Graph Unlearning for MLaaS: Toward Flexible Privacy Adjustment via Influenced Subgraph

Yang Li, Yi Zhao, *Member, IEEE*, Qi Tan, Yinggui Wang , Lei Wang , Tao Wei , Min Zhu, Ke Xu, *Fellow, IEEE*, and Youjian Zhao

***Abstract*—Graph unlearning focuses on removing specific information from graph neural networks (GNNs). Existing methods often assume full access to training data, which is impractical in machine learning-as-a-service (MLaaS) scenarios, where data owners, model developers, and service providers operate separately. Since service providers typically lack access to training data, graph unlearning can only be performed by model developers. In this paper, we introduce an innovative subgraph-based certified graph unlearning (SCGU) method for MLaaS scenarios, enabling service providers to directly modify model parameters to achieve graph unlearning. Specifically, we first define the influenced subgraph, which is only closely related to the unlearning target. Based on the message-passing mechanism of GNNs, we then localize the computation of the gradient difference to the influenced subgraph. Moreover, we introduce an indicator, L-CODEC, to identify the most important parameters for unlearning and compute only their Hessian matrix via finite differences, thereby localizing all parameter estimation within the influenced subgraph. Once an unlearning request is received, the service provider only requires the influenced subgraph to compute relevant variables and then directly modifies the model parameters to achieve graph unlearning. We evaluate the model utility, unlearning efficiency, and efficacy of our SCGU across multiple datasets and GNN architectures. Experimental results show that SCGU is at least 4 times faster than retraining, achieves a parameter approximation to the retrained model that is 3%- 5% closer, and outperforms all baselines except retraining in model utility and unlearning efficacy across most scenarios.**

***Index Terms*—Machine learning, security and protection, graph neural networks, graph unlearning, MLaaS.**

Received 10 December 2025; revised 26 May 2026; accepted 9 July 2026. Date of publication 20 July 2026; date of current version 1 September 2026. This work was supported in part by the National Science Foundation for Distinguished Young Scholars of China under Grant 62425201, in part by the National Natural Science Foundation of China under Grant 62472036 and Grant U22B2031, in part by the Beijing Nova Program, and in part by the Open Research Fund of State Key Laboratory of Internet Architecture under Grant HLW2025MS04. *(Corresponding authors: Yi Zhao; Ke Xu.)*

Yang Li, Min Zhu, Ke Xu, and Youjian Zhao are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: li-yang23@mails.tsinghua.edu.cn; minzhu@tsinghua.edu.cn; xuke@tsinghua.edu.cn; zhaoyoujian@tsinghua.edu.cn).

Yi Zhao is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China, and also with the State Key Laboratory of Internet Architecture, Tsinghua University, Beijing 100084, China (e-mail: zhaoyi@bit.edu.cn).

Qi Tan is with the College of Computer Science and Software Engineering, Shenzhen University, Guangdong 518060, China (e-mail: tanqi@szu.edu.cn).

Yinggui Wang, Lei Wang, and Tao Wei are Independent Researchers, Beijing 100081, China (e-mail: wyinggui@gmail.com; wyinggui@gmail.com; wyinggui@gmail.com).

Digital Object Identifier 10.1109/TDSC.2026.3715465

I. Introduction

GRAPH neural networks (GNNs) have been widely used across various graph-structured data domains, including recommendation systems [1], financial systems [2], [3], and bioinformatics [4], [5]. The rise of machine learning as a service (MLaaS) has transformed GNN deployment into a tripartite ecosystem: data owners provide training data, model developers build GNN models, and service providers manage deployment. A successful example is the cooperation between Careem[1] and *Amazon Web Services (AWS)*. Specifically, *Careem* (i.e., data owner) provided the identity graph to the *Neptune* ML team (i.e., model developer) to train a relational graph convolutional network (RGCN) [6] model for fraud detection. After training, the model was deployed on the *AWS* platform (i.e., service provider) [7], and can be queried by the data owner (i.e., *Careem*) via the inference API. More cases include *Google Cloud*'s Vertex AI [8] and *Microsoft*'s Azure [9].

The widespread application of GNNs has also raised concerns about privacy, as trained GNN models can retain users' private information from the training data [10]. In this case, attackers can obtain private information through various methods, including edge-level (i.e., StealLink [11]) and node-level (i.e., MIA-Graph [12]) membership inference attacks (MIAs) with only black box access to the model. Regulations like the European General Data Protection Regulation (GDPR) [13] and the California Consumer Privacy Act (CCPA) [14], have all emphasized the *right to be forgotten (RTBF)*, requiring service providers to erase the influence of specific data upon users' request. Therefore, *Careem*'s users can delete their accounts and exercise their RTBF, requiring the company to modify the model deployed on *AWS*.

As illustrated by the blue arrows in Fig. 1, to meet the above requirement, the data owner (e.g., *Careem*) can ask the model developer (e.g., *Neptune* ML team) to perform the unlearning process and then request the service provider (e.g., *AWS*) to redeploy the updated model. The model developer can leverage graph unlearning methods to efficiently unlearn the required information [15], [16], [17]. Although various (certified) graph unlearning methods have been proposed, the unlearning process will take longer due to the uncertainty of developer task scheduling. Therefore, as shown by the brown lines in Fig. 1, we argue

[1] *Careem* is a leading technology company in the greater Middle East, offering an all-in-one app for transportation, food and grocery delivery, payment management, etc.

1545-5971 © 2026 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

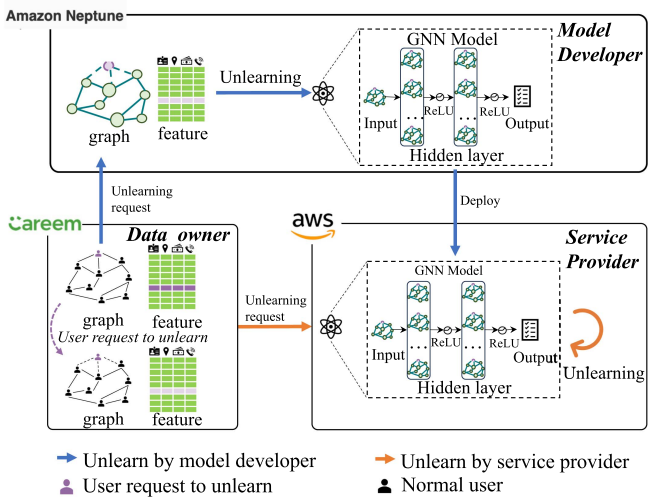


Fig. 1. An overview of different kinds of unlearning processes of GNNs in MLaaS. Blue lines show the model developer performing unlearning and the service provider redeploying the model. Blue lines indicate the service provider directly performing unlearning. Purple node indicates the user requesting data removal. The trained model is deployed in an independent environment that cannot access the training data.

that graph unlearning should also be deployable as a service, where service providers (e.g., *AWS*) directly receive unlearning requests (e.g., edges, nodes, or attributes to be unlearned, along with other necessary data) from data owners (e.g., *Careem*) and perform the unlearning process. Most graph unlearning methods cannot be directly applied to this setting because they require modifying GNN architectures or the training process, such as adding extra layers [16] or reference models [18], [19], [20], splitting data into shards [15], which is inconsistent with the conventional deployment of GNN models in MLaaS scenarios. Moreover, the service provider has access only to model-related information and cannot access all user data. This makes certified graph unlearning approaches inapplicable, as they generally assume access to the entire graph and estimate parameter changes using the Hessian and the gradient of the loss over the entire graph at the original model parameters [21], [22].

Graph unlearning for MLaaS requires addressing many new challenges. First, from the data owners' perspective, graph unlearning in MLaaS should use only data relevant to the unlearning requests. Second, from the perspective of model developers and service providers, graph unlearning should be compatible with existing deployment practices, minimizing modifications to the GNN architecture or training process. Third, unlearning methods should possess flexibility, supporting independent execution by either model developers or service providers.

To address the above requirements of data owners, model developers, and service providers, we propose a novel subgraph-based certified graph unlearning (SCGU) framework for the MLaaS scenario. Following the influence function principle [22], [23], we estimate the parameter change caused by unlearning with the gradient difference of parameters and the inverse Hessian matrix. However, unlike existing methods that rely on global graph data, SCGU localizes computation into subgraphs. First, we define the influenced subgraph consisting of the nodes and edges associated with the unlearning target[2] based on the local equivalence of nodes [24], thereby localizing the computation of gradient difference to the subgraph. Furthermore, by employing the conditional independence coefficient indicator (L-CODEC) from [25], we identified the subset of parameters that are most important to the unlearning target, and localized the computation of the inverse Hessian matrix of these parameters within the influenced subgraph via finite difference, which only takes the parameter weights and their sum of gradient from the last two training epochs. This mechanism allows model developers to pre-compute the sum of the gradients and share them with the service provider. Upon receiving an unlearning request, the service provider can directly perform unlearning using only the influenced subgraph. This method requires no modification to the GNN architecture or training process, ensuring perfect compatibility with existing MLaaS deployment practices.

In summary, our contributions are as follows:

- To the best of our knowledge, our proposed SCGU is the first framework that allows service providers under the MLaaS scenario to directly modify model parameters to achieve graph unlearning.
- Using approximate estimation, we decompose the model parameters after unlearning into multiple components, and the service provider only needs the influenced subgraph to perform graph unlearning, thereby minimizing the risk of privacy leakage from irrelevant nodes.
- To balance the trade-off between privacy protection and the unlearning effect, we propose a structural perturbation mechanism that injects controlled topological noise into the influenced subgraph, thereby reducing the size of the affected subgraph while reducing privacy leakage.
- Extensive experiments with 3 different unlearning tasks, 5 datasets, and 4 GNN backbone models demonstrate that our proposed SCGU can achieve results with parameters closer to those of a retrained model and outperform other baseline methods on unlearning efficacy in terms of AUC score and prediction loss.

The rest of the paper is organized as follows. Section II proposes the preliminaries and formulates the unlearning target in the MLaaS scenario. Then Section III introduces the system model and threat model. Section IV provides the intuition and details of our proposed SCGU. Section V discusses the experimental results on model utility, unlearning efficiency, and unlearning efficacy of our SCGU and other baseline methods. Finally, we briefly introduce related work in Section VI and summarize our work in Section VII.

[2]An unlearning target is the data elements (i.e., nodes, edges, or features) that need to be removed from the model, while an unlearning request includes unlearning targets and additional data (e.g., training graph or subgraph) required to complete the unlearning process. For instance, if a *Careem* user requests account deletion, the user's corresponding node would be the unlearning target, while data of other related users might also be included in the unlearning request to complete the process.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

TABLE I
SYMBOLS AND THEIR DEFINITIONS USED IN THIS PAPER

Symbol	Definition
G	graph
$\mathcal{V}$	nodes set
$\mathcal{E}$	edges set
$\mathcal{X}$	feature set
$\mathcal{V}_{trn}$	training node set
$\mathcal{Y}$	set of node label
$\mathcal{Y}_{trn}$	set of training node label
ΔG	unlearning target
$\Delta\mathcal{V}$	set of nodes to be unlearned
$\Delta\mathcal{E}$	set of edges to be unlearned
$\Delta\mathcal{X}$	set of features to be unlearned
Δ	gradient difference of model parameters
θ	parameter of GNN model
H_θ	Hessian matrix
$\nabla L(\theta,\cdot,\cdot)$	gradient of model parameters on loss
k	depth (layer number) of GNN

II. PRELIMINARIES

We first introduce the concepts, notations, and application scenarios related to GNNs in MLaaS and graph unlearning. Unless otherwise specified, we use bold uppercase letters (e.g., $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g., $\mathbf{x}$) to denote vectors, and normal letters (e.g., c) to denote scalars. Moreover, $G=(\mathcal{V},\mathcal{E},\mathcal{X})$ represents an attributed graph. Here $\mathcal{V}=\{v_1,v_2,\ldots,v_n\}$, $\mathcal{E}\subset\mathcal{V}\times\mathcal{V}$ and $\mathcal{X}=\{x_{1,1},x_{1,2},\ldots,x_{n,c}\}$ denotes the set of nodes, edges and features (vectorized attributes), respectively, where c is the feature dimension, and $x_{i,j}$ represents the feature value of node v_i at the j-th dimension. We list the main symbols used in this paper and their definitions in Table I for clarity.

A. GNNs in MLaaS

MLaaS is an emerging cloud service that enables model developers to conveniently build and deploy GNNs. Most MLaaS services have two phases: model training and deployment. During the training phase, model developers load the training graph provided by data owners and train GNNs accordingly. During the deployment phase, model developers first upload the trained GNN to service providers, who then deploy it in an independent environment to make the inference service available. Since the two phases have different design objectives and resource preferences, the training and deployment contexts are usually optimized and maintained separately.

GNNs are a class of deep learning models designed to handle graph-structured data. GNNs take graph topology data and node/edge features as input, producing node representations as output. Formally, the representation of node v in the k-th layer of a GNN model can be formulated as:

$$\mathbf{a}_v^{(k)}=AGGREGATE^{(k)}\left(\left\{\mathbf{h}_u^{(k-1)}:u\in\mathcal{N}(v)\right\}\right)$$
$$\mathbf{h}_v^{(k)}=COMBINE^{(k)}\left(\mathbf{h}_v^{(k-1)},\mathbf{a}_v^{(k-1)}\right) \quad (1)$$

where $AGGREGATE(\cdot)$ is the aggregation operation to collect information from node v's neighbor nodes and $COMBINE(\cdot)$ is the combination function to update the representation of v w.r.t. aggregated information. The designs of the aggregate and combine functions vary across models. $\mathbf{h}_v^{(k)}$ is the representation of node v obtained from the k-th layer, $\mathbf{h}_v^{(0)}=\mathbf{x}_v$. $\mathbf{a}_v^{(k-1)}$ is the integrated representation of neighbor nodes collected by the aggregation operation, $\mathcal{N}(v)$ is the neighbor node set of node v that contains nodes directly connected to it in graph G.

In this work, we focus on the widely used node classification task as the downstream task for GNNs. In the node classification task, GNNs are often equipped with an additional classification layer (e.g., a fully connected layer) to predict node labels from the node representations. Specifically, with the set of training nodes $\mathcal{V}_{trn}\subset\mathcal{V}$ and corresponding labels $\mathcal{Y}_{trn}\subset\mathcal{Y}$ provided by data owners, where $\mathcal{Y}_{trn}=\{Y_1,\ldots,Y_m\}$ contains labels for $m=|\mathcal{V}_{trn}|$ training nodes, the goal of model developers during training phase is to get the optimal GNN model θ^* that minimize the following training objective

$$\theta^*=\arg\min_\theta\sum_{v\in\mathcal{V}_{trn}}L(\theta,v,G) \quad (2)$$

where $L(\theta,v,G)$ denote the loss for node v w.r.t. model parameter θ and graph G, a typical choice of which is cross-entropy loss. And the goal of service providers during deployment is to run the trained GNN, take the inference graph and node IDs as input, and output the predicted labels for the node IDs. Here we consider that the computation of $L(\cdot,\cdot,\cdot)$ also relies on other necessary information, such as $\mathcal{Y}_{trn}$, and omit them for simplicity. For sake of simplicity, we also abbreviate $\sum_{v\in\mathcal{V}_{trn}}$ as $\sum_{\mathcal{V}_{trn}}$ and omit v from the notation $L(\cdot,\cdot,\cdot)$. For example, we use $\sum_{\mathcal{V}_{trn}}L(\theta,G)$ to denote $\sum_{v\in\mathcal{V}_{trn}}L(\theta,v,G)$. Unless otherwise specified, we use the similar meaning for $\sum$ and $L(\cdot,\cdot,\cdot)$.

Inductive and transductive GNNs in MLaaS: Inductive GNNs allow the model to use different graphs during training and inference, while transductive GNNs require the same graph. In this paper, we focus on *inductive GNNs in MLaaS*. That is, the service provider deploys the GNN and provides an inference API. The data owner uploads the inference graph and node IDs for prediction through the API. The service provider returns the predicted labels after completing the inference. In this setting, the service provider stores only model parameters and does not store training data. Therefore, the data is always controlled by the data owner and model developer. For the unlearning task, the data owner uploads the unlearning request incrementally. However, the data in the unlearning request should appear in the training graph, making this step transductive.

B. Graph Unlearning

Given the unlearning targets (e.g., nodes, edges, or features to be unlearned), graph unlearning aims to remove their influence from the trained GNN. Unlearning can be produced by model developers and service providers. Based on the different unlearning objectives, we focus on three common types of graph unlearning tasks: node unlearning, edge unlearning, and feature

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

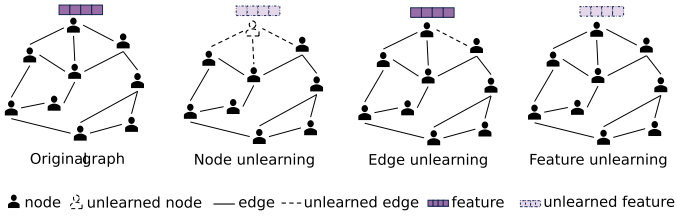


Fig. 2. Illustration of different unlearning tasks and corresponding unlearning targets. Unlearning targets are shown in dashed lines.

unlearning. $\Delta G = (\Delta\mathcal{V}, \Delta\mathcal{E}, \Delta\mathcal{X})$ denotes the unlearning targets contained, and the remaining graph can be represented as $G \ominus \Delta G = \{\mathcal{V} \setminus \Delta\mathcal{V}, \mathcal{E} \setminus \Delta\mathcal{E}, \mathcal{X} \setminus \Delta\mathcal{X}\}$.

As illustrated in Fig. 2, for the above three different types of unlearning tasks, the corresponding unlearning targets can be described as:

1) Node unlearning: Given a set of nodes to be unlearned ($\Delta\mathcal{V} \subset \mathcal{V}_{trn}$), node unlearning requires the removal of their features and structural information. For example, users of *Careem* can deregister their account and ask *Careem* to delete their information. In this case, the information that needs to be deleted from the graph includes the features and the edges directly related to these nodes, i.e., $\Delta G = (\Delta\mathcal{V}, \phi_{\mathcal{E}}(\Delta\mathcal{V}), \phi_{\mathcal{X}}(\Delta\mathcal{V}))$. Here $\phi_{\mathcal{E}}(\Delta\mathcal{V})$ and $\phi_{\mathcal{X}}(\Delta\mathcal{V})$ represent the set of edges and features directly associated with nodes in $\Delta\mathcal{V}$, respectively.

2) Edge unlearning: Given a set of edges (i.e., connections between users of *Careem*) to be unlearned ($\Delta\mathcal{E} \subset \mathcal{E}$), edge unlearning requires removing the information of these node pairs. The unlearning target can be formulated as $\Delta G = (\varnothing, \Delta\mathcal{E}, \varnothing)$.

3) Feature unlearning: All features related to the given set of nodes (i.e., $\mathcal{V}_x$) should be unlearned. To further illustrate, users of *Careem* can ask *Careem* to stop using their profiles for GNN training. The unlearning target can be formulated as $\Delta G = (\varnothing, \varnothing, \Delta\mathcal{X})$, where $\Delta\mathcal{X} = \{x_{i,j} | v_i \in \mathcal{V}_x\}$ is the set of node feature.

In this paper, we focus on certified graph unlearning methods because they can yield promising results by bounding the difference between the parameters of the unlearned and retrained models. Introductions to other types of graph unlearning can be found in Section VI.

Certified graph unlearning: We follow the commonly used (ϵ, δ)-certified graph unlearning [21] to measure the efficacy of unlearning, and formally define the certified graph unlearning problem for GNNs as follows:

Definition 1 ((ϵ, δ)-certified graph unlearning). Let Θ be the hypothesis space of model parameters, $\mathcal{A}$ be the optimization process, and $\mathcal{U}$ be the unlearning process. Given the graph G and unlearning target $\Delta G = (\Delta\mathcal{V}, \Delta\mathcal{E}, \Delta\mathcal{X})$ that specifies the node/edge/feature information to be unlearned, if $\forall\mathcal{T} \in \Theta$ we have

$$\begin{aligned} \Pr\left(\mathcal{U}(G, \Delta G, \mathcal{A}(G)) \in \mathcal{T}\right) &\leq e^{\epsilon} \Pr\left(\mathcal{A}(G \ominus \Delta G) \in \mathcal{T}\right) + \delta \\ \Pr\left(\mathcal{A}(G \ominus \Delta G) \in \mathcal{T}\right) &\leq e^{\epsilon} \Pr\left(\mathcal{U}(G, \Delta G, \mathcal{A}(G)) \in \mathcal{T}\right) + \delta \end{aligned} \tag{3}$$

then $\mathcal{U}$ is an (ϵ, δ)-certified graph unlearning process, where $G \ominus \Delta G$ denotes the graph data after removing the information specified by ΔG.

The basic intuition is that the unlearned model should be as similar as possible to the retrained model. Following this intuition, existing certified graph unlearning methods try to get the unlearned model w.r.t. the original model, and approximate the change in model weights as [22]:

$$\hat{\theta}^* - \theta^* \approx H_{\theta^*}^{-1} \Delta \tag{4}$$

where $H_{\theta^*} = \sum_{\mathcal{V}_{trn}} \nabla^2 L(\theta^*, v, G)$ represents the Hessian matrix of the original model calculated on graph data, representing the magnitude of the change in the parameters. $\Delta = \sum_{\mathcal{V}_{trn} \setminus \Delta V} \left(\nabla L(\theta^*, v, G) - \nabla L(\theta^*, v, G \ominus \Delta G)\right)$ represents the gradient difference of remaining nodes before and after the graph changes, indicating the changing direction of model parameters. To mitigate privacy exposure during unlearning, we draw inspiration from the graph influence function (GIF) [22], which takes the node dependency into consideration and limits the calculation scope of Δ to nodes influenced by the unlearning target, that is:

$$\Delta = \sum_{\hat{\mathcal{V}}} \nabla L(\theta^*, G) - \sum_{\hat{\mathcal{V}}'} \nabla L\left(\theta^*, G \ominus \Delta G\right) \tag{5}$$

where

$$\begin{aligned} \hat{\mathcal{V}} &= \cup_{v \in \Delta\mathcal{V}} \mathcal{N}_k(v) \cap \mathcal{V}_{trn} \\ \hat{\mathcal{V}}' &= \cup_{v \in \Delta\mathcal{V}} \mathcal{N}_k(v) \cap (\mathcal{V}_{trn} \setminus \Delta V) \end{aligned} \tag{6}$$

represent the set of nodes influenced by the unlearning target on the original graph and the remaining graph, respectively. $\mathcal{N}_k(v)$ is the set of nodes whose shortest path distance to node v in G is less than k. However, the Hessian and gradient difference calculations are still performed on the entire graph, which does not fit our scenario. We take a further step (detailed in Section IV-B) to limit the computation to the influenced subgraph, thereby avoiding the privacy exposure of unrelated nodes.

III. Graph Unlearning in MLaaS

In this section, we describe the scenarios explored in this paper. Relevant definitions and background information are detailed in Section II.

A. System Model

We focus on graph unlearning in MLaaS, where GNNs are developed locally but deployed on MLaaS servers, primarily for node classification tasks. As illustrated in Fig. 1, there are three types of entities: data owner, model developer, and service provider.

Data owner: It refers to the entities that have full access to graph data $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ and take them as their intellectual property (e.g. *Careem*). Data owners can authorize (part of) their graph data to model developers to train GNNs. After the GNN is trained and deployed, data owners can submit unlearning requests at any time. In our scenario, they consider providing only the data necessary for unlearning to model developers or service providers.

Model developer: It refers to the entities responsible for training GNNs (e.g. *Neptune* ML team). Model developers use graph data authorized by data owners to train GNNs and provide

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

relevant materials, such as models and code, to service providers, thereby supporting their GNN deployment. Model developers own the trained GNNs θ^* and have access to the graph data G provided by data owners. Since model developers have access to the data and model, they can take unlearning requests from data owners and execute the unlearning process themselves.

Service provider: It refers to the entities that own MLaaS Server (e.g. *AWS*). After receiving the trained GNN from the model developers, service providers deploy a model instance on the MLaaS server: it loads only the architecture and learned weights, deploys the corresponding inference and unlearning services, and does not retain the training data. During unlearning, service providers have full access to the trained GNNs θ^* and to other information provided, such as graph data transmitted by data owners via unlearning requests. In our scenario, service providers can deploy both inference and unlearning services of the given GNNs, allowing data owners to directly use unlearning services to remove the influence of specific data samples (nodes, edges, or features) on GNNs without exposing their full graph to the service provider.

Unlearning request: According to RTBF, data owners have the right to request the removal of their data from the model to protect their privacy. The unlearning request consists of the unlearning target $\Delta G = (\Delta\mathcal{V}, \Delta\mathcal{E}, \Delta\mathcal{X})$ and necessary data to support the unlearning process, i.e., the training graph or subgraphs sampled from it. Data owners can send an unlearning request to model developers, asking them to perform unlearning. They can also directly use the unlearning service deployed by service providers to modify the deployed GNNs.

In this paper, we primarily discuss the case where the unlearning process is performed by service providers, since model developers can access the entire training graph, whereas service providers can only access data uploaded by data owners. If the unlearning method can be deployed by service providers, then it can naturally be executed by model developers. Existing unlearning methods typically require access to the entire graph, which is impractical for service providers who can only access the data uploaded by data owners. Our proposed SCGU can be executed directly by service providers, thereby eliminating the need for model developers to manually adjust and upload the model. This can circumvent the constraints imposed by model developers' task scheduling and enhance operational efficiency.

B. Threat Model

The goal of graph unlearning is to protect data privacy and remove data's influence on GNNs. In our scenario, we assume that service providers will faithfully implement unlearning agreements and have no motive to steal data. The rationale for this assumption stems from the consideration that business reputation and SLA agreements make breach-of-contract damages and brand losses from data theft far outweigh the benefits, and compliance audits make malicious behavior easy to trace. While we assume the service provider is honest, we limit the uploaded information to an influenced subgraph to prioritize efficiency and defense-in-depth. From an efficiency perspective, this localization significantly reduces the communication and computation overhead. From a security standpoint, this aligns with the principle of data minimization. In the event of a future server intrusion, sharing less data can reduce the risk of leaks and make privacy protection more reliable.

However, even if the service provider is honest, the deployed GNN still faces the risk of privacy breaches from external sources. Existing work has shown that malicious adversaries can detect whether specific data is present in the training data through black box access to the model [11], [12], [26]. In this threat model, we model the adversary as an external third party with black box access: after the data owner (i.e., *Careem*) submits an unlearning request to the service provider, the adversary will launch a privacy attack on the nodes or edges claimed to be unlearned. If the unlearning mechanism is effective, the adversary will be unable to extract any information from the model's API response. This setup allows us to verify the privacy protection performance of the graph unlearning method under practical conditions.

IV. Subgraph-Based Certified Graph Unlearning

We first introduce intuition and an overview of our proposed SCGU, and then describe the detailed methodology.

A. Intuition and Overview

1) Intuition: To address the challenges posed to data owners, model developers, and service providers in the MLaaS scenario, we propose the subgraph-based certified graph unlearning (SCGU) framework to minimize privacy exposure by identifying and isolating components related to the unlearning request, and confining their computation to a subgraph.

Recall from Section II-B, the parameter change by unlearning can be estimated by the Hessian matrix of the original model and the gradient difference of remaining nodes before and after the graph change as:

$$\hat{\theta}^* - \theta^* \approx H_{\theta^*}^{-1}\Delta \tag{7}$$

where $H_{\theta^*} = \sum_{\mathcal{V}_{trn}} \nabla^2 L(\theta^*, v, G)$ represents the Hessian matrix of the original model calculated on graph data. $\Delta = \sum_{\mathcal{V}_{trn}\setminus\Delta V}(\nabla L(\theta^*, v, G) - \nabla L(\theta^*, v, G \ominus \Delta G))$ represents the gradient difference of remaining nodes before and after the graph changes. However, the calculation of the Hessian and gradient difference relies on the entire graph. This does not fit our scenario, where the service provider does not store the training graph. To resolve this conflict, we need to "localize" the entire derivation to the smallest region related to unlearning.

Localizing Gradient Difference within Influenced Subgraphs: We follow existing locally developed methods and begin by analyzing the goal of retraining, which is to find the optimal model $\hat{\theta}^*$ that minimizes the loss function $\sum_{\mathcal{V}_{trn}\setminus\Delta\mathcal{V}} L(\theta, v, G \ominus \Delta G)$, given the remaining graph $G \ominus \Delta G$, the remaining training nodes $\mathcal{V}_{trn} \setminus \Delta\mathcal{V}$, and their corresponding labels $\mathcal{Y}_{trn} \setminus \Delta Y$. According to the localized equivalence proposition [24], by categorizing the remaining nodes into those influenced and uninfluenced by the unlearning request, we can transform the

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

retraining objective into

$$\hat{\theta}^* = \arg\min_{\theta} \sum_{\mathcal{V}_{trn}\setminus\Delta\mathcal{V}} L(\theta, G\ominus\Delta G) = \arg\min_{\theta}\left(\sum_{\mathcal{V}_{trn}} L(\theta,G) - \sum_{\hat{\mathcal{V}}} L(\theta,G) + \sum_{\hat{\mathcal{V}}'} L(\theta, G\ominus\Delta G)\right) \tag{8}$$

where $\hat{\mathcal{V}}$ and $\hat{\mathcal{V}}'$ follow the same definition in (6). This objective function is consistent with the existing work. However, it should be noted that $\sum_{\hat{\mathcal{V}}} L(\theta, G)$ and $\sum_{\hat{\mathcal{V}}'} L(\theta, G\ominus\Delta G)$ are still calculated on the entire graph.

It can be found that $\sum_{\hat{\mathcal{V}}} L(\theta, G)$ and $\sum_{\hat{\mathcal{V}}'} L(\theta, G\ominus\Delta G)$ only calculate the loss of nodes influenced by the unlearning target, i.e., $\hat{\mathcal{V}}$ and $\hat{\mathcal{V}}'$. In fact, calculating the loss of these nodes does not require the entire graph. Specifically, according to the message-passing mechanism of GNNs [27], for a GNN with k layers, a node's gradient is only related to its k-hop neighbors. Based on this observation, we can obtain an influenced subgraph formed by the k-hop neighbors of nodes influenced by the unlearning target, which can be formally defined as follows:

Definition 2 (Influenced Subgraph). Given a graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, a node set $\hat{\mathcal{V}} \subseteq \mathcal{V}$, and the number of GNN layers k, the influenced subgraph $G_k(\hat{\mathcal{V}})$ of $\hat{\mathcal{V}}$ is the subgraph formed by nodes in $\hat{\mathcal{V}}$ and their k-hop neighbors in G, formally

$$G_k(\hat{\mathcal{V}}) = \left(\hat{\mathcal{V}}_k, \hat{\mathcal{E}}_k, \hat{\mathcal{X}}_k\right) \tag{9}$$

where $\hat{\mathcal{V}}_k = \cup_{v\in\hat{\mathcal{V}}}\mathcal{N}_k(v)$ is the set of nodes within k-hop distance from any node in $\hat{\mathcal{V}}$, $\hat{\mathcal{E}}_k = \left\{(u,v)|u,v\in\hat{\mathcal{V}}_k, (u,v)\in\mathcal{E}\right\}$ is the set of edges between nodes in $\hat{\mathcal{V}}_k$, and $\hat{\mathcal{X}}_k = \{x_{i,:}|v_i\in\hat{\mathcal{V}}_k\}$ is the feature set of nodes in $\hat{\mathcal{V}}_k$.

Thus, by extracting the influenced subgraph $G_k(\hat{\mathcal{V}})$, we can compute the loss of nodes influenced by unlearning the target without involving the entire graph. And (8) can be reformulated to

$$\hat{\theta}^* = \arg\min_{\theta} \sum_{\mathcal{V}_{trn}\setminus\Delta\mathcal{V}} L(\theta, G\ominus\Delta G) = \arg\min_{\theta}\left(\sum_{\mathcal{V}_{trn}} L(\theta,G) - \sum_{\hat{\mathcal{V}}} L\left(\theta, G_k(\hat{\mathcal{V}})\right) + \sum_{\hat{\mathcal{V}}'} L\left(\theta, G'_k\left(\hat{\mathcal{V}}'\right)\right)\right) \tag{10}$$

where $G_k(\hat{\mathcal{V}})$ is the influenced subgraph consisting of $v\in\mathcal{V}$ and its k-hop neighbors on G and $G'_k(\hat{\mathcal{V}}')$ is the updated influenced subgraph after removing unlearning targets from $G_k(\hat{\mathcal{V}})$.

Based on the influence function [22] and (10), we can use the gradient of the original model parameters (i.e., $\nabla L(\theta^*)$) to approximate that of retrained model parameters (i.e., $\nabla L(\hat{\theta}^*)$). By performing a first-order Taylor expansion of $\nabla L(\hat{\theta}^*)$ around the original model parameters θ^*, we can have

$$\nabla\mathcal{L}(\hat{\theta}^*) \approx \nabla\mathcal{L}(\theta^*) + \nabla^2\mathcal{L}(\theta^*)\left(\hat{\theta}^* - \theta^*\right) \tag{11}$$

where $\mathcal{L}(\theta) = \sum_{\mathcal{V}_{trn}} L(\theta, G) - \sum_{\hat{\mathcal{V}}} L(\theta, G_k(\hat{\mathcal{V}})) + \sum_{\hat{\mathcal{V}}'} L(\theta, G'_k(\hat{\mathcal{V}}'))$ for brevity.

Recall that as the optimal model on the remaining graph, the gradient of the retrained model on the remaining nodes should be 0, i.e., $\nabla\mathcal{L}(\hat{\theta}^*) = 0$. We can then estimate the parameter of the unlearned model $\hat{\theta}$ as follows

$$\hat{\theta}^* \approx \theta^* + \left(H_{\mathcal{V}_{trn}}\right)^{-1}\Delta \tag{12}$$

where $H_{\mathcal{V}_{trn}} = \sum_{\mathcal{V}_{trn}} \nabla^2 L(\theta, G)$ is the global Hessian component that relate to the original graph[3], $\Delta = \sum_{\hat{\mathcal{V}}} \nabla L(\theta, G_k(v)) - \sum_{\hat{\mathcal{V}}'} \nabla L(\theta, G'_k(v))$ is the gradient difference of the influenced nodes on the subgraph before and after unlearning. The Hessian matrix indicates the scale of parameter updates, while the gradient difference indicates the direction.

Localize Hessian Computation within Influenced Subgraph: In (12), only $H_{\mathcal{V}_{trn}}$ is related to the entire graph, while the other terms only require the influenced subgraph $G_k(\hat{\mathcal{V}})$. However, computing and storing the full Hessian matrix is $\mathcal{O}(|\theta|^3)$ and $\mathcal{O}(|\theta|^2)$, respectively, and thus infeasible for large-scale graphs and models. Existing methods mitigate this problem via Hessian-vector product (HVP) techniques [22], [24], but they require performing second-order gradient operations during the unlearning phase. This operation violates our MLaaS scenario, since the service provider cannot access the training data, nor maintain the corresponding computation graphs.

Following the observation that not all parameters in a model store information from a specific training sample with equal importance [16], [25], [28], we employ the conditional independence coefficient (L-CODEC) $\hat{T}_n$ from [25] to identify the subset of parameters that are most related to the unlearning target, thereby reducing the number of parameters required for Hessian matrix.

$\hat{T}_n(L, A)$ measures the statistical dependence between activations A and losses L. If $\hat{T}_n(L, A) = 0$, then activations of parameter and loss are independent.[4] By applying multiple random perturbations to the features of the influenced nodes, we collect the model's prediction loss under these perturbations, denoted by L, and the parameter activations, denoted by A. Then, we calculate the dependency of the activations w.r.t. the prediction loss conditionally using $\hat{T}_n(L, A)$, thus determining the subset of parameters P most related to the unlearning target.

Consistent with [25], we calculate the Hessian matrix of these parameters after obtaining θ_P through finite difference:

$$H = \frac{\Delta\mathcal{L}(\theta_P)\Delta\mathcal{L}(\theta_P)^\top}{|\Delta\theta_P|} \tag{13}$$

where $\Delta\mathcal{L}(\theta_P) = \nabla_1\mathcal{L}(\theta_P, G) - \nabla_2\mathcal{L}(\theta_P, G)$ is the gradient difference of the selected parameters at the last and penultimate epoch of full training, and $|\Delta\theta_P|$ is the corresponding parameter difference.

[3]Following the prior work [22], [24], we neglect the term $\sum_{\hat{\mathcal{V}}} \nabla^2 L(\theta, G_k(\hat{\mathcal{V}}))$ and $\sum_{\hat{\mathcal{V}}'} \nabla^2 L(\theta, G'_k(\hat{\mathcal{V}}'))$ here since ΔG is usually a tiny subset of G.

[4]Similarly, given selected activations Z, $\hat{T}_n(L, A|Z)$ indicates the conditional dependency of A on L w.r.t. Z

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

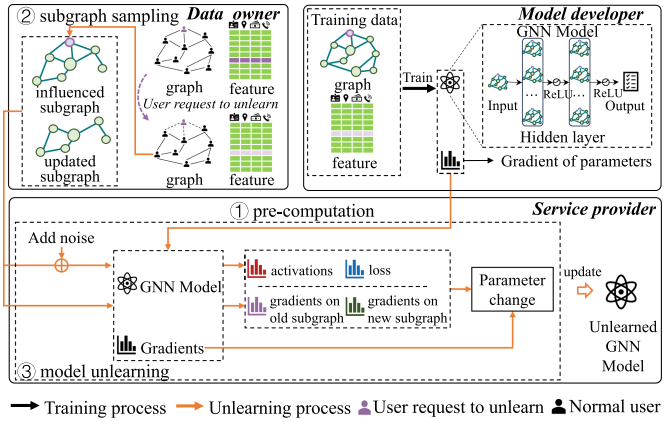


Fig. 3. An overview of our SCGU. During the pre-computation procedure, the model developer computes the Hessian of the weights on the training data and uploads it to the service provider. During the subgraph sampling procedure, the data owner first samples and processes the subgraph with respect to the unlearning target, then uploads it to the service provider. Finally, the service provider computes the gradients of the weights over subgraphs and estimates the parameter differences during the model unlearning procedure.

It can be observed that both parameter subset calculation and gradient difference computation can be performed within the influenced subgraph. Therefore, by recording and saving the gradients and weights of the last two epochs at the end of training, the data owner is only required to transmit the influenced subgraph $G_k(\hat{\mathcal{V}})$ to the service provider in the unlearning phase, thus extending the certified graph unlearning method to the MLaaS scenario.

2) Overview: As illustrated in Fig. 3, our proposed SCGU comprises three procedures: pre-computation, subgraph sampling, and model unlearning. In the pre-computation procedure, the model developer calculates and saves the model's parameters and the gradients from the last two training epochs, then uploads them to the service provider. In the subgraph sampling procedure, the data owner first samples the influenced subgraph from the original graph with respect to the unlearning target, then processes it according to the unlearning request. Subsequently, the data owner transmits the subgraphs to the service provider. Finally, in the model unlearning procedure, the service provider calculates the gradients of nodes influenced by unlearning within the subgraphs and combines them with the Hessian uploaded by the model developer to compute the model's parameter update.

While we primarily describe the case where the model unlearning procedure is performed by the service provider, it can also be easily performed by the model developer, who has access to all training data. In this case, after pre-computation, the model developer does not need to upload the gradients and parameters to the service provider. After receiving the unlearning request from the data owner, the model developer performs the model unlearning procedure directly. The unlearned model is then uploaded to the service provider for redeployment.

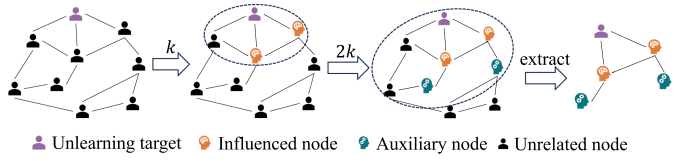


Fig. 4. An overview of our sampling technique. It takes the unlearning target as the core and radiates outward, sampling layer by layer to form the influenced subgraph. The sampling process is repeated $2k$ times. We took $k = 1$ here for clarity.

B. Detailed Methodology

Pre-computation: The pre-computation procedure is completed before unlearning. During pre-computation, the model developer is required to record the sum of gradients of model weights w.r.t. the entire graph ($H_{\mathcal{V}_{trn}}$) and the corresponding weights of the last and penultimate training epoch, and upload it to the service provider, allowing the service provider to store it in the hosting environment. It does not need the model developer to perform any extra operations. If the unlearning is conducted by model developers, they only need to record the sum of gradients of model weights $\left(\sum_{\mathcal{V}_{trn}} \nabla L(\theta, G)\right)$. This can be readily accomplished during the training phase.

The service provider stores the gradient sum over all training samples with respect to the model parameters. The aggregation of gradients compresses information from individual nodes into a summary, making it impossible for adversaries to reverse the aggregation and reconstruct the raw data. Besides, the model developer collects the gradients from the last two epochs, when the gradient differences between batches are minimal, since the training tends to stabilize and the specific information of individual samples is further reduced.

Subgraph sampling: To approximate the parameter updates in (10), the data owner needs to provide the influenced subgraph $G_k(\hat{\mathcal{V}})$ to the service provider. However, the size of $G_k(\hat{\mathcal{V}})$ grows exponentially with respect to the depth of GNN due to the neighborhood explosion [29]. This not only increases communication cost, but also exposes many non-target nodes that are only structurally related to the unlearning target.

To mitigate this, we introduce a structural perturbation mechanism by randomly perturbing the expansion process during subgraph construction. Specifically, starting from the unlearning target, we randomly mask a limited subset of neighboring nodes and edges at each hop, forming a perturbed approximation of the influenced subgraph.

As shown in Fig. 4, we start from the unlearned nodes, progressively sample at most b_l neighboring nodes for each frontier node for $2k$ steps, and then construct $G_k(\hat{\mathcal{V}})$ using the union of these nodes. We repeat the sampling process $2k$ times to obtain sufficient nodes to support the computation for a k-layer GNN: the first k steps collect influenced nodes whose gradients are used to estimate the parameter update, while the last k steps collect auxiliary nodes that only support message passing for the influenced nodes and do not contribute their own gradients. For node and feature unlearning tasks, we directly construct the influenced subgraph based on the target node. For the edge unlearning task, where the target is the association

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

between nodes, we construct the influenced subgraph based on the endpoints of these edges.

We quantify the structural exposure induced by subgraph sampling using the relative size of the uploaded subgraph. Let $\mathcal{V}_{sam}$ and $\mathcal{V}_{full}$ be the node sets obtained by perturbed sampling and full $2k$-hop expansion, respectively, and d be the average degree. The sampled subgraph grows no faster than $|\Delta\mathcal{V}|\left(1+\sum_{r=1}^{2^{\wedge}k}\prod_{l=1}^{r}b_l\right)$, whereas the full-size subgraph grows approximately as $|\Delta\mathcal{V}|\left(1+\sum_{r=1}^{2^{\wedge}k}d^r\right)$. Therefore, when $b_l \ll d$, structural perturbation substantially reduces the relative exposure compared with full expansion. The design is not to eliminate exposure or provide a formal privacy guarantee, but rather to make exposure explicitly controlled by the sampling budgets, yielding a tunable trade-off among exposure, unlearning efficacy, and efficiency.

Subsequently, the data owner processes $G_k(\hat{\mathcal{V}})$ based on the unlearning target $\Delta G = (\Delta\mathcal{V}, \Delta\mathcal{E}, \Delta\mathcal{X})$ to obtain $G'_k(\hat{\mathcal{V}}')$. For feature unlearning tasks, the unlearned features (i.e., $\Delta\mathcal{X}$) are set to zero. For node unlearning tasks, all target nodes (i.e., $\Delta\mathcal{V}$) and the edges directly connected to them are removed. For edge unlearning tasks, the target edges (i.e., $\Delta\mathcal{E}$) are removed from the influenced subgraph. The data owner then sends $G_k(\hat{\mathcal{V}}), G'_k(\hat{\mathcal{V}}')$ and the unlearning target ΔG to the service provider.

Model unlearning: In the model unlearning procedure, the service provider first selects the parameter subset to compute the Hessian matrix, then computes the gradient differences of the influenced nodes before and after unlearning based on the influenced subgraph, and finally estimates the parameter change by combining the Hessian matrix and the gradient differences.

After receiving the influenced subgraph $G_k(\hat{\mathcal{V}})$, the service provider first performs multiple perturbations on the influenced nodes. Specifically, in the j-th perturbation, for each influenced node $v \in \hat{\mathcal{V}}$, the service provider randomly sample noise $\delta^j \sim N(0,1)$ and perturb its features as

$$\hat{x}_v^j = x_v + \delta^j \tag{14}$$

thus construct the j-th perturbed influenced subgraph $\hat{G}_k^j(\hat{\mathcal{V}})$. After that, the service provider feed $\hat{G}_k^j(\hat{\mathcal{V}})$ to the GNN and collects its predicted loss l^j on the influenced nodes, as well as the activations a_{Li}^j associated with each parameter, where a_{Li}^j is the activation values of i-th parameter in layer L under the j-th perturbation. After m perturbations, the service provider collects m sets of activations $A = \left(a_{11}^1, \ldots, a_{L1}^m, \ldots, a_{Li}^m\right)$ and losses $L = \left(l^1, l^2, \ldots, l^m\right)$.

Then, following the L-CODEC framework [30], the service provider greedily constructs the parameter subset θ_P based on dependencies between activations and losses, iteratively selecting parameters whose associated activations maximize the (un)conditional dependency gain until the scores fall below the threshold. Specifically, we first iterate over all activations, select the one with the highest unconditional dependency $\hat{T}_n(L, A)$ as the starting element Z_1, and add the associated parameter to the subset θ_P. We then iteratively evaluate the conditional dependencies $\hat{T}_n(L, A_{rest}|A_P)$ of all remaining activations A_{rest} on L, and add the parameter whose activation has the highest score to θ_P until all $\hat{T}_n(L, A_{rest}|A_P)$ are below threshold (e.g., ≤ 0).

Finally, the service provid'er computes the gradients of parameters $\sum_{\hat{\mathcal{V}}}\nabla L(\theta^*, G_k(\hat{\mathcal{V}}))$ and $\sum_{\hat{\mathcal{V}}'}\nabla L(\theta^*, G'_k(\hat{\mathcal{V}}'))$, as well as the gradient changes $\Delta = \sum_{\hat{\mathcal{V}}}\nabla L(\theta^*, G_k(\hat{\mathcal{V}})) - \sum_{\hat{\mathcal{V}}'}\nabla L(\theta^*, G'_k(\hat{\mathcal{V}}'))$. Thereafter, combining the Hessian H computed via finite difference, the service provider calculates the parameter difference according to (12).

Batch unlearning and synchronization: Since the influenced subgraph we proposed does not need to be connected, and the perturbation is applied to all influenced nodes, our method can naturally extend to batch unlearning. In batch unlearning, we process multiple unlearning requests using a single unlearning operation by merging multiple subgraphs into a single graph. For handling sequence unlearning requests, after the batch unlearning operation, the service provider can upload a small state summary (e.g., the parameter change) to the model developer. The model developer can use this cache to update the parameters and gradients from the last epoch; the updated parameters and gradients can then be treated as the new parameters and gradient for the last training epoch. The model developer can then upload them to the service provider for upcoming unlearning requirements.

Complexity Analysis: Since the pre-computation can be performed efficiently during training, the unlearning process mainly involves activation and loss generation, Hessian and gradient updates, and parameter updates. The time complexity is $O(m|\theta| + m|\theta|\log m + |\theta_P|^3)$, where $|\theta_P| \ll |\theta|$ is the size of the parameter subset, $O(m|\theta|)$ is the time complexity of activation and loss generation, $O(m|\theta|\log m)$ is the time complexity of parameter subset construction, and $O(|\theta_P|^3)$ is the time complexity of Hessian matrix inversion. The space complexity is $O(4|\theta_P|)$, mainly related to the stored gradient and parameters.

C. Theorectical Analysis

In this section, we theoretically prove that under the approximations of graph localization and parameter localization, SCGU achieves (ϵ', δ')-certified unlearning. We first present assumptions below.

Assumption 1. For the training objective of a given GNN model, we assume the objective loss function $\mathcal{L}(\cdot)$ is L-Lipschitz continuous and λ-strongly convex. For graph localization and parameter localization, we assume the influenced subgraph contains unlearning targets and their k-hop neighborhood.

$P \subseteq \{1, \ldots, d\}$ represents the set of parameter indices selected by L-CODEC, $|P| = p$ represents the number of selected parameters, and the remaining set of parameter indices is $R = \{1, \ldots, d\} \setminus P$. A selection matrix $S_P \in \{0,1\}^{d\times p}$ selects θ_P from θ, such that $\theta_P = S_P^\top\theta$, similarly defining $S_R \in \{0,1\}^{d\times(d-p)}$. Similarly, we have $\Delta_p = S_P^\top\Delta$, and $\Delta_r = S_R^\top\Delta$.

Let $\hat{\theta}^*_{Full}$ denote the parameters obtained by performing unlearning using the Hessian matrix calculated exactly over all parameters, and $\hat{\theta}^*_{SCGU}$ denote the parameters obtained by performing unlearning using the Hessian matrix calculated approximately only over the selected parameters θ_P. If we divide the Hessian matrix calculated over all parameters into blocks

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

according to P and R, we can obtain:

$$H = \begin{bmatrix} H_{PP} & H_{PR} \\ H_{RP} & H_{RR} \end{bmatrix} = \begin{bmatrix} S_P^\top H_{\theta^*} S_P & S_P^\top H_{\theta^*} S_R \\ S_R^\top H_{\theta^*} S_P & S_R^\top H_{\theta^*} S_R \end{bmatrix} \tag{15}$$

Since $\mathcal{L}(\theta, G)$ is twice differentiable and λ-strongly convex, we have $H_{\theta^*} \succeq \lambda I_d$, therefore that H_{θ^*} is invertible, and $\|H_{\theta^*}^{-1}\|_2 \leq 1/\lambda$. Since $H_{\theta^*} \succeq \lambda I_d$, its principal submatrices also satisfy $H_{PP} \succeq \lambda I_d$, and $\|H_{PP}^{-1}\|_2 \leq 1/\lambda$. SCGU only estimates the Hessian on θ_P. Let $\widetilde{H}_{PP}$ denote H_{PP} obtained by finite difference approximation. The parameter update of SCGU can be expressed as $\hat{\theta}^*_{SCGU} = \theta^* + S_P \widetilde{H}_{PP}^{-1} \Delta_P$.

Based on the above description, we first analyze the distance between the ideally certified unlearned model $\hat{\theta}^*_{Full}$ and the model obtained by SCGU $\hat{\theta}^*_{SCGU}$.

Lemma 1. Let $\rho_H = \|\widetilde{H}_{PP} - H_{PP}\|_2$, and $\rho_H < \lambda$, $\eta_P = \|\Delta_R - H_{RP} H_{PP}^{-1} \Delta_P\|_2$. Then $\hat{\theta}^*_{Full}$ and $\hat{\theta}^*_{SCGU}$ satisfies:

$$\left\| \hat{\theta}^*_{Full} - \hat{\theta}^*_{SCGU} \right\|_2 \leq \frac{\eta_P}{\lambda} + \frac{\rho_H \|\Delta_P\|_2}{\lambda(\lambda - \rho_H)} \triangleq C_P$$

Proof. The proof proceeds in two main steps by introducing an intermediate state $\hat{\theta}^*_P$, which represents the model obtained by unlearning via the exact Hessian matrix on the selected parameter θ_P. First, we bound the distance between the full parameter update $\hat{\theta}^*_{Full}$ and this intermediate state $\hat{\theta}^*_P$. Next, we bound the error introduced by approximating the Hessian (i.e., the distance between $\hat{\theta}^*_P$ and $\hat{\theta}^*_{SCGU}$) using Weyl's inequality and the resolvent identity. The final bound C_P is obtained via the triangle inequality. The complete and detailed proof is provided in the Appendix, available online.

Previous work has proved that graph unlearning methods based on the Hessian matrix computed with all parameters satisfy ϵ, δ-certified unlearning. We further prove that SCGU satisfies a relaxed ϵ', δ'-certified unlearning.

Theorem 1. Let $\hat{\theta}^*_{\text{SCGU}}$ denote the output of SCGU before randomization. The randomized output

$$\tilde{\theta}^*_{\text{SCGU}} = \hat{\theta}^*_{\text{SCGU}} + \zeta, \qquad \zeta \sim \mathcal{N}\left(0, \sigma^2 I_d\right).$$

satisfies (ϵ', δ')-certified graph unlearning, where for any auxiliary parameter $\delta_a \in (0, 1)$,

$$\epsilon' = \epsilon + \epsilon_a, \qquad \epsilon_a = \frac{C_P}{\sigma} \sqrt{2 \ln \frac{1.25}{\delta_a}},$$

and

$$\delta' = \max\left\{ e^{\epsilon_a} \delta + \delta_a, \delta + e^{\epsilon} \delta_a \right\}.$$

Proof. The complete proof can be found in the Appendix, available online. The core idea is to utilize the bounded parameter distance C_P. By viewing the outputs of the full-parameter method and SCGU as two Gaussian mechanisms, we apply the mean-shift bound for Gaussian mechanisms to relate their output distributions. Combining this shift with the existing (ϵ, δ)-certified unlearning guarantees of the full-parameter method, we can derive the relaxed privacy bounds ϵ' and δ' for SCGU.

TABLE II
DATASETS STATISTICS

name	#nodes	#edges	#features	#classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,723	3,703	6
PubMed	19,717	88,648	500	3
CS	18,333	163,788	6,805	15
Physics	34,493	495,924	8,415	5

V. EXPERIMENTS

In this section, we systematically evaluate the performance of our SCGU with various unlearning methods for GNNs across multiple dimensions. We first introduce the basic experimental setup in Section V-A, including datasets, baselines, unlearning tasks, and backbone models. In Section V-B, we analyze the impact of different unlearning methods on model utility by comparing their influence on node classification tasks. In Section V-C, we investigate the efficiency of these methods across datasets of varying scales. Additionally, we employ membership inference attacks (MIAs) to assess the efficacy of unlearning in our method and baselines, as described in Section V-D. Finally, in Section V-E, we analyze the impact of different sampling strategies on the model utility, unlearning efficiency, and efficacy of our proposed SCGU method.

A. Experimental Setup

Datasets: We adopt the widely studied node classification task as the downstream task, which encompasses many real-world GNN-based applications. We conduct experiments on the following real-world datasets, including Cora [31], CiteSeer [31], PubMed [31], Coauthor-CS (CS for short) [32], and Coauthor-Physics (Physics for short) [32]. These datasets are commonly used as benchmarks for GNN performance in node classification and link prediction tasks. Among them, Cora, CiteSeer, and PubMed are citation network datasets in which nodes represent research publications, edges represent citation relationships between any pair of publications, and node features are bag-of-words representations of keywords. CS and Physics are coauthor networks, where nodes represent authors and edges represent the collaboration relationship. More details can be found in Table II.

Backbone Model: We use four widely-used GNNs as backbone models, including GCN [31], GAT [33], GIN [34], and SGC [35], to test the effectiveness of SCGU across different types of GNNs. Among them, GCN, GAT, and GIN are non-linear GNNs that contain multiple GNN layers connected by non-linear activation functions. SGC is a linear version of GCN that removes all nonlinearities and retains only the final linear layer for prediction.

Unlearning Tasks: We consider three common types of unlearning tasks on graphs, including edge unlearning, feature unlearning, and node unlearning. Unless otherwise specified, we uniformly require that each unlearning task requires unlearning at most 5% of the corresponding instance. We randomly select the unlearning target to ensure objectivity.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

Baselines: We compare SCGU against several existing certified graph unlearning methods and introduce additional retraining baselines. The details of the comparison methods are as follows. We did not use additional graph unlearning methods because they either can only be used with certain GNNs as backbone models [36], [37] or do not provide unlearning guarantees [18], [19], which does not align with the scope of the methods we discussed.

- Retrain. The most straightforward approach is to delete the data that needs to be unlearned, then retrain the model from scratch. This is an exact unlearning method and naturally guarantees no unlearned information is used in model training.
- GraphEraser [15]. It is an efficient retraining-based method that first divides the training graph data into several independent shards and then trains submodels on each shard separately. During unlearning, it locates the shard where the target data resides, updates the shard data, and then retrains the corresponding sub-model. Specifically, the GraphEraser method can employ two sharding strategies: the label propagation algorithm (LPA) [38] and the k-means algorithm [39]. Chen et al. [15] denote the GraphEraser method employing the LPA strategy as BLPA, and the one using the k-means strategy as BEKM. Both of them serve as our baselines.
- CGU [21]. It generalizes the influence function [23] to graph data, supporting graph unlearning tasks. CGU directly computes the analytical solution of the Hessian matrix and estimates the parameter change. However, it is only applicable to linear GNNs such as SGC.
- CEU [40]. It specializes in edge unlearning by introducing a novel influence function that accounts for local neighborhood dependencies. Similar to CGU, it provides closed-form parameter updates but is theoretically restricted to linear or convex GNNs.
- IDEA [24]. It is a flexible framework for certified graph unlearning that also approximates an unlearned model of the original model using the influence function. It uses the Hessian-vector product and a stochastic estimation method to iteratively approximate parameter changes. Compared to CGU, it combines multiple unlearning tasks and produces flexible, certified graph unlearning for various GNN models.

Since IDEA offers a more flexible framework, other certified graph unlearning methods (e.g., GIF [22]) can be considered as special cases of IDEA. Therefore, we use IDEA as the baseline and do not compare SCGU with other certified graph unlearning methods.

Evaluation Metrics: We measure our SCGU in three aspects: model usability, unlearning efficiency, and efficacy, and compare them with other methods.

- Model utility: we use the F1 score of the node classification task as a metric. The higher the F1 score, the less the model's utility is affected by unlearning.
- Unlearning efficiency: we record the running time of different unlearning methods to reflect their unlearning efficiency. The shorter the running time, the higher the unlearning efficiency.
- Unlearning efficacy: since the unlearning task includes a clear target, we use MIA methods to assess whether the target data remains in the training data. We use the AUC score obtained from the attacks as a metric for unlearning efficacy. The lower the AUC score, the more difficult it is for the attack methods to determine whether the unlearning target is included in the training data, thus indicating the better efficacy of the unlearning method.

Implement Details: All of our experiments were conducted on a server equipped with 4 Intel Xeon Gold 6348 CPUs and 5 Nvidia A100 GPUs. SCGU is implemented with PyTorch and PyTorch Geometric Library. Without specification, we uniformly sample up to 3 neighbor nodes for each node in each layer. For baseline methods, we use their official implementations and freeze the random seed in the relevant library to ensure they learn from the same training dataset.

B. Model Utility

To analyze the impact of different unlearning methods on model utility, we compare the performance of GNN architectures on node classification after employing these methods. Better classification results indicate that the unlearning method preserves the model's utility more effectively. As mentioned earlier, we conduct experiments on all five datasets: Cora, Citeseer, PubMed, CS, and Physics. We use Retrain, GraphEraser, and IDEA as baseline methods, and the F1 score as the metric for evaluating classification performance.

Table III illustrates the impact of different unlearning methods on model usability, and the results are scaled up by a factor of 100 for the purpose of clear visualization. The higher the F1 score, the less impact the method has on model utility. SCGU achieves competitive results in most cases, particularly in node unlearning and feature unlearning tasks. For example, on the Cora dataset with the GIN model, SCGU achieves an F1 score of 84.62 for the node unlearning task, outperforming the IDEA baseline (83.69) and other retraining-based methods. These results highlight the effectiveness of SCGU in maintaining model utility while performing unlearning tasks.

Additionally, since the goal of certified graph unlearning is to obtain a model sufficiently similar to a retrained model, we also compared the similarity of the models obtained by SCGU and IDEA to retrained models. We use the L2 norm of the difference in model parameters as the metric. A lower L2 norm indicates that the model is closer to the retrained model, thus more similar to it. We conducted comparisons on Cora, Citeseer, and PubMed datasets using all four backbone models. We show the results in Fig. 5 for models obtained under different unlearning tasks. SCGU consistently achieves lower L2 norms across various unlearning tasks on most datasets and models, indicating that it can produce models that are more similar to retrained models.

C. Unlearning Efficiency

We conduct experiments on the Cora, Citeseer, and PubMed datasets to demonstrate the efficiency of various unlearning methods across datasets of different scales. Since CGU only supports SGC as the backbone model for unlearning, we adopt

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

TABLE III

F1 SCORE RESULTS ON DIFFERENT DATASETS AND BACKBONE MODELS. UNLEARNING TASKS ARE DENOTED AS $\mathcal{T}_{node}$ (NODE UNLEARNING), $\mathcal{T}_{edge}$ (EDGE UNLEARNING) AND $\mathcal{T}_{feat}$ (FEATURE UNLEARNING), RESPECTIVELY. **BOLD** INDICATES THE OPTIMAL RESULTS. THE F1 SCORES ARE SCALED BY A FACTOR OF 100 FOR CLEARER VISUALIZATION. ALL RESULTS ARE ILLUSTRATED IN THE FORMAT OF MEAN ± STANDARD DEVIATION

model	method	Cora			Citeseer			Pubmed			CS			Physics		
		$\mathcal{T}_{node}$	$\mathcal{T}_{edge}$	$\mathcal{T}_{feat}$	$\mathcal{T}_{node}$	$\mathcal{T}_{edge}$	$\mathcal{T}_{feat}$	$\mathcal{T}_{node}$	$\mathcal{T}_{edge}$	$\mathcal{T}_{feat}$	$\mathcal{T}_{node}$	$\mathcal{T}_{edge}$	$\mathcal{T}_{feat}$	$\mathcal{T}_{node}$	$\mathcal{T}_{edge}$	$\mathcal{T}_{feat}$
GIN	Retrain	84.62±0.01	83.89±0.01	**83.76±0.02**	72.47±0.01	**73.17±0.01**	**73.07±0.02**	86.43±0.00	**86.78±0.00**	**86.71±0.01**	91.22±0.00	91.22±0.00	91.38±0.00	95.57±0.00	**95.65±0.00**	**95.78±0.00**
	BLPA	65.54±0.03	62.66±0.03	-	59.58±0.05	59.88±0.02	-	84.28±0.00	84.10±0.00	-	71.55±0.01	65.71±0.03	-	80.83±0.02	80.70±0.01	-
	BEKM	75.94±0.02	76.31±0.02	-	67.45±0.02	69.13±0.02	-	85.25±0.01	86.65±0.00	-	89.66±0.01	89.30±0.00	-	92.56±0.00	92.84±0.00	-
	CEU	-	81.67±0.01	-	-	71.27±0.01	-	-	86.66±0.00	-	-	91.25±0.00	-	-	95.63±0.00	-
	IDEA	83.69±0.01	81.25±0.02	80.96±0.03	72.43±0.01	67.39±0.04	70.21±0.03	86.53±0.00	86.34±0.01	86.54±0.00	91.53±0.00	**91.28±0.00**	**91.55±0.00**	94.91±0.01	95.57±0.00	93.39±0.05
	SCGU	**84.62±0.01**	**83.39±0.01**	79.46±0.00	**72.47±0.01**	**72.77±0.00**	*71.77±0.01*	**86.43±0.00**	*86.61±0.00*	*84.63±0.02*	**91.22±0.00**	90.91±0.00	90.31±0.00	**95.57±0.00**	95.43±0.00	*95.47±0.00*
GAT	Retrain	85.98±0.01	86.47±0.00	**86.22±0.00**	75.98±0.00	75.78±0.00	**75.98±0.00**	84.21±0.00	84.25±0.00	84.26±0.00	92.71±0.00	92.69±0.00	92.71±0.00	95.88±0.00	**95.95±0.00**	95.92±0.00
	BLPA	49.96±0.01	50.04±0.03	-	60.54±0.02	65.17±0.02	-	72.04±0.01	71.90±0.01	-	71.85±0.00	70.09±0.02	-	73.21±0.00	72.55±0.00	-
	BEKM	63.62±0.02	65.17±0.02	-	62.64±0.02	63.90±0.04	-	70.61±0.00	71.85±0.00	-	76.74±0.00	75.50±0.00	-	78.81±0.00	77.50±0.00	-
	CEU	-	86.22±0.01	-	-	75.98±0.00	-	-	83.84±0.00	-	-	92.54±0.00	-	-	94.88±0.00	-
	IDEA	85.68±0.02	86.20±0.01	85.54±0.01	75.74±0.00	76.04±0.00	75.98±0.01	84.16±0.00	84.14±0.00	84.33±0.00	92.68±0.00	92.74±0.00	**92.76±0.00**	**95.89±0.00**	95.87±0.00	**95.98±0.00**
	SCGU	**85.98±0.01**	**86.59±0.01**	75.28±0.02	**75.98±0.00**	**76.38±0.00**	74.27±0.00	**84.21±0.00**	**84.31±0.00**	**84.65±0.00**	**92.71±0.00**	**92.78±0.00**	92.22±0.00	*95.88±0.00*	*95.89±0.00*	95.77±0.00
GCN	Retrain	84.13±0.01	**84.26±0.01**	**84.26±0.01**	74.87±0.01	**74.77±0.01**	74.67±0.01	85.07±0.00	**85.02±0.00**	**85.04±0.00**	92.78±0.00	**92.38±0.00**	**93.00±0.01**	96.12±0.00	**96.07±0.00**	96.04±0.00
	BLPA	72.99±0.00	70.33±0.01	-	64.20±0.00	68.23±0.00	-	69.99±0.03	72.61±0.01	-	86.53±0.00	86.46±0.00	-	77.54±0.00	76.52±0.01	-
	BEKM	64.50±0.01	66.13±0.00	-	66.91±0.01	67.87±0.01	-	70.34±0.00	70.77±0.00	-	80.12±0.00	79.97±0.00	-	75.81±0.00	75.61±0.00	-
	CEU	-	79.83±0.01	-	-	71.27±0.01	-	-	81.11±0.01	-	-	92.01±0.00	-	-	95.25±0.00	-
	IDEA	83.17±0.02	80.07±0.01	83.47±0.01	69.97±0.01	68.17±0.02	69.97±0.02	82.63±0.00	80.33±0.03	82.60±0.00	92.51±0.00	91.85±0.00	92.01±0.01	95.47±0.00	95.40±0.00	**95.77±0.00**
	SCGU	**84.13±0.01**	*83.76±0.00*	84.13±0.01	**74.87±0.01**	*74.37±0.01*	**75.18±0.01**	**85.07±0.00**	*83.37±0.00*	*83.99±0.00*	**92.78±0.00**	*91.97±0.00*	91.88±0.00	**96.12±0.00**	*95.70±0.00*	95.61±0.00
SGC	Retrain	84.50±0.00	**84.50±0.00**	84.50±0.00	**72.87±0.01**	**72.87±0.01**	**72.87±0.01**	84.80±0.00	**84.80±0.00**	**84.80±0.00**	93.15±0.00	**93.42±0.00**	**92.98±0.00**	95.94±0.00	**95.97±0.00**	**96.05±0.00**
	BLPA	72.99±0.00	70.33±0.01	-	64.20±0.00	68.23±0.00	-	69.99±0.03	72.61±0.01	-	86.53±0.00	86.46±0.00	-	77.54±0.00	76.52±0.01	-
	BEKM	63.91±0.00	66.64±0.01	-	67.69±0.01	67.45±0.01	-	70.76±0.00	70.46±0.00	-	80.33±0.00	79.78±0.00	-	76.67±0.00	75.64±0.00	-
	CGU	87.46±0.00	84.31±0.00	**87.11±0.00**	**74.84±0.00**	67.87±0.00	72.74±0.00	74.85±0.00	75.59±0.00	72.98±0.00	87.70±0.00	77.54±0.00	87.77±0.00	92.49±0.00	91.65±0.00	92.39±0.00
	CEU	-	79.83±0.01	-	-	69.67±0.01	-	-	80.76±0.01	-	-	91.97±0.00	-	-	94.91±0.00	-
	IDEA	82.07±0.01	79.63±0.01	81.92±0.02	68.71±0.02	67.63±0.02	69.19±0.02	82.55±0.01	79.06±0.02	82.63±0.01	92.43±0.01	91.24±0.00	92.44±0.01	95.33±0.00	95.09±0.00	95.07±0.00
	SCGU	**84.50±0.00**	83.39±0.00	82.90±0.00	72.87±0.01	*72.57±0.01*	*72.77±0.01*	**84.80±0.00**	*83.20±0.00*	84.30±0.00	**93.15±0.00**	*92.97±0.00*	91.44±0.00	**95.94±0.00**	*95.72±0.00*	*95.19±0.00*

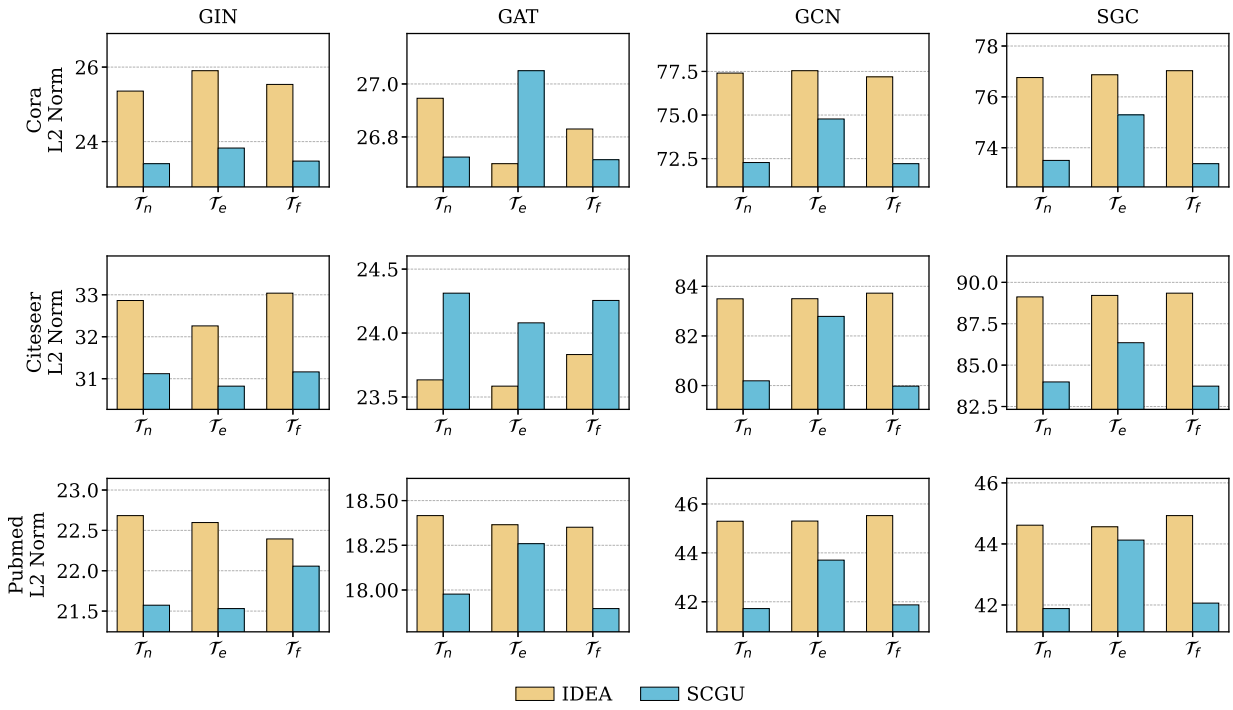


Fig. 5. Distance between unlearned models and retrained models measured by L2 norm. A lower L2 norm indicates that the model is closer to the retrained model.

SGC for all other baselines to ensure a fair comparison. We use the same setup as in the model utility experiment to compare efficiency across node, feature, and edge unlearning tasks, and uniformly require each task to unlearn 5% of the corresponding instances.

The experimental results are illustrated in Table IV. Since GraphEraser does not support the feature unlearning task, we use "-" to indicate that no results are obtained. GraphEraser has the lowest efficiency because it requires locating the shards that contain the data to be unlearned and then retraining the corresponding model. If data is distributed across multiple shards, the number of sub-models that need retraining will also increase, leading to a longer unlearning process. SCGU is more efficient than retraining, GraphEraser, and CGU on almost all datasets and tasks because it computes only a portion of the Hessian matrix, whereas CGU computes the full Hessian matrix. IDEA outperforms SCGU on certain datasets like Citeseer, since it utilizes stochastic estimation to estimate the inverse Hessian matrix in an iterative manner, with a time complexity of $O(t|\theta|)$, where t is the total number of iterations. However, the stochastic estimation method requires a second-order gradient operation, which requires the original graph; our method localized the

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
RUNNING TIME ON DIFFERENT DATASETS AND UNLEARNING TASKS. UNLEARNING TASKS ARE DENOTED WITH THE SAME SYMBOL IN TABLE III, AND TIMES ARE MEASURED IN SECONDS. OPTIMAL RESULTS ARE HIGHLIGHTED IN **BOLD**

Dataset	Task	Retrain	BLPA	BEKM	CGU	CEU	IDEA	SCGU
cora	$\mathcal{T}_{node}$	1.41	3.12	2.88	0.91	-	0.20	**0.06**
	$\mathcal{T}_{edge}$	1.46	3.54	3.47	0.63	0.12	0.24	**0.11**
	$\mathcal{T}_{feat}$	1.43	-	-	0.87	-	0.20	**0.10**
citeseer	$\mathcal{T}_{node}$	2.52	3.49	3.72	1.18	-	**0.14**	0.29
	$\mathcal{T}_{edge}$	2.41	3.59	4.08	0.29	0.13	**0.15**	0.39
	$\mathcal{T}_{feat}$	2.33	-	-	1.19	-	**0.14**	0.18
pubmed	$\mathcal{T}_{node}$	2.7	9.95	10.04	0.33	-	0.20	**0.15**
	$\mathcal{T}_{edge}$	2.72	10.13	10.35	0.32	0.13	**0.15**	0.25
	$\mathcal{T}_{feat}$	2.67	-	-	0.32	-	0.20	0.24
CS	$\mathcal{T}_{node}$	34.08	18.44	19.97	5.08	-	1.61	**0.90**
	$\mathcal{T}_{edge}$	32.26	13.79	13.44	3.18	1.62	**1.86**	2.04
	$\mathcal{T}_{feat}$	32.14	-	-	4.89	-	**1.80**	2.12
Physics	$\mathcal{T}_{node}$	138.73	56.48	55.32	0.85	-	1.69	**0.18**
	$\mathcal{T}_{edge}$	131.95	56.71	57.73	**0.11**	1.24	2.05	0.36
	$\mathcal{T}_{feat}$	139.73	-	-	0.77	-	1.68	**0.18**

computation of the Hessian matrix and gradient differences to the influenced subgraph, making it deployable by the service provider.

D. Unlearning Efficacy

Membership Inference Attack: We primarily employ MIA methods to evaluate the unlearning efficacy of our SCGU relative to other unlearning methods. Although there are other attack methods targeting GNNs, such as model inference attacks, property inference attacks [41], [42], [43], and data reconstruction attacks [44], these methods primarily do not aim to infer the membership of specific data samples. Specifically, model inference attacks focus on extracting model-related information. Property inference attacks primarily aim to infer properties of the graph, such as node degrees [41], [42], or properties of nodes (such as gender or income in social networks [43]). Data reconstruction attacks aim to reconstruct graphs similar to the training graph.

We used the unlearned models obtained from the model utility experiments on the Cora, Citeseer, PubMed, CS, and Physics datasets as target models and employed state-of-the-art MIA methods to predict whether unlearned information remains in the training data. For evaluating node unlearning efficacy, we adopt MIA-Graph [12]; for edge unlearning efficacy, we adopt StealLink [11]. MIA-Graph first trains a shadow GNN [5] on the same graph, and then trains the classifier on the shadow model's posteriors to predict the membership of target nodes. StealLink first extracts pairwise features by querying the target GNN for node posteriors and then trains a binary classifier on these features to predict the existence of edges. We use the same adversary setting for both MIA-Graph and StealLink, which gives the adversary black box access to the target GNN and its graph topology data.

The results on unlearning efficacy of different unlearning methods are illustrated in Fig. 6(a) and (b). Since CGU only supports SGC as the backbone, we present CGU results only within the SGC model results and mark the others with "-". We use the AUC score as a metric to reflect the effectiveness of attacks. The lower the AUC score, the worse the attack effect, which indicates better unlearning efficacy. As illustrated in Fig. 6(a), our proposed SCGU achieved optimal unlearning performance after the node unlearning task across 4 test scenarios and outperformed retraining-based methods in 9 of the remaining 16 scenarios. This indicates that SCGU can effectively unlearn node information. In the unlearning effects after the edge unlearning task illustrated in Fig. 6(b), our proposed SCGU achieved the optimal results outside of retraining-based methods in 14 out of the total 20 scenarios. We attribute this to the shard&retrain method losing some edge information during sharding and to aggregating predictions from multiple submodels, which further complicates the MIA method's ability to obtain accurate predictions.

[5]We follow the initial setting in [12] and use a 2-layer GCN as shadow GNN

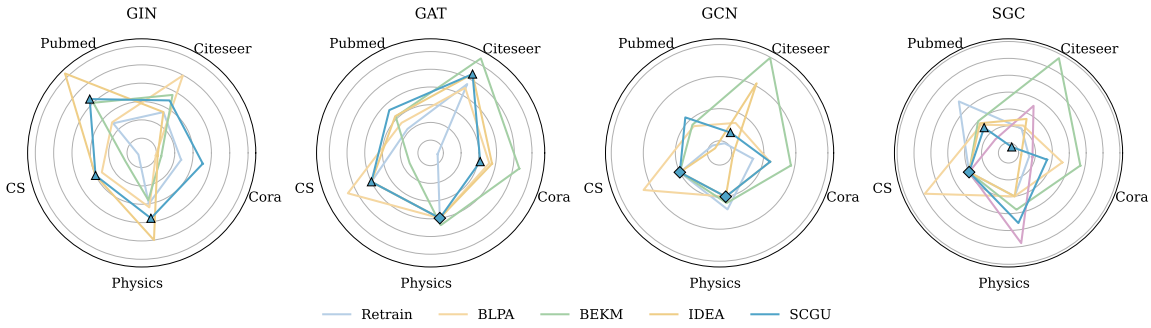
TABLE V
AVERAGE LOSS VALUE FOR UNLEARNING METHODS. THE FEATURES OF UNLEARNED NODES ARE SET TO ZERO. LOWER VALUES INDICATE BETTER PERFORMANCE. **BOLD** INDICATES THE OPTIMAL RESULTS ACHIEVED BY SCGU OUTSIDE OF RETRAINING-BASED METHODS

dataset	model	Retrain(↓)	CGU(↓)	IDEA(↓)	SCGU(↓)
cora	GIN	0.0125	-	1.0282	1.4031
	GAT	0.9228	-	0.9479	1.2367
	GCN	0.4802	-	0.6376	**0.5420**
	SGC	0.4282	1.6198	0.5358	0.7826
citeseer	GIN	0.1907	-	3.5446	**2.0259**
	GAT	1.2214	-	1.2293	1.3513
	GCN	0.7869	-	0.9999	**0.8121**
	SGC	0.7260	1.4835	0.9045	**0.7826**
pubmed	GIN	0.2633	-	1.2203	**0.8900**
	GAT	0.6112	-	0.6347	0.7329
	GCN	0.4362	-	0.5131	**0.4599**
	SGC	0.4354	1.0791	0.5423	**0.4516**
CS	GIN	0.0063	-	0.1730	0.5215
	GAT	0.2133	-	0.2329	0.7222
	GCN	0.1017	-	0.2325	0.3026
	SGC	0.0780	2.0730	0.3215	**0.2924**
Physics	GIN	0.0176	-	0.0686	0.2299
	GAT	0.1720	-	0.1426	0.4641
	GCN	0.1398	-	0.1030	0.1832
	SGC	0.1709	1.0200	0.1733	0.1944

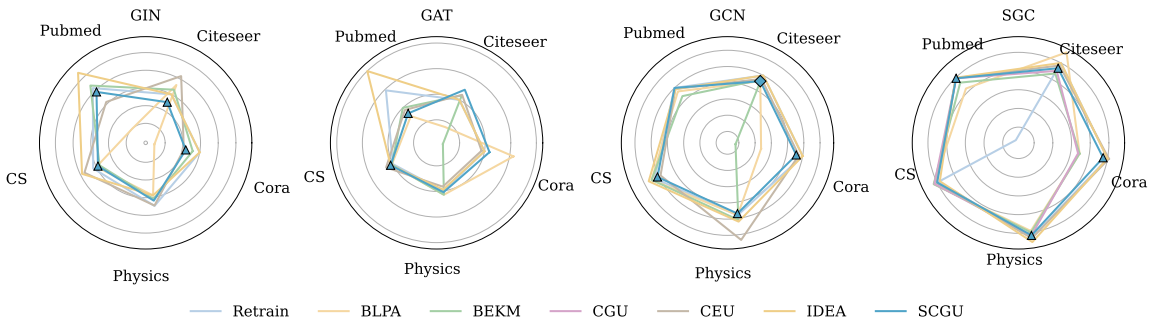
Since there are no existing membership inference attack methods targeting feature unlearning tasks, we adopt the method from [24] and use the average loss value as a metric for unlearning efficacy. We set the feature values for the target nodes to zero and calculate the average loss. A lower loss value indicates better feature unlearning efficacy.

The average loss values for various methods are shown in Table V. Because CGU only supports SGC as the backbone model, we use "-" to indicate that results for other models are not available. The results closest to Retrain are shown in bold. On more than half of the datasets and models, SCGU yields results closest to the retraining method, indicating that our method can obtain models that are more closely aligned with retrained models in terms of feature unlearning efficacy.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.



(a) Membership inference attack results after node unlearning.



(b) Membership inference attack results after edge unlearning.

Fig. 6. Membership inference attack results after node unlearning and edge unlearning. ♢ indicates the optimal results achieved by SCGU, and △ indicates the optimal results achieved by SCGU outside of retraining-based methods. The gradually spreading circles represent gradually increasing AUC values.

Deletion Inference Attack: Inspired by [45], we also evaluate whether the behavioral difference between the original and unlearned models can reveal unlearned edges. We use unlearned edges as positive samples and the same number of retained edges as negative samples. Following StealLink, we train the attack model on the original GNN's predictions, then apply it to both the original and unlearned GNNs to predict edge existence. The difference between the two prediction scores is used to infer whether an edge was unlearned. An AUC closer to 0.5 indicates better privacy, since the difference score can be either positive or negative.

All results are shown in Table VI. The results show that retraining-based methods, including Retrain, BLPA, and BEKM, typically maintain an AUC of around 0.5, indicating retraining does not significantly change the model's behavior. CEU and IDEA exhibit significant bias in some cases. In contrast, SCGU achieves comparable results in most cases. This is likely because SCGU updates only the parameters most relevant to the unlearning requirements, thereby reducing unnecessary influence on model behavior.

TABLE VI
DELETION INFERENCE ATTACK RESULTS AFTER EDGE UNLEARNING. **BOLD** INDICATES THE OPTIMAL RESULT

dataset	Model	Retrain	BLPA	BEKM	CEU	IDEA	SCGU
Cora	GIN	**0.4962**	0.4635	0.5097	0.5209	0.5697	0.5079
	GAT	0.5151	**0.4893**	0.4805	0.5571	0.3937	0.5185
	GCN	0.5063	0.4906	0.4592	0.5166	0.5111	**0.4997**
	SGC	0.5122	0.4929	0.4398	**0.505**	0.5418	0.5391
Citeseer	GIN	0.588	0.6156	0.5332	0.6085	0.5715	**0.5205**
	GAT	**0.4867**	0.477	0.4023	0.5778	0.5339	0.478
	GCN	0.5408	0.5603	0.522	0.5484	0.4692	**0.4963**
	SGC	0.4742	0.5406	0.5196	**0.5098**	0.4692	0.5333
Pubmed	GIN	0.4991	0.5337	0.501	0.5053	0.5361	**0.5005**
	GAT	0.546	**0.4942**	0.494	0.5421	0.4454	0.5265
	GCN	0.557	0.419	**0.5158**	0.5456	0.4219	0.4183
	SGC	0.516	0.4414	**0.5101**	0.4882	0.4591	0.5166
CS	GIN	0.5045	0.5112	0.4934	0.4979	0.517	**0.5007**
	GAT	**0.5016**	0.4935	0.4981	0.5299	0.5224	0.5018
	GCN	0.4959	0.4838	0.5146	**0.4976**	0.5036	0.5065
	SGC	0.4981	0.492	0.5197	**0.5018**	0.5064	0.5049
Physics	GIN	0.5036	**0.4996**	0.4965	0.5073	0.4987	0.504
	GAT	0.4938	0.4926	0.4941	0.5152	0.5054	**0.5033**
	GCN	0.5061	0.4971	0.4928	0.5028	0.5044	**0.502**
	SGC	0.5041	0.4934	0.488	0.5119	**0.503**	0.5156

E. Impact on Sampling Strategy

As discussed in Section IV, structural perturbation is introduced to reduce the exposure of non-target nodes during subgraph construction. To evaluate the impact of structural perturbation, we perform a quantitative analysis of the node unlearning task on the CS dataset using two representative backbone models, GCN and SGC. Specifically, we vary the sampling size b_l and evaluate its effect on structural exposure, model utility, unlearning efficiency, and unlearning efficacy. We use the CS dataset because its average degree is high enough to yield sufficiently large influenced subgraphs, and focus on

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
IMPACT OF SAMPLING STRATEGY ON STRUCTURAL EXPOSURE, MODEL UTILITY, UNLEARNING EFFICIENCY, AND EFFICACY

b_l	structural exposure	GCN utility	GCN efficiency	GCN efficacy	SGC utility	SGC efficiency	SGC efficacy
1	32.88%	0.9298	0.7414	0.5010	0.9311	0.2282	0.5000
2	67.95%	0.9264	1.1357	0.5000	0.9286	0.7241	0.5000
3	87.61%	0.9295	0.5894	0.4990	0.9193	0.7679	0.5000
4	94.91%	0.9278	0.5610	0.4990	0.9318	0.7538	0.5000
5	98.16%	0.9295	0.9383	0.5000	0.9131	0.5456	0.5000
6	99.19%	0.9275	0.9453	0.5000	0.9346	0.5148	0.4940
7	99.71%	0.9271	0.4893	0.5000	0.9295	0.5493	0.4950
8	100%	0.9191	1.3726	0.5000	0.9318	0.5584	0.5000

node unlearning because node-level exposure can be directly measured by the size of the sampled influenced subgraph.

Table VII shows that subgraph sampling can substantially reduce the disclosed subgraph size while preserving unlearning quality. When $b_l = 1$, only 33% of the fully influenced subgraph is uploaded, reducing the number of exposed nodes by about 67%. As b_l increases, the exposure ratio quickly approaches 1, indicating that a small sampling budget is already sufficient to cover most of the influenced subgraph on CS. Nevertheless, the model utility on both GCN and SGC remains comparable to that of the full-sized influenced subgraph, and the unlearning efficacy remains close to the ideal value of 0.5. The improvement in utility and unlearning efficacy is marginal, suggesting that the full $2k$-hop influenced subgraph contains redundant information for estimating the unlearning update. Overall, the results show that the sampling strategy effectively reduces subgraph exposure without noticeably sacrificing model utility or unlearning efficacy.

VI. RELATED WORK

In this section, we first introduce existing GNN methods for MLaaS scenarios, with a primary focus on privacy protection, and then describe methods for graph unlearning.

GNNs in MLaaS Scenarios: As MLaaS services become more popular, GNNs in MLaaS settings have also begun to receive increasing attention. Some work has begun on studying the privacy protection of GNNs deployed in MLaaS, focusing on designing methods to protect private information in graphs. LPGNet [46] designed a new method of using graph edge structures during training to provide differential privacy (DP) guarantees for edges. GraphGuard [47] proposed a training-data-free method to detect graph data misuse in MLaaS platforms, as well as an unlearning method to mitigate the influence of data misuse. Their method mainly focuses on protecting edge information and uses feature data, both related and unrelated to the unlearning target.

There are also works that attempt to obtain private information about the graph by attacking GNNs with black box access. StealLink [11] comprehensively classified the adversary's background knowledge from three dimensions and designed eight different edge-stealing attack methods. LinkTeller [26] infers private edge information by designing adversarial queries. He et al. [48] systematically defined threat models for node-level membership inference and proposed three attack methods. MIA-graph [12] expanded membership inference attacks to graph data, training shadow models to attack target GNNs.

Moreover, some works attempt to steal information about GNNs, such as their parameters, to build sufficiently similar alternative models. DeFazio et al. [49] extend the learning-based extraction approach to GNNs. They first collect query samples by iteratively perturbing the subgraph of the victim GNNs' original graph, then train a similar stealing model via these samples. Wu et al. [50] systematically formalize threat models and propose various attack methods based on different background knowledge of the attacker. Shen et al. [51] proposed a model extraction attack method for inductive GNNs. They first initialize the graph structure by performing K-nearest neighbors clustering based on node attributes, then update it using a graph structure learning framework. StealGNN [52] proposed a data-free model extraction attack framework that replaces the need for real-world data with synthetic graphs generated by a generative model. However, we mainly focus on protecting data privacy, thus leaving the protection of model information via unlearning for future work.

Graph Unlearning: The goal of graph unlearning is to eliminate the influence of certain training data on GNN so that it behaves as if it has never seen such data. The ideal approach is to remove this data and retrain the GNN model, but this is often costly. Many graph unlearning methods attempt to unlearn information in GNNs through (re)training. GraphEraser [15] and GUIDE [17] partition the training graph data into shards, and train sub-models on them independently. During unlearning, they would retrain the corresponding sub-model on the related shard after deleting the unlearned data. CCGU [53] further designed a mapping method between the community graph and the original graph, so as to flexibly adjust the scale of data required for retraining. GUKD [19], MEGU [20], and D2DGN [18] introduce knowledge distillation methods that use teacher models to guide student models in accomplishing unlearning requests. SUMMIT [54] utilizes multi-objective optimization to optimize the trade-off between unlearning structural data and preserving model utility. GNNDelete [16] adds an extra unlearning layer to eliminate the effect of unlearning data. DGU [55] proposed a graph unlearning method for GNNs on continuous-time dynamic graphs, and designed a 2-layer MLP-Mixer model to predict the parameter changes caused by unlearning. HSCD [56] proposes a hypergraph unlearning framework. It utilizes a size-based hyperedge selection and coverage aggregation method to efficiently remove the influence of specific hyperedges. However, despite some of them targeting different graph types, these methods do not guarantee the efficacy of their unlearning, thereby limiting their applications. Moreover, most of them require modifying the GNN architecture or using a specific training process, which is difficult to integrate with the general training and deployment processes for GNNs in MLaaS scenarios.

On the other hand, some work tries to approximate the changes of model weights before and after unlearning, thereby directly modifying the GNN. We focus on certified graph unlearning methods because they can provide upper bounds on the discrepancy between the unlearned and retrained models. Chien et al. analyzed the certified graph unlearning mechanism based

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

on influence function and proposed certified graph unlearning methods [21], [37], [57]. Wu et al. proposed CEU [40] and EraEdge [58] for unlearning edges visible to GNNs during training. GIF [22] and IDEA [24] establish distinct scopes of influence for different graph unlearning tasks (node unlearning, edge unlearning, feature unlearning) and compute the corresponding parameter changes. Unlike these influence-function-based methods that rely on global graph computations, our SCGU localizes updates to an influenced subgraph and a selected subset of parameters, ensuring efficiency and privacy during unlearning for MLaaS. Besides influence-function-based methods, GraphEditor [36] uses ridge regression as the objective, and updates the model by computing the closed-form solution. Projector [59] projects the weight parameters into a subspace unrelated to the features of target nodes to achieve unlearning. ETR [28] proposed a two-stage method that balances the unlearning effect and model utility by erasing first and then rectifying the parameters. However, it provides unlearning certification only at the erasing stage, leaving the rectifying stage unconstrained.

Federated unlearning: Federated unlearning addresses the issue of guaranteeing the right to be forgotten in federated learning scenarios. FedEraser [60] is one of the earliest proposed methods for federated unlearning. It achieves unlearning through retraining, accelerating the recovery phase by utilizing historical parameter updates stored on the server. FedRecover [61] uses retained updates from clients to estimate parameter updates during retraining and employs the L-BFGS [62] algorithm for efficient approximation of the Hessian matrix. FAST [63] directly adjusts model parameters and derives the unlearned model by utilizing updates retained by the server from all clients. Wu et al. [64] proposed a knowledge distillation-based federated unlearning mechanism, using knowledge distillation to quickly recover the global model performance after removing historical model updates from unlearned clients. Zhao et al. [65] proposed a knowledge elimination strategy called Momentum Degradation (MoDe), which decouples the federated unlearning process from training, making it applicable to any model architecture that has completed federated learning without altering the training process. Recently, ULIA [66] revealed that the unlearning process itself might leak sensitive labels. Furthermore, studies including FinBack [67], PILE [68], and ROBY [69] have explored the effects of federated unlearning from the perspectives of privacy preservation and backdoor attacks. However, these federated unlearning methods are not designed for GNNs, and their client-server collaborative training paradigm requires data owners to participate in model maintenance, which contradicts our scenario that data owners only provide training graphs. Therefore, these methods cannot be directly applied, and we still consider existing graph unlearning methods as baselines.

VII. Conclusion

In this paper, we presented a practical approach to extend certified graph unlearning to MLaaS scenarios, aiming to minimize the data needed during the unlearning process. By decomposing the Hessian matrix and precomputing the global Hessian component for the entire graph, SCGU enables data owners to provide only the affected subgraphs to service providers. Additionally, we introduce a sampling technique to further reduce data transmission overhead while preserving unlearning performance. Our SCGU is compatible with existing deployment practices, thus it can be deployed by both model developers and service providers. Extensive evaluations on multiple datasets and GNN architectures across various unlearning tasks demonstrate that SCGU can effectively unlearn target data while maintaining model utility. Besides, in our method, we introduced a sampling method to obtain an influenced subgraph. Although our strategy is empirically motivated rather than theoretically optimal, we view it as a trade-off between structure preservation and privacy exposure. Future work includes formalizing this trade-off to provide stronger guarantees regarding the minimality or optimality of the influenced subgraph, as well as exploring more effective approximation methods to reduce computational overhead during unlearning.

References

[1] D. Jin et al., "A survey on fairness-aware recommender systems," *Inf. Fusion*, vol. 100, 2023, Art. no. 101906.

[2] D. Wang et al., "A semi-supervised graph attentive network for financial fraud detection," in *Proc. IEEE Int. Conf. Data Mining*, 2019, pp. 598–607.

[3] J. Wang, S. Zhang, Y. Xiao, and R. Song, "A review on graph neural network methods in financial applications," *J. Data Sci.*, vol. 20, no. 2, pp. 111–134, 2022.

[4] X. Li et al., "BrainGNN: Interpretable brain graph neural network for fMRI analysis," *Med. Image Anal.*, vol. 74, 2021, Art. no. 102233.

[5] Y. Zhou, S. Graham, N. A. Koohbanani, M. Shaban, P.-A. Heng, and N. Rajpoot, "CGC-Net: Cell graph convolutional network for grading of colorectal cancer histology images," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop*, 2019, pp. 388–398.

[6] M. Schlichtkrull et al., "Modeling relational data with graph convolutional networks," in *Proc. Eur. Semantic Web Conf.*, 2018, pp. 593–607.

[7] "How Careem is detecting identity fraud using graph-based deep learning and amazon neptune AWS machine learning blog," Section: Amazon Neptune, Nov. 2021. [Online]. Available: https://aws.amazon.com/blogs/machine-learning/how-careem-is-detecting-identity-fraud-using-graph-based-deep-learning-and-amazon-neptune/

[8] Google Cloud, "Vertex AI platform," Accessed: Jan. 2, 2025.Online. Available: https://cloud.google.com/vertex-ai?hl=en

[9] Microsoft Azure, "Azure machine learning - ML as a service| Microsoft Azure," Accessed: Jan. 29, 2024. [Online]. Available: https://azure.microsoft.com/en-US/products/machine-learning

[10] Q. Tan, Q. Li, Y. Zhao, Z. Liu, X. Guo, and K. Xu, "Defending against data reconstruction attacks in federated learning: An information theory approach," in *Proc. USENIX Secur.*, 2024, pp. 325–342.

[11] X. He, J. Jia, M. Backes, N. Z. Gong, and Y. Zhang, "Stealing links from graph neural networks," in *Proc. USENIX Secur.*, 2021, pp. 2669–2686.

[12] I. E. Olatunji, W. Nejdl, and M. Khosla, "Membership inference attack on graph neural networks," in *Proc. IEEE Int. Conf. Trust, Privacy Secur. Intell. Syst. Appl.*, 2021, pp. 11–20.

[13] J. Rosen, "The right to be forgotten," *Stanford Law Rev.*, vol. 64, 2011, Art. no. 88.

[14] S. L. Pardau, "The California consumer privacy act: Towards a European-style privacy regime in the United States?," *J. Technol. Law Policy*, vol. 23, no. 1, 2018, Art. no. 2.

[15] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang, "Graph unlearning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 499–513.

[16] J. Cheng, G. Dasoulas, H. He, C. Agarwal, and M. Zitnik, "GNNDelete: A general strategy for unlearning in graph neural networks," in *Proc. Int. Conf. Learn. Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=X9yCkmT5Qrl

[17] C.-L. Wang, M. Huai, and D. Wang, "Inductive graph unlearning," in *Proc. USENIX Secur.*, 2023, pp. 3205–3222.

[18] Y. Sinha, M. Mandal, and M. Kankanhalli, "Distill to delete: Unlearning in graph networks with knowledge distillation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 37, no. 1, pp. 428–437, Jan. 2026.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.

[19] W. Zheng, X. Liu, Y. Wang, and X. Lin, "Graph unlearning using knowledge distillation," in *Proc. Int. Conf. Inf. Commun. Secur.*, 2023, pp. 485–501.

[20] X. Li, Y. Zhao, Z. Wu, W. Zhang, R.-H. Li, and G. Wang, "Towards effective and general graph unlearning via mutual evolution," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 13682–13 690.

[21] E. Chien, C. Pan, and O. Milenkovic, "Certified graph unlearning," in *Proc. NeurIPS Workshop: New Front. Graph Learn.*, 2022.

[22] J. Wu, Y. Yang, Y. Qian, Y. Sui, X. Wang, and X. He, "GIF: A general graph unlearning strategy via influence function," in *Proc. ACM Web Conf.*, 2023, pp. 651–661.

[23] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1885–1894.

[24] Y. Dong, B. Zhang, Z. Lei, N. Zou, and J. Li, "IDEA: A flexible framework of certified unlearning for graph neural networks," in *Proc. ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2024, pp. 621–630.

[25] R. Mehta, S. Pal, V. Singh, and S. N. Ravi, "Deep unlearning via randomized conditionally independent hessians," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10422–10431.

[26] F. Wu, Y. Long, C. Zhang, and B. Li, "LINKTELLER: Recovering private edges from graph neural networks via influence analysis," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 2005–2024.

[27] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1025–1035.

[28] Z.-R. Yang, J. Han, C.-D. Wang, and H. Liu, "Erase then rectify: A training-free parameter editing approach for cost-effective graph unlearning," in *Proc. AAAI Conf. Artif. Intell.*, 2025, pp. 13044–130 51.

[29] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *Proc. Int. Conf. Learn. Representations*, 2020.

[30] M. Azadkia and S. Chatterjee, "A simple measure of conditional dependence," *Ann. Statist.*, vol. 49, no. 6, pp. 3070–3102, 2021.

[31] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2022.

[32] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," 2018, *arXiv:1811.05868*..

[33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018.

[34] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. Int. Conf. Learn. Representations*, 2019.

[35] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2019, vol. 97, pp. 6861–6871.

[36] W. Cong and M. Mahdavi, "GraphEditor: An efficient graph representation learning and unlearning approach," 2022. [Online]. Available: https://openreview.net/forum?id=tyvshLxFUtP

[37] C. Pan, E. Chien, and O. Milenkovic, "Unlearning graph classifiers with limited data resources," in *Proc. ACM Web Conf.*, 2023, pp. 716–726.

[38] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E*, vol. 76, Sep. 2007, Art. no. 036106.

[39] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[40] K. Wu, J. Shen, Y. Ning, T. Wang, and W. H. Wang, "Certified edge unlearning for graph neural networks," in *Proc. ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2023, pp. 2606–2617.

[41] Z. Zhang, M. Chen, M. Backes, Y. Shen, and Y. Zhang, "Inference attacks against graph neural networks," in *Proc. USENIX Secur.*, 2022, pp. 4543–4560.

[42] X. Wang and W. H. Wang, "Group property inference attacks against graph neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 2871–2884.

[43] V. Duddu, A. Boutet, and V. Shejwalkar, "Quantifying privacy leakage in graph embedding," in *Proc. ACM EAI Int. Conf. Mobile Ubiquitous Syst.: Comput., Netw. Serv.*, 2021, pp. 76–85.

[44] M. Lin, E. Dai, J. Xu, J. Jia, X. Zhang, and S. Wang, "Stealing training graphs from graph neural networks," in *Proc. ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2025, pp. 777–788.

[45] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang, "When machine unlearning jeopardizes privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 896–911.

[46] A. Kolluri, T. Baluta, B. Hooi, and P. Saxena, "LPGNet: Link private graph networks for node classification," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 1813–1827.

[47] B. Wu et al., "Graphguard: Detecting and counteracting training data misuse in graph neural networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024.

[48] X. He, R. Wen, Y. Wu, M. Backes, Y. Shen, and Y. Zhang, "Node-level membership inference attacks against graph neural networks," 2021,*arXiv:2102.05429*.

[49] D. DeFazio and A. Ramesh, "Adversarial model extraction on graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2019.

[50] B. Wu, X. Yang, S. Pan, and X. Yuan, "Model extraction attacks on graph neural networks: Taxonomy and realisation," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2022, pp. 337–350.

[51] Y. Shen, X. He, Y. Han, and Y. Zhang, "Model stealing attacks against inductive graph neural networks," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 1175–1192.

[52] Y. Zhuang et al., "Unveiling the secrets without data: Can graph neural networks be exploited through data-free model extraction attacks," in *Proc. USENIX Secur.*, 2024, pp. 5251–5268.

[53] Y. Li, S. Zhang, G. Zhang, and D. Cheng, "Community-centric graph unlearning," in *Proc. AAAI Conf. Artif. Intell.*, 2025, pp. 18548–18556.

[54] C. Zhang, W. Wang, Z. Tian, and S. Yu, "Forgetting and remembering are both you need: Balanced graph structure unlearning," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 6751–6763, 2024.

[55] H. Zhang, B. Wu, X. Yang, X. Yuan, X. Liu, and X. Yi, "Dynamic graph unlearning: A general and efficient post-processing method via gradient transformation," in *Proc. ACM Web Conf.*, 2025, pp. 931–944.

[56] J. Liang, Z. Chen, Q. Luo, Z. Xie, G. Liu, and Z. Cai, "Hypergraph unlearning: A size-based hyperedge selection and coverage aggregation approach," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 6651–6661, 2025.

[57] E. Chien, C. Pan, and O. Milenkovic, "Efficient model updates for approximate unlearning of graph-structured data," in *Proc. Int. Conf. Learn. Representations*, 2022.

[58] K. Wu, J. Shen, Y. Ning, and W. H. Wang, "Fast yet effective graph unlearning through influence analysis," 2022. [Online]. Available: https://openreview.net/forum?id=er_nz4Q9Km7

[59] W. Cong and M. Mahdavi, "Efficiently forgetting what you have learned in graph representation learning via projection," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2023, pp. 6674–6703.

[60] G. Liu, X. Ma, Y. Yang, C. Wang, and J. Liu, "Federaser: Enabling efficient client-level data removal from federated learning models," in *Proc. IEEE/ACM Int. Symp. Qual. Serv.*, 2021, pp. 1–10.

[61] X. Cao, J. Jia, Z. Zhang, and N. Z. Gong, "Fedrecover: Recovering from poisoning attacks in federated learning using historical information," in *Proc. IEEE Symp. Secur. Privacy*, 2023, pp. 1366–1383.

[62] J. Nocedal, "Updating quasi-newton matrices with limited storage," *Math. Computation*, vol. 35, no. 151, pp. 773–782, 1980.

[63] X. Guo et al., "FAST: Adopting federated unlearning to eliminating malicious terminals at server side," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 2, pp. 2289–2302, Mar./Apr. 2024.

[64] C. Wu, S. Zhu, and P. Mitra, "Federated unlearning with knowledge distillation," 2022, *arXiv:2201.09441*.

[65] Y. Zhao, P. Wang, H. Qi, J. Huang, Z. Wei, and Q. Zhang, "Federated unlearning with momentum degradation," *IEEE Internet Things J.*, vol. 11, no. 5, pp. 8860–8870, Mar. 2024.

[66] W. Wang et al., "Label inference attacks against federated unlearning," in *Proc. Int. Conf. Knowl. Sci., Eng. Manage.*, 2025, pp. 1–16.

[67] X. Tang et al., "Finback: Infiltrating backdoors into gradient compressors on federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 12460–12475, 2025.

[68] X. Tang, M. Shen, Q. Li, L. Zhu, T. Xue, and Q. Qu, "PILE: Robust privacy-preserving federated learning via verifiable perturbations," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 6, pp. 5005–5023, Nov./Dec. 2023.

[69] X. Tang et al., "ROBY: A Byzantine-robust and privacy-preserving serverless federated learning framework," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 7824–7838, 2025.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:21:33 UTC from IEEE Xplore. Restrictions apply.