specifically, they used the deleted data to update ML models by performing a Newton step to approximate the influence of the deleted data and remove it, then they introduced a random noise to the training objective function to ensure the certifiability. With a similar idea, Izzo *et al.* [17] propose an approximate data deletion method with a computation time independent of the size of the dataset. Another concurrent work [7] proposes a similar formula for edge and node unlearning tasks on simple graph convolution model, and further analyzes the theoretical error bound of the estimated influences under the Lipschitz continuous condition.

6 CONCLUSION AND FUTURE WORK

In this work, we study the problem of graph unlearning that removes the influence of target data from the trained GNNs. We first unify different types of graph unlearning tasks *w.r.t.* node, edge, and feature into a general formulation. Then we recognize the limitations of existing influence function when solving graph data and explore the remedies. In particular, we design a graph-oriented influence function that considers the structural influence of deleted nodes/edges/features on their neighbors. Further deductions on the closed-form solution provide a better understanding of the unlearning mechanism. We conduct extensive experiments on four GNN models and three benchmark datasets to justify the advantages of GIF for diverse graph unlearning tasks in terms of unlearning efficacy, model utility, and unlearning efficiency.

At present, the research on graph unlearning is still at an early stage and there are several open research questions. In future work, we would like to explore the potential of graph influence function in other applications, such as recommender systems [37, 42], influential subgraph discovery, and certified data removal. Going beyond explicit unlearning requests, we will focus on the auto-repair ability of the deployed model — that is, to discover the adversarial attacks to the model and revoke their negative impact. Another promising direction is the explainability of unlearning algorithms that can improve the experience in human-AI interaction.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (2020AAA0106000), the National Natural Science Foundation of China (9227010114, U21B2026), and the CCCD Key Lab of Ministry of Culture and Tourism.

REFERENCES

[1] Nasser Aldaghri, Hessam Mahdavifar, and Ahmad Beirami. 2021. Coded Machine Unlearning. *IEEE Access* 9 (2021), 88137–88150.
[2] Samyadeep Basu, Phillip Pope, and Soheil Feizi. 2021. Influence Functions in Deep Learning Are Fragile. In *ICLR*.
[3] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine Unlearning. In *IEEE Symposium on Security and Privacy*. 141–159.
[4] Yinzhi Cao and Junfeng Yang. 2015. Towards Making Systems Forget with Machine Unlearning. In *IEEE Symposium on Security and Privacy*. 463–480.
[5] Chong Chen, Fei Sun, Min Zhang, and Bolin Ding. 2022. Recommendation Unlearning. In *WWW*. 2768–2777.
[6] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2022. Graph Unlearning. In *SIGSAC*.
[7] Zizhang Chen, Peizhao Li, Hongfu Liu, and Pengyu Hong. 2022. Characterizing the Influence of Graph Elements. *CoRR* abs/2210.07441 (2022).
[8] Eli Chien, Chao Pan, and Olgica Milenkovic. 2022. Certified Graph Unlearning. *CoRR* abs/2206.09140 (2022).
[9] Weilin Cong and Mehrdad Mahdavi. 2022. GRAPHEDITOR: An Efficient Graph Representation Learning and Unlearning Approach. (2022).
[10] R Dennis Cook and Sanford Weisberg. 1980. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics* 22 (1980), 495–508.
[11] Cynthia Dwork. 2011. Differential Privacy. In *Encyclopedia of Cryptography and Security, 2nd Ed.* 338–340.
[12] Antonio Ginart, Melody Y. Guan, Gregory Valiant, and James Zou. 2019. Making AI Forget You: Data Deletion in Machine Learning. In *NeurIPS*. 3513–3526.
[13] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In *CVPR*. 9301–9309.
[14] Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. 2020. Certified Data Removal from Machine Learning Models. In *ICML*, Vol. 119. 3832–3842.
[15] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.
[16] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.
[17] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. 2021. Approximate Data Deletion from Machine Learning Models. In *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*. 2008–2016.
[18] Masayuki Karasuyama and Ichiro Takeuchi. 2010. Multiple incremental decremental learning of support vector machines. *IEEE Trans. Neural Networks* 21 (2010), 1048–1059.
[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR (Poster)*.
[20] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *ICML*, Vol. 70. 1885–1894.
[21] Chanhee Kwak, Junyeong Lee, Kyuhong Park, and Heeseok Lee. 2017. Let Machines Unlearn - Machine Unlearning and the Right to be Forgotten. In *AMCIS*. Association for Information Systems.
[22] Sihang Li, Xiang Wang, An Zhang, Yingxin Wu, Xiangnan He, and Tat-Seng Chua. 2022. Let Invariant Rationale Discovery Inspire Graph Contrastive Learning. In *ICML*, Vol. 162. 13052–13065.
[23] Neil G. Marchant, Benjamin I. P. Rubinstein, and Scott Alfeld. 2021. Hard to Forget: Poisoning Attacks on Certified Machine Unlearning. *CoRR* abs/2109.08266 (2021).
[24] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. 2020. Descent-to-Delete: Gradient-Based Methods for Machine Unlearning. *CoRR* abs/2007.02923 (2020).
[25] Tongyao Pang, Huan Zheng, Yuhui Quan, and Hui Ji. 2021. Recorrupted-to-Recorrupted: Unsupervised Deep Learning for Image Denoising. In *CVPR*. Computer Vision Foundation / IEEE, 2043–2052.
[26] Stuart L Pardau. 2018. The California consumer privacy act: Towards a European-style privacy regime in the United States. *J. Tech. L. & Pol'y* 23 (2018), 68.
[27] Barak A. Pearlmutter. 1994. Fast Exact Multiplication by the Hessian. *Neural Comput.* 6, 1 (1994), 147–160.
[28] Protection Regulation. 2018. General data protection regulation. *Intouch* 25 (2018).
[29] Benjamin I. P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. 2009. ANTIDOTE: understanding and defending against poisoning of anomaly detectors. In *Internet Measurement Conference*. ACM, 1–14.
[30] Enayat Ullah, Tung Mai, Anup Rao, Ryan A. Rossi, and Raman Arora. 2021. Machine Unlearning via Algorithmic Stability. *CoRR* abs/2102.13179 (2021).
[31] Rianne van den Berg, Thomas N. Kipf, and Max Welling. 2017. Graph Convolutional Matrix Completion. *CoRR* abs/1706.02263 (2017).
[32] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. *CoRR* abs/1710.10903 (2017).
[33] Wenjie Wang, Xinyu Lin, Fuli Feng, Xiangnan He, Min Lin, and Tat-Seng Chua. 2022. Causal Representation Learning for Out-of-Distribution Recommendation. In *WWW*. ACM, 3562–3571.
[34] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*. ACM, 165–174.
[35] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.
[36] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised Graph Learning for Recommendation. In *SIGIR*. ACM, 726–735.
[37] Jiancan Wu, Xiang Wang, Xingyu Gao, Jiawei Chen, Hongcheng Fu, Tianyu Qiu, and Xiangnan He. 2022. On the Effectiveness of Sampled Softmax Loss for Item Recommendation. *CoRR* abs/2201.02327 (2022).
[38] Yingxin Wu, Xiang Wang, An Zhang, Xiangnan He, and Tat-Seng Chua. 2022. Discovering Invariant Rationales for Graph Neural Networks. In *ICLR*.
[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*. OpenReview.net.
[40] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*, Vol. 80. 5449–5458.
[41] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*. ACM, 974–983.
[42] An Zhang, Wenchang Ma, Xiang Wang, and Tat-Seng Chua. 2022. Incorporating Bias-aware Margins into Contrastive Loss for Collaborative Filtering. In *NeurIPS*.
[43] Jie Zhang, Dongdong Chen, Qidong Huang, Jing Liao, Weiming Zhang, Huamin Feng, Gang Hua, and Nenghai Yu. 2022. Poison Ink: Robust and Invisible Backdoor Attack. *IEEE Trans. Image Process.* 31 (2022), 5691–5705.

A NON-CONVEX AND NON-DIFFERENTIAL DISCUSSION

Prior studies in other domains show that influence functions in non-convex shallow networks can achieve excellent results, such as CNN[20]. However, when the number of layers goes too deep, the prediction will become inaccurate since the convexity of the objective function will be broken heavily (*i.e.,* the curvature value of the network at optimal model parameters might be quite large[2]). [20] proposes a convex quadratic approximation of the loss around the optimal parameter θ and smooth approximations to non-differentiable losses. Concurrent work[7] provides some error analysis based on simple graph convolutions. Thus, strict and general error analysis related to influence functions in deep networks[40] is still an unexplored territory.

B PROOF OF THEOREM 5

Consider the n-th term of H_n^{-1},

$$H_n^{-1} = (I-H)H_{n-1}^{-1} + I \tag{27}$$

$$\Rightarrow H_n^{-1} - H^{-1} = (I-H)(H_{n-1}^{-1} - H^{-1}) \tag{28}$$

$$\Rightarrow H_n^{-1} = (I-H)^n + H^{-1}. \tag{29}$$

here $(I-H)$ is a square $|\theta|\times|\theta|$ matrix with $|\theta|$ linearly independent eigenvectors $V_i \in \mathbb{R}^{|\theta|\times 1}$ and the corresponding eigenvalue λ_i (where $i = 1, ..., |\theta|$). Then it can be factorized as

$$(I-H) = TUT^{-1}, \tag{30}$$

where T is a square $|\theta| \times |\theta|$ matrix whose i-th column is the eigenvector v_i, and U is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, $U_{ii} = \lambda_i$

$$(I-H)^n = TU^nT^{-1}, \tag{31}$$

T and T^{-1} are fixed value matrix. Since $\forall|\lambda_i| < 1$, denote $|\lambda| = \max_{1\leqslant i\leqslant|\theta|} |\lambda_i|$.

$$\forall 1 \leqslant s,t \leqslant |\theta|, |u_{st}|^n \leqslant |\lambda|^n, \quad \lim_{n\to\infty} |\lambda|^n = 0, \tag{32}$$

$$\text{Thus,} \quad \lim_{n\to\infty} U^n = 0 \Rightarrow \lim_{n\to\infty} (I-H)^n = \lim_{n\to\infty} TU^nT^{-1} = 0. \tag{33}$$

$$\tag{34}$$

And therefore, we prove that $\lim_{n\to\infty} H_n^{-1} = H^{-1}$.

C DERIVATION OF ONE-LAYER GCN MODEL

We first calculate the first-order gradient of the arbitrary element in matrix P_{log} in Corollary 8, then we derive its Hessian matrix in Corollary 10, finally we substitute the results into Equation (20) and finish the proof.

Notations: The number of nodes is $M = |\mathcal{V}|$ with C classes. For $\odot, A, B \in \mathbb{R}^{s\times t}, A\odot B = (a_{ij}b_{ij})_{s\times t} \in \mathbb{R}^{s\times t}$. $\otimes$ is the Kronecker product. Denote $Vec(A)$ as the vectorization of matrix A, which is a linear transformation which converts the matrix into a column vector.$P = (p_{mn})_{M\times C}, P_{log} = (l_{mn})_{M\times C}, T = (t_{mn})_{M\times C}$,. P_{log} is equivalent to take the logarithm of each element of the matrix P. $I_{c\times c} \in \mathbb{R}^{C\times C}$ is an all-one matrix. $I_{M\times N}^{(M_i,N_i)} = (i_{ij})_{M\times N} \in \mathbb{R}^{M\times N}$, $i_{M_i,N_i} = 1$ and the rest are all zero. $I_c \in \mathbb{R}^{C\times C}$ is an identity matrix. $H = [h1, h2, .., h_M]^T, \forall h_j \in \mathbb{R}^{1\times F}$, $W = (w_{ij})_{F\times C} \in \mathbb{R}^{F\times C}$, $\forall 1 \leqslant M_i \leqslant M, 1 \leqslant C_i \leqslant C$ We define $\nabla_W l_{M_i,C_i} \in \mathbb{R}^{F\times C}$, whose s-th and t-th column element is $(\nabla_W l_{M_i,C_i})_{st} = \frac{\partial l_{M_i,C_i}}{\partial w_{st}}$.

We now deduce the first order gradient.

Corollary 8. *For any element l_{M_i,C_i} of matrix P_{log}, we have* $\nabla_W l_{M_i,C_i} = (H)^T[I_{M\times C}^{(M_i,C_i)} - I_{M\times M}^{(M_i,M_i)} Softmax(T)]$

Proof:

$$P_{log} = LogSoftmax(T), \quad \Rightarrow dP_{log} = dLogSoftmax(T), \tag{35}$$

$$\Rightarrow l_{mn} = log(\frac{e^{t_{mn}}}{\sum_{i=1}^{C} e^{t_{mi}}}) \quad \Rightarrow dl_{mn} = dt_{mn} - \frac{\sum_{i=1}^{C}(e^{t_{mi}} dt_{mi})}{\sum_{i=1}^{C} e^{t_{mi}}}. \tag{36}$$

Substitute the equation above back into (35)

$$dP_{log} = dT - (Softmax(T)\odot dT) I_{C\times C}. \tag{37}$$

For training point z_i, denote the prediction of the corresponding label in P is p_{M_i,C_i}. With expectation to decompose the left side to the following form

$$dl_{M_i,C_i} = \nabla_W l_{M_i,C_i} \odot dW. \tag{38}$$

We first decompose it into

$$dl_{M_i,C_i} = \nabla_T l_{M_i,C_i} \odot dT. \tag{39}$$

Calculating the corresponding term in equation (37),

$$\nabla_T l_{M_i,C_i} = I_{M\times C}^{(M_i,C_i)} - I_{M\times M}^{(M_i,M_i)} Softmax(T) \tag{40}$$

$$\text{Since} \quad T = HW \quad \Rightarrow \nabla_W l_{M_i,C_i} = H^T \nabla_T l_{M_i,C_i} \tag{41}$$

$$\Rightarrow \nabla_W l_{M_i,C_j} = (H)^T[I_{M\times C}^{(M_i,C_i)} - I_{M\times M}^{(M_i,M_i)} Softmax(T)] \tag{42}$$

$$\nabla_{Vec(W)} l_{M_i,C_i} = Vec((H)^T\{I_{M\times C}^{(M_i,C_i)} - I_{M\times M}^{(M_i,M_i)} Softmax(T)\}) \tag{43}$$

In what follows, we derive the Hessian matrix. For ease of notation, we denote $\nabla_{Vec(W)} l_{M_i,C_i} = Vec(\nabla_W l_{M_i,C_i})$, the i-th element of $Vec(W)$ is $w_i^{(v)}$.

Lemma 9. *For $\forall T \in \mathbb{R}^{M\times C}$, we have*

$$\begin{aligned} dSoftmax(T) =& Softmax(T)\odot dT \\ & - Softmax(T)\odot[(Softmax(T)\odot dT) I_{C\times C}] \end{aligned} \tag{44}$$

Corollary 10. *For $\forall 1 \leqslant M_i \leqslant M, 1 \leqslant C_i \leqslant C$, we define $\nabla_W^2 l_{M_i,C_i} \in \mathbb{R}^{FC\times FC}$, whose s-row t-column element is $(\nabla_W^2 l_{M_i,C_i})_{st} = \frac{\partial^2 l_{M_i,C_i}}{\partial w_s^{(v)} \partial w_t^{(v)}}$. Then, $\nabla_W^2 l_{M_i,C_i}$ can be block into the following form:*

$$\nabla_W^2 l_{M_i,C_i} = diag\{D_1^{(i)}, D_2^{(i)}, ..., D_c^{(i)}\}, \quad D_j^{(i)} = p_{M_i,j}(1-p_{M_i,j}) h_{M_i}^T h_{M_i} \tag{45}$$

Proof:

$$d\nabla_{vec(W)} l_{M_i,C_i} = -dVec((\widehat{H})^T (I_{M\times M}^{(M_i,M_i)} Softmax(T)) \quad (46)$$

$$\Rightarrow d\nabla_{vec(W)} l_{M_i,C_i} = -(I_C \otimes (\widehat{H})^T) Vec(d(I_{M\times M}^{(M_i,M_i)} Softmax(T)) \quad (47)$$

$$\Rightarrow d\nabla_{vec(w)} l_{M_i,C_i} = -(I_C \otimes (\widehat{H})^T)(I_C \otimes I_{M\times M}^{(M_i,M_i)}) Vec(dSoftmax(T)) \quad (48)$$

Utilizing Lemma 9, we have

$$\begin{aligned} Vec(dSoftmax(T)) =& diag(Softmax(T)) \\ & \cdot [I - (I_{C\times C} \odot I_M) diag(SoftmaxT)] \\ & \cdot (I_{C\times C} \odot H) Vec(dW) \end{aligned} \quad (49)$$

$$\begin{aligned} \Rightarrow d\nabla_{vec(W)} l_{M_i,C_i} = -(I_C \otimes (\widehat{H})^T)(I_C \otimes I_{M\times M}^{(M_i,M_i)}) \cdot \\ diag(Softmax(T)\{I - (I_{C\times C} \odot I) diag(SoftmaxA)\}(I_C \odot H) Vec(dW) \end{aligned} \quad (50)$$

$$\begin{aligned} \Rightarrow \nabla_W^2 l_{M_i,M_i} = -(I_C \otimes (\widehat{H})^T)(I_C \otimes I_{M\times M}^{(M_i,M_i)}) \\ diag(Softmax(T)\{I_{M\times C} - diag(Softmax(T)\}(I_C \otimes H) \end{aligned} \quad (51)$$

Then we multiply the corresponding matrices together and simplify the result.

$$\Rightarrow \nabla_W^2 l_{M_i,C_i} = diag\{D_1^{(i)}, D_2^{(i)}, ..., D_C^{(i)}\} \cdot \quad (52)$$

$$D_j^{(i)} = H^T I_{M\times M}^{(M_i,M_i)} \tilde{P}_j (1 - \tilde{P}_j) H = p_{M_i,j}(1 - p_{M_i,j}) h_{M_i}^T h_{M_i} \quad (53)$$

We then have

$$H_{\theta_0} = diag\{D_1, D_2, ..., D_C\}, \quad D_j = \sum_{z_i \in \mathcal{G}} p_{M_i,j}(1 - p_{M_i,j}) h_{M_i}^T h_{M_i} \quad (54)$$

$$\begin{aligned} \nabla_{\theta_0} \Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})} =& Vec(\sum_{z_i \in \mathcal{V}^{rm}} \nabla_{W_0} l_{M_i,C_i} \\ & + \sum_{z_i \in N_K(\mathcal{V}^{rm})} (\nabla_{W_0} l_{M_i,C_i} - \nabla_{W_0} \hat{l}_{M_i,C_i}) \end{aligned} \quad (55)$$

$$H^T P_{z_i} = (\hat{P}_{M_i,C_1} h_{M_i}^T, \hat{P}_{M_i,C_2} h_{M_i}^T, .., \hat{P}_{M_i,c_c} h_{M_i}^T) \in \mathbb{R}^{F\times C} \quad (56)$$

$$\Rightarrow H_{\theta_0} \nabla_{\theta_0} \Delta\mathcal{L}_{(\mathcal{G}\setminus\Delta\mathcal{G})} = diag(D_1^{-1} E_1, D_2^{-1} E_2, ..., D_C^{-1} E_C) \quad (57)$$

$$\Rightarrow \hat{W} - W_0 = (w_1, w_2, .., w_C), \quad w_i = D_i^{-1} E_i \quad (58)$$

$$D_j = \sum_{z_i \in \mathcal{G}} p_{M_i,j}(1 - p_{M_i,j}) h_{M_i}^T h_{M_i}, \qquad E_j = E_j^{(rm)} + E_j^{(nei)}$$

$$E_j^{(rm)} = \sum_{z_i \in \mathcal{V}^{rm}} \hat{p}_{M_i,C_j} h_{M_i}^T, \quad (59)$$

$$E_j^{(nei)} = \sum_{z_i \in N_K(V^{rm}} (p'_{M_i,C_j} h_{M_i}^T - \hat{p}'_{M_i,C_j} \hat{h}_{M_i}^T) \quad (60)$$

D HYPERPARAMETER ANALYSIS

Since the scaling coefficient λ controls the convergence condition of the efficient estimation algorithm, we here investigate how the choice of λ influences the utility of the unlearned model in the edge unlearning tasks. We omit other tasks as they exhibit a similar trend. We fix the unlearning ratio to 0.05 and the iteration number to 100. We compare the F1 score of the retrain version and the unlearned version of GCN and GAT on Cora and Citeseer datasets. As the results shown in Figure 4, we have the following **Obs**ervations:

- **Obs5: The scaling coefficient exerts large impacts on the performance of GIF.** Specifically, when λ is small, the performance of the unlearned model is poor, which suggests that the convergence condition is not achieved. However, when λ exceeds a threshold, the performance of the unlearned model rises rapidly as λ increases. Thereafter, the performance reaches saturation, that is, increasing λ will not bring performance gain. The saturation performance is close to "Retrain", verifying the superiority of GIF. In addition, the saturation point of λ varies across GNN models and datasets. For example, when implemented on GAT, the peak performance is achieved when *lambda* is larger than 16000 and 20000 on Cora and Citeseer, respectively; While on GCN, $\lambda = 1000$ is sufficient for both Cora and Citeseer. In general, we suggest picking a large λ for convergence.

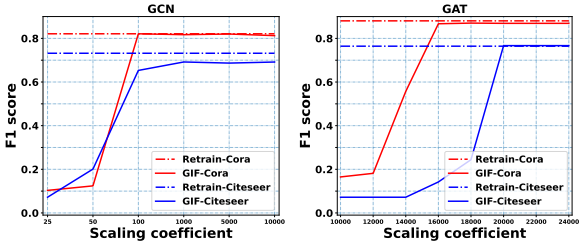

Figure 4: Comparsion of F1 scores over different scaling coefficients in GCN and GAT models