Certified Edge Unlearning for Graph Neural Networks

Kun Wu
kwu14@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

Jie Shen
jie.shen@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

Yue Ning
yue.ning@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

Ting Wang
inbox.ting@gmail.com
Pennsylvania State University
University Park, Pennsylvania, USA

Wendy Hui Wang
hui.wang@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

ABSTRACT

The emergence of evolving data privacy policies and regulations has sparked a growing interest in the concept of "machine unlearning", which involves enabling machine learning models to forget specific data instances. In this paper, we specifically focus on *edge unlearning* in Graph Neural Networks (GNNs), which entails training a new GNN model as if certain specified edges never existed in the original training graph. Unlike conventional unlearning scenarios where data samples are treated as independent entities, edges in graphs exhibit correlation. Failing to carefully account for this data dependency would result in an incomplete removal of the requested data from the model. While retraining the model from scratch by excluding the specific edges can eliminate their influence, this approach incurs a high computational cost. To overcome this challenge, we introduce CEU, a Certified Edge Unlearning framework. CEU expedites the unlearning process by updating the parameters of the pre-trained GNN model in a single step, ensuring that the update removes the influence of the removed edges from the model. We formally prove that CEU offers a rigorous theoretical guarantee under the assumption of convexity on the loss function. Our empirical analysis further demonstrates the effectiveness and efficiency of CEU for both linear and deep GNNs – it achieves significant speedup gains compared to retraining and existing unlearning methods, while maintaining comparable model accuracy to retraining from scratch.

CCS CONCEPTS

• **Security and privacy** → **Privacy protections**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Graph unlearning, Graph Neural Networks, trustworthy machine learning

KDD '23, August 6–10, 2023, Long Beach, CA, USA.

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599271

Wendy:There should be no privacy policies in CCS Concepts. Only Privacy protections and Machine learning are sufficient.Kun:fixed

1 INTRODUCTION

Legislation such as the General Data Protection Regulation (GDPR)[32], the California Consumer Privacy Act (CCPA)[28], and the Personal Information Protection and Electronic Documents Act (PIPEDA)[29] has introduced requirements for companies to honor user requests for the removal of private data. This has sparked discussions around the concept of the "right to be forgotten" [23], which empowers users to have more control over their data by requesting its deletion from learned models. When a company has already utilized user data to train their machine learning (ML) models, these models must be appropriately manipulated to reflect data deletion requests.

In this paper, our focus lies on Graph Neural Networks (GNNs) as the target model and the removal of edges as the unlearning request. To illustrate this scenario, let's consider an online social network platform where users request the elimination of their sensitive social relations. The platform owner is legally bound to remove the edges associated with these sensitive social relations from any GNN model trained on the graph containing those edges. This ensures that the model no longer "remembers" those sensitive social relations.

Naively erasing edges from a GNN model by fully retraining can be excessively time-consuming, particularly for complex GNN models trained on large training graphs. As a result, recent efforts have focused on developing efficient methods for exact unlearning [8, 11] as well as approximate unlearning [10, 27] specifically tailored for GNNs. In this paper, our emphasis is on approximate graph unlearning methods that facilitate the removal of requested edges from the model without retraining from scratch. Our approach is inspired by the concept of *influence function*, which enables the estimation of the impact of individual data samples on learning models [22]. To prove that the resulting model has removed the information related to the deleted edges, our goal is to provide a rigorous certified guarantee [16, 17] of the statistical indistinguishability between the retrained model and the resulting model after the unlearning process.

Despite the plethora of research on machine unlearning for non-graph datasets (e.g., [4–6, 24]), none of these approaches can be directly applied to GNNs due to the presence of data dependency within graphs. Failing to carefully account for this data dependency would result in an incomplete removal of the requested data from

the model. While recent efforts have been made to develop exact and approximate edge unlearning methods for GNNs [8, 10, 11], exact unlearning methods suffer from potentially significant loss of model accuracy [8, 11]. On the other hand, the existing approximate unlearning methods either lack a certified guarantee [9] or are limited to GNN models with specific structures [10] [1].

Our contributions. In this paper, we design CEU, a Certified Edge Unlearning method designed to remove requested edges from GNNs without retraining from scratch while providing a provable guarantee of the unlearning model. Our contributions are outlined as follows.

► **Unlearning through influence analysis.** We formulate the unlearning problem as finding a closed-form update on the model parameters. To achieve this, we introduce a novel influence function that efficiently computes the necessary update, while also taking into account the neighborhood of the removed edges. We address several theoretical and practical challenges of deriving edge influence by providing an influence estimator that is computationally and memory efficient.

► **Certified unlearning.** We undertake in-depth theoretical analysis and present non-trivial findings. We provide formal proofs demonstrating that CEU can deliver a rigorous (ϵ, δ)-approximation guarantee under the assumption of a strongly convex loss function. Additionally, we derive both worst-case and data-dependent bounds for the statistical distance between the retrained model and the model obtained through unlearning using CEU.

► **Empirical analysis.** Through extensive empirical study, we showcase the efficiency and effectiveness of CEU for both linear and deep GNN models. Specifically, for linear GNNs, we demonstrate that CEU achieves effective unlearning with a remarkable 16.2-fold speedup compared to retraining from scratch. Notably, our method outperforms the exact graph unlearning approach [8] in terms of both model accuracy and unlearning efficiency, exhibiting a 63% improvement in model accuracy and a 3.7-fold speedup. Additionally, it surpasses the existing certified graph unlearning method [10] in terms of unlearning efficiency, achieving a speedup of at least two orders of magnitude. Moving on to deep GNNs, our empirical results highlight the high efficiency of CEU, providing a speedup of up to 5 times compared to retraining while maintaining similar model accuracy. Furthermore, we quantitatively assess the efficacy of unlearning by conducting a link membership inference attack [20] on unlearning models. We demonstrate that the attack accuracy of inferring the removed edges from the unlearning model is comparable to that from the retrained model, indicating the successful removal of the targeted edges.

2 RELATED WORK

Machine unlearning [2, 18, 26] refers to a process that aims to remove the impact of a set of data samples in the training data from a trained model. From the certainty of unlearning, existing methods of machine unlearning can be divided into *exact unlearning* and *approximate unlearning*.

[1] The theoretical analysis and algorithmic techniques of [10] are closely tied to linear GNNs such as simple graph convolutions (SGC) and their generalized PageRank (GPR) extensions.

Exact machine unlearning. In exact unlearning, a model is naively retrained from scratch after removing certain data samples from the dataset. This is generally computationally expensive. Several attempts have been made to make unlearning more efficient than retraining from scratch. An earlier study converts ML algorithms to statistical query (SQ) learning so that unlearning processes only need to retrain the summation of SQ learning [5]. The *SISA* (sharded, isolated, sliced, and aggregated) approach [2] trains a set of constituent models on disjoint data *shards*. Given an unlearning request, only the affected shards and their constituent models are retrained.

Some recent works [8, 11] extend exact unlearning to the graph setting. *GraphEraser* [8] adapts the SISA approach to graph unlearning. It splits graphs into disjoint partitions. Upon receiving an unlearning request, only the model on the affected shards is retrained. However, as shown in our empirical studies later (Section 6), GraphEraser suffers from a significant loss of model accuracy, as splitting the training graph into disjoint partitions damages the original graph structure. *GraphEditor* [11] designs an exact unlearning solution of linear GNNs. However, it is restricted to the linear structure only. It also cannot deal with efficient batch removal of a large number of edges.

Approximate machine unlearning. Approximate unlearning relaxes the requirement for exact unlearning by requiring that the removed data is statistically unlearned with the guarantee that the unlearned model cannot be distinguished from an exact deletion model [17], where the indistinguishability is defined in a similar manner as differential privacy [13]. Certified unlearning can be realized by adding noise either on the weights [15, 16, 26, 33, 41] or on the loss function [17]. In the context of graph unlearning, Chien *et al.* [10] provide the first certified GNN unlearning solution. However, their approach is restricted to GNN models of certain structures such as Simple Graph Convolution (SGC) and its generalized PageRank (GPR). Their implementation cannot be easily adapted to general GNNs. Furthermore, their approach cannot support batch edge removal. Our empirical results show that CEU is much faster than [10] in batch edge unlearning, with a speed-up of at least two orders of magnitude. Their follow-up work [27] extends to a particular type of nonlinear GNN models based on Graph Scattering Transform (GST). However, [27] considers node unlearning not edge unlearning. On the other hand, the approximate edge unlearning solution proposed by Cheng *et al.* [9] cannot provide any certified guarantee.

3 PROBLEM FORMULATION

Before presenting the technical details of our unlearning solution, let us describe the problem setting along with the definition of certified guarantee first.

Problem setup. Let $\mathcal{G}$ be a set of graphs. In this paper, we only consider undirected graphs. Let Θ be the parameter space of GNN models. A learning algorithm $\mathcal{A}_{\mathrm{L}}$ is a function that maps an instance $G(V, E) \in \mathcal{G}$ to a parameter $\theta \in \Theta$. Let θ_{OR} be the parameters of $\mathcal{A}_{\mathrm{L}}$ trained on G. Any user can submit an edge unlearning request to remove specific edges from G. In practice, unlearning requests are often submitted sequentially. For efficiency, we assume these requests are processed in a batch. Let E_{UL} denote the batch

of edges that are requested to be removed. As a response to these requests, $\mathcal{A}_{\mathrm{L}}$ has to erase the impacts of E_{UL} on $\mathcal{A}_{\mathrm{L}}$ and produce an unlearned model. A straightforward approach is to retrain the model on $G(V, E\backslash E_{\mathrm{UL}})$ from scratch and obtain the model parameters θ_{RE}. However, due to the high computational cost of retraining, an alternative solution is to apply an *unlearning* process $\mathcal{A}_{\mathrm{UL}}$ that takes E_{UL} and θ_{OR} as input and outputs an *unlearned model*.

Certified guarantee. Approximate unlearning requires some format of guarantee that the information related to the deleted data has been removed from the model. Intuitively, if the result of unlearning is likely to be obtained by retraining, then the unlearning algorithm has successfully eliminated the influence of the removed data points from the model. Following this intuition, we adapt the concept of *certified removal* [17, 26] to our setting to measure the difference between the retrained model and one obtained by unlearning. Broadly speaking, certified removal defines the indistinguishability between the retrained model and the unlearned model in a similar manner as (ϵ, δ)-differential privacy [13]. In particular, it defines the notion of (ϵ, δ)-*approximate unlearning* which is formalized as follows.

Definition 1 ((ϵ, δ)-**Approximate Unlearning**). *Given a learning algorithm $\mathcal{A}_{\mathrm{L}}$ and two constants $\epsilon, \delta > 0$, an unlearning algorithm $\mathcal{A}_{\mathrm{UL}}$ performs (ϵ, δ)-certified unlearning for $\mathcal{A}_{\mathrm{L}}$ if*

$$P\big(\mathcal{A}_{\mathrm{UL}}(D, z, \mathcal{A}_{\mathrm{L}}(D))\big) \leq e^{\epsilon} P\big(\mathcal{A}_{\mathrm{L}}(D\backslash z)\big) + \delta, \tag{1}$$

and

$$P\big(\mathcal{A}_{\mathrm{L}}(D\backslash z)\big) \leq e^{\epsilon} P\big(\mathcal{A}_{\mathrm{UL}}(D, z, \mathcal{A}_{\mathrm{L}}(D))\big) + \delta, \tag{2}$$

where z is the sample to be removed.

Intuitively, Def. 1 guarantees that the unlearned model is "approximately" the same as the retrained model, where the difference between the unlearned and retrained model is bounded by the parameters of ϵ and δ. Smaller ϵ and δ indicate that the unlearned model is closer to the retrained model. Trivially, a $(0, 0)$-approximate unlearning model is equivalent to the retrained model.

We adapt the notion (ϵ, δ)-approximate unlearning to edge removal, and formalize the edge unlearning problem as follows:

Definition 2 ((ϵ, δ)-**approximate Edge Unlearning**). *Given a graph $G(V, E)$, a set of edges $E_{\mathrm{UL}} \subset E$ that are requested to be removed from G, a graph learning algorithm $\mathcal{A}_{\mathrm{L}}$ and its readout function f, then an edge unlearning algorithm $\mathcal{A}_{\mathrm{UL}}$ performs (ϵ, δ)-certified unlearning for $\mathcal{A}_{\mathrm{L}}$ if:*

$$P\big(\mathcal{A}_{\mathrm{UL}}(G, E_{\mathrm{UL}}, \mathcal{A}_{\mathrm{L}}(G))\big) \leq e^{\epsilon} P\big(\mathcal{A}_{\mathrm{L}}(G_{\mathrm{UL}})\big) + \delta, \tag{3}$$

and

$$P\big(\mathcal{A}_{\mathrm{L}}(G_{\mathrm{UL}})\big) \leq e^{\epsilon} P\big(\mathcal{A}_{\mathrm{UL}}(G, E_{\mathrm{UL}}, \mathcal{A}_{\mathrm{L}}(G))\big) + \delta, \tag{4}$$

where $\epsilon, \delta > 0$, and $G_{\mathrm{UL}} = G(V, E\backslash E_{\mathrm{UL}})$.

While Def. 1 is defined for removing a single data sample, we extend it to the removal of a set of samples (edges) to handle batch edge removal. Our goal is to seek the unlearning mechanism $\mathcal{A}_{\mathrm{UL}}$ that can remove multiple edges at once with (ϵ, δ)-approximate guarantee while its computational complexity is significantly cheaper than retraining.

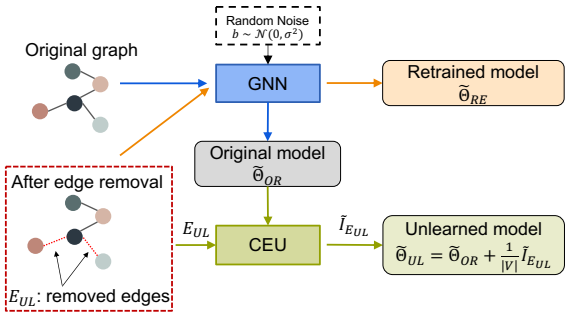


Figure 1: Framework of CEU. Orange lines indicate the process of retraining and green lines indicate unlearning.

4 METHODOLOGY

Given a graph $G(V, E)$ as input, we can find a model represented by θ that fits the data by minimizing an empirical loss. In this paper, we consider cross-entropy loss [12] for node classification as our loss function. The **or**iginal model θ_{OR} is optimized by the following:

$$\theta_{\mathrm{OR}} = \arg\min_{\theta} \frac{1}{|V|} \sum_{v \in V} \mathcal{L}(\theta; v, E). \tag{5}$$

Assume a set of edges E_{UL} is deleted from G and let the new graph after the deletion be represented as $G_{\mathrm{UL}} = G(V, E\backslash E_{\mathrm{UL}})$, **re**training the model will obtain a new model parameter θ_{RE} on G_{UL}:

$$\theta_{\mathrm{RE}} = \arg\min_{\theta} \frac{1}{|V|} \sum_{v \in V} \mathcal{L}(\theta; v, E\backslash E_{\mathrm{UL}}). \tag{6}$$

A major difficulty, as expected, is that obtaining θ_{RE} is prohibitively slow for complex networks and large datasets. To overcome this challenge, we will identify a closed-form update $I_{E_{\mathrm{UL}}}$ to θ_{OR}:

$$\theta_{\mathrm{UL}} \approx \theta_{\mathrm{OR}} - I_{E_{\mathrm{UL}}}, \tag{7}$$

where $I_{E_{\mathrm{UL}}}$ has the same dimension as the learning model θ_{OR}. Intuitively, θ_{UL} approximates the retraining. Such approximation, however, may not be able to provide any unlearning guarantee, as the direction of the gradient residual of θ_{UL} may still be able to leak information about the removed edges.

Overview of CEU. We design CEU as a two-step process. In Step 1, CEU adds the perturbation to the loss function, aiming to hide the real gradient residual and provide the certified unlearning guarantee. Let $\tilde{\theta}_{\mathrm{OR}}$ be the parameters of the model trained with the noisy loss function. In Step 2, CEU estimates the one-shot update on the parameters $\tilde{\theta}_{\mathrm{OR}}$ through influence analysis. Figure 1 illustrates an overview of CEU. Next, we describe the details of the two steps.

4.1 Step 1: Adding Perturbation on Loss Function

To enable unlearning with a certified guarantee, we follow the same idea of certified data removal [17] and add a linear noise term to the training loss, aiming to hide the real gradient residual. We use $\mathcal{L}_b$ to denote the loss function with noise, which is formalized as follows:

$$\mathcal{L}_b = L(\theta, E) + \frac{\lambda}{2}||\theta||^2 + b^{\intercal}\theta, \tag{8}$$

where b is drawn randomly from a given distribution $\mathcal{N}(0, \sigma^2)$. The randomness in b will mask any potential information leaked by the estimated edge influence. The resulting perturbed learning problem can be solved using standard convex optimization methods.

4.2 Step 2: Unlearning through Influence Analysis

Intuitively, updating model parameters for unlearning can be interpreted from the optimization perspective that the model forgets E_{UL} by "reversing" the influence $\tilde{\theta}_{\text{UL}}$ of E_{UL} from the model. The challenge is how to estimate the influence of $\tilde{\theta}_{\text{UL}}$ on the model.

Influence functions [22] enable efficient approximation of the effect of some particular training points on a model's prediction. Intuitively, the influence function computes the parameters after the removal of z by upweighting z on the parameters with some small ζ:

$$\hat{\theta}_{\zeta,z} = \arg\min_{\theta} \frac{1}{m} \sum_{z_i \neq z} \mathcal{L}(\theta; z_i) + \zeta \mathcal{L}(\theta; z), \tag{9}$$

where m is the number of data points in the original dataset, and ζ is a small constant. The influence function is not restricted to a single point. We can define a set of points Z and compute $\hat{\theta}_{\zeta,Z}$.

However, most of the existing influence functions cannot be directly applied to the GNN setting, as removing one edge $e(v_i, v_j)$ from the graph can affect not only the prediction of v_i and v_j but also those of neighboring nodes of v_i and v_j, due to the aggregation function of GNN models. To address this challenge, we design a new influence function for GNNs that take the neighborhood into consideration when estimating the influence of the neighborhood of removing an edge on model parameters.

In general, an ℓ-layer GNN aggregates the information of the ℓ-hop neighborhood of each node. Thus removing an edge $e(v_i, v_j)$ will affect not only v_i and v_j but also all nodes in the ℓ-hop neighborhood of v_i and v_j. To capture such aggregation effect in the derivation of edge influence, first, we define the set of nodes (denoted as V_e) that will be affected by removing an edge $e(v_i, v_j)$ as: $V_e = \mathcal{N}(v_i) \cup \mathcal{N}(v_j) \cup \{v_i, v_j\}$, where $\mathcal{N}(v)$ is the set of nodes connected to v in ℓ hops.

$$V_{E_{\text{UL}}} = \bigcup_{e \in E_{\text{UL}}} V_e. \tag{10}$$

To revert the *influence* of E_{UL} on the target model following Eqn. (9), we compute the new parameters $\theta_{\zeta,E_{\text{UL}}}$ after the removal of E_{UL} as follows:

$$\theta_{\zeta,V_{E_{\text{UL}}}} = \arg\min_{\theta} \frac{1}{|V|} \sum_{v \in V} \mathcal{L}_b(\theta; v, E) + \zeta \Big(\sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E \backslash E_{\text{UL}}) - \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E) \Big). \tag{11}$$

At a high level, Eqn. (11) contains three terms. While the first one measures the loss of the original model, the second and the third ones together compute the loss of the nodes which are affected by the removal of E_{UL}. Following this reasoning, Eqn. (11) is equivalent to Eqn. (6) when $\zeta = \frac{1}{|V|}$, where $|V|$ is the total number of nodes (more details can be found in Appendix A.1). In other words, removing an edge is equivalent to upweighting it by $\zeta = \frac{1}{|V|}$. Following this reasoning, instead of solving the problem in Eqn. (11), we formulate the optimization as a closed-form update on the original model $\tilde{\theta}_{\text{OR}}$:

$$\tilde{\theta}_{\text{UL}} = \tilde{\theta}_{\text{OR}} + \frac{1}{|V|} \tilde{I}_{E_{\text{UL}}}, \tag{12}$$

where $\tilde{I}_{E_{\text{UL}}}$ is the *influence* of E_{UL} on the target model with noisy loss. By utilizing this formulation, we can describe changes of the training graph structure by edge removal as a one-shot update on model parameters.

In this paper, we take a *second-order* update strategy that utilizes second-order derivatives to calculate the closed-form update $\tilde{I}_{E_{\text{UL}}}$. Our second-order update result is present in the following theorem.

Theorem 3. *Given the parameters θ_{OR} obtained by $\mathcal{A}_{\text{UL}}$ on a graph G, and the loss function $\mathcal{L}$, assume that $\mathcal{L}$ is twice-differentiable and convex in θ, then the influence of a set of edges E_{UL} is:*

$$\tilde{I}_{E_{\text{UL}}} = -H^{-1}_{\tilde{\theta}_{\text{OR}}} \Big(\nabla_\theta \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta_{\text{OR}}; v, E \backslash E_{\text{UL}}) - \nabla_\theta \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E) \Big), \tag{13}$$

where $H_{\tilde{\theta}_{\text{OR}}} := \nabla^2 \frac{1}{|V|} \sum_{v \in V} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}, v, E)$, and $H^{-1}_{\tilde{\theta}_{\text{OR}}}$ is the inverse Hessian of the loss at $\tilde{\theta}_{\text{OR}}$.

A full derivation of the second-order update (Eqn. (13)) is provided in Appendix A.2.

Theorem 3 assumes the loss function is convex. Given the non-convexity nature of GNN models, the Hessian matrix can be non-invertible and thus there may not have a solution for the influence estimation. To address this issue, we follow [22] and add a damping term λ_1 to $H_{\tilde{\theta}_{\text{OR}}}$ (i.e., $(H_{\tilde{\theta}_{\text{OR}}} + \lambda_1 I)$) if $H_{\tilde{\theta}_{\text{OR}}}$ has negative eigenvalues, where λ_1 is the same as the regularization rate λ in Eqn. (8). This can be implemented by adding L_2 regularization on the parameters. Our empirical analysis (Sec. 6) will show this solution enables effective unlearning in practice.

Designing an influence estimator with time and memory efficiency. There are several practical and theoretical challenges in calculating the influence (Eqn. (13)). First, for large graphs, even storing a Hessian matrix in memory is expensive: in our experiments, we will show that Hessian matrices are huge, e.g. the Hessian matrix on the CS dataset has a size of around $10^5 \times 10^5$ which would cost 50 GB memory. Second, even under the promise that the linear system is feasible, computing the inverse of a matrix of huge size is prohibitive. To address these two challenges, we design an algorithm that approximates the inverse Hessian. Note that the existing certified graph unlearning method [10] did not use any influence estimator. Instead, it computes the exact inverse Hessian.

The starting point of our algorithm is a novel perspective that solving the linear system (Eqn. (13)) can be thought of as finding a *stationary point* of the following quadratic function:

$$f(x) = \arg\min_{x} \frac{1}{2} x^T B x - k^T x,$$

where $B = \tilde{H}_{\theta_{\text{OR}}}$ and

$$k = \nabla_\theta \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{OR}; v, E \backslash E_{\text{UL}}) - \nabla_\theta \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{OR}; v, E). \tag{14}$$

The random noise b does not appear in B by taking the second order derivation, and b does not appear in k as both terms in Eqn. (14) contains b and is canceled out. Note that even the function $f(x)$ is non-convex, there is rich literature establishing convergence guarantee to stationary points using gradient-descent-type algorithms; see e.g. [1].

We employ the implementation [25] that combines Hessian-vector product (HVP) [30] and the conjugate gradient (CG) [36] to approximate the inverse Hessian. CG exhibits promising computational efficiency for minimizing quadratic functions [31]. It is well-known that as long as the step size satisfies the Wolfe conditions [38, 39] and the objective function is Lipschitz and bounded from below, the sequence of iterates produced by CG asymptotically converges to a stationary point of $f(x)$, which corresponds to a solution $\tilde{I}_{E_{\mathrm{UL}}}$ that satisfies Eqn. (13). Note that these regularity conditions are satisfied when the training data are bounded. Hence, we have the following convergence guarantee of influence estimation.

Lemma 4 (Theorem 2.1 of [31]). *The CG method generates a sequence of iterates $\{x_t\}_{t\geq 1}$ such that $\lim_{t\to+\infty} f(x_t) = 0$. In addition, the per-iteration time complexity is $O(|x|)$ where $|x|$ denotes the dimension of x.*

We note, however, that an appealing feature of Eqn. (13) is that it does not need to find a solution with an exact-zero gradient. This enables us to terminate CG early by monitoring the magnitude of the gradients. Our empirical study also shows that CG can get a good approximation in a small number of iterations.

Besides time efficiency, we have the following lemma showing that the CG method is memory-efficient.

Lemma 5. *The CG method can be implemented using $O(|\theta|)$ memory.*

The proof of Lemma 5 can be found in Appendix A.3.

5 CERTIFIED UNLEARNING GUARANTEE

As the unlearned model by CEU approximates the retrained model, ideally it should provide the theoretical guarantee that the unlearned model is statistically indistinguishable from the retrained one. Next, we derive conditions under which the second-order update by CEU can provide the (ϵ, δ)-approximate unlearning guarantee. To construct theoretical guarantees for our approach, we make the following assumptions on the GNN models.

Assumption 6. *For the given GNN model and its loss function L: (1) L is a strictly convex loss function that is twice differentiable; (2) $||\nabla L||_2 \leq c_1$; (3) $\nabla^2 L$ is γ_1-Lipschitz; (4) ∇L is γ_2-Lipschitz; and (5) the node features x_v is bounded: $||x_v||_2 \leq 1, \forall v \in V$. Here c_1, γ_1, γ_2 are positive constants.*

These assumptions can be satisfied by a wide range of GNN learning models such as Simple Graph Convolution (SGC) [7, 40] and Graph Linear Network (GLN) [37] which can achieve the comparable performance compared with deep GNNs [14, 37, 40]. It is important to note that, although our theoretical analysis relies on the assumption of strictly convex loss function, our algorithmic techniques are generic and can be applied to various GNN models. We will show that CEUcan achieve notable empirical performance on both linear and deep GNNs (Section 6).

The existing certified unlearning works utilize the gradient residual $||\nabla \mathcal{L}||_2$ for analysis of the unlearning model. Intuitively, for strongly convex loss functions (as assumed), the gradient residual is zero as the optimum is unique. Hence, the norm of the gradient residual $||\nabla \mathcal{L}||_2$ can reflect the distance between the retrained and the unlearned models. Based on the gradient residual norm, we show that CEU can establish the (ϵ, δ)-approximation guarantee by the following Theorem 7.

Theorem 7 (Theorem 3 from [17]). *Let $\mathcal{A}_{\mathrm{L}}$ be the learning algorithm that returns the unique optimum of the loss $\mathcal{L}_b$ and let $\mathcal{A}_{\mathrm{UL}}$ be the unlearning mechanism. Suppose that $||\nabla \mathcal{L}_b||_2 \leq \epsilon'$ for some computable bound $\epsilon' > 0$. If $b \sim \mathcal{N}(0, c\epsilon'/\epsilon)^d$ with some constants $c, \epsilon > 0$, where d is the parameter dimension, then $\mathcal{A}_{\mathrm{UL}}$ provides (ϵ, δ)-approximation guarantee for $\mathcal{A}_{\mathrm{L}}$, where $\delta = 1.5e^{-c^2/2}$.*

Theorem 7 requires the *gradient residual norm* $||\nabla \mathcal{L}_b||_2$ to be bounded appropriately in order to provide the approximation guarantee. First, we present the worst-case bound of $||\nabla \mathcal{L}_b||_2$ in Theorem 8.

Theorem 8 (**Worst-case Bound**). *Assume Assumption 6 holds. Then*

$$||\nabla \mathcal{L}_b(\tilde{\theta}_{UL}, E\backslash E_{\mathrm{UL}})||_2 \leq \frac{\gamma_1 \gamma_2^2 c_1^2}{\lambda^4 |V|} \Big(\sum_{v \in E_{\mathrm{UL}}} n_v \Big)^2, \tag{15}$$

where n_v denotes the number of neighbors of node v, λ is the regularization rate (Eqn. 8), and $|V|$ is the number of nodes in the training graph.

The proof of Theorem 8 is provided in Appendix A.4. As $\frac{1}{\lambda^4}$ in Theorem 8 can be large, the worst-case bound can be impractically loose. Therefore, next, we derive the *data-dependent* bound on the gradient residual norm in the following theorem.

Theorem 9 (**Data-dependent Bound**). *Suppose Assumption 6 holds. Then*

$$||\nabla \mathcal{L}_b(\tilde{\theta}_{\mathrm{UL}}, E\backslash E_{\mathrm{UL}})||_2 \leq \gamma_1 \frac{1}{|V|^2} ||\tilde{H}^{-1}_{\theta_{\mathrm{OR}}} \Delta||_2^2, \tag{16}$$

where $|V|$ is the number of nodes in the graph and

$$\Delta = \nabla_\theta \sum_{v \in V_{E_{\mathrm{UL}}}} \mathcal{L}(\tilde{\theta}_{\mathrm{OR}}; v, E) - \nabla_\theta \sum_{v \in V_{E_{\mathrm{UL}}}} \mathcal{L}(\tilde{\theta}_{\mathrm{OR}}; v, E\backslash E_{\mathrm{UL}}).$$

The proof of Theorem 9 can be found in Appendix A.5. The data-dependent bound (Eqn. (16)) can be computed efficiently by using the efficient influence estimator present in Sec. 4.2. We will show that the data-dependent bound is much tighter than the worst-case bound (Theorem 8) in our empirical evaluation (Sec. 6).

CEU only can guarantee certified unlearning for strongly convex loss functions that have Lipschitz-continuous gradients. While it performs well even for deep GNNs, as we will demonstrate in the empirical evaluation, it requires additional measures to verify the effectiveness of edge removal. In our empirical analysis, we evaluate the closeness between the unlearned model and the retrained one by comparing their model accuracy. We also use the *membership inference attacks* [20] as the external measurement tool to evaluate the extent to which the unlearned model has forgotten the removed edges. More details of our evaluation can be found in Section 6.

6 EXPERIMENTS

In this section, we empirically verify the efficiency and effectiveness of CEU. The code and datasets are publicly available [2].**Kun:I updated the link. But I still need to clean the code. It is a little messy right now.**

6.1 Experimental Setup

All experiments are executed on a GPU server with NVIDIA A100 (40G). All the algorithms are implemented in Python with PyTorch. Each experiment is repeated 10 times and the average is reported.

Datasets. We use three well-known datasets, namely **Cora** [34], **Citeseer** [43], and **CS** [35] datasets, that are popularly used for performance evaluation of GNNs [35, 44]. The statistical information of these datasets can be found in Appendix B.

GNN models. We consider two types of settings of GNN models:

- **Linear models**: We consider a simplified GCN model that contains only one layer and a softmax function (without normalization). The proof that the 1-layer GNN model has a strongly convex loss function is included in Appendix A.6.
- **Deep models**: We consider three representative GNN models, namely **GCN** [21], **GraphSAGE** [19], and **GIN** [42]. For these GNN models, we consider various network complexity (up to four hidden layers) in the experiments, with the same number of neurons as 32 at each layer respectively. All GNN models are trained for 1,000 epochs with an early stop condition that the validation loss does not decrease for 20 epochs.

We randomly split each graph into a training set (70%), a validation set (10%), and a test set (20%). More details of the setup of model parameters can be found in Appendix B.

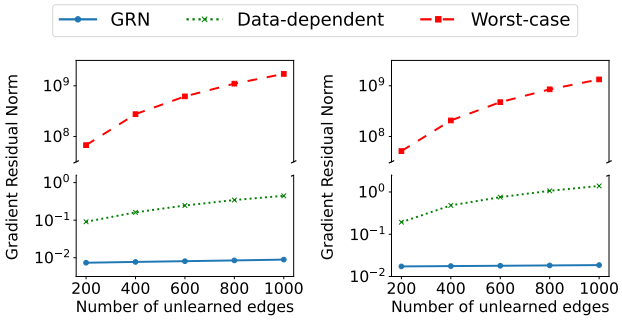
Edges for removal. We randomly pick k = {200, 400, 600, 800, 1,000}) edges from Cora and CiteSeer datasets, and k = {2,000, 4,000, 6,000, 8,000, 10,000}) edges from CS dataset for removal. We pick more edges from the CS dataset as its number of edges is orders of magnitude higher than the other two datasets (Table 4).

Metrics. We evaluate the performance of CEU in terms of *efficiency*, *efficacy*, and *model accuracy*: (1) **Unlearning efficiency**: we measure the running time of CEU and retraining time for a given set of edges; (2) **Target model accuracy**: we measure *accuracy* of node classification, i.e., the percentage of nodes that are correctly classified by the model, as the accuracy of the target model. Higher accuracy indicates better accuracy retained by the unlearned model; (3) **Unlearning efficacy**: We utilize a black-box edge membership inference attack (MIA) named *StealLink* [20] to empirically evaluate the extent to which the unlearned model has forgotten the removed edges.[3] StealLink predicts whether particular edges exist in the training graph. We measure AUC as the accuracy of StealLink of inferring whether the removed edges were present in the original graph. Intuitively, a higher AUC indicates higher attack accuracy. AUC close to 0.5 indicates that the adversary's belief of these edges in the original training graph is close to a random guess.

Noise setup. We follow the setting of [10] and set $\lambda = 0.01$ and $\sigma = 0.1$ (Eqn. (8)). We use the same ϵ (ϵ = 0.1 - 10) as in [10].

[2]https://github.com/kunwu522/certified_edge_unlearning

[3]Implementation of StealLink: https://github.com/xinleihe/link_stealing_attack



(a) Cora (b) CiteSeer

Figure 2: Tightness of bounds (GRN: Gradient residual norm).

Baselines. We consider three baselines of exact and approximate GNN unlearning for comparison with CEU.

- **Exact graph unlearning:** We consider GraphEraser [2], the only existing exact graph unlearning method. GraphEraser has two partitioning strategies denoted as *balanced LPA* (**BLPA**) and *balanced embedding k-means* (**BEKM**). We consider both BLPA and BEKM as the baseline methods. We use the same set of the number of shards as in [8] for both BLPA and BEKM.
- **Approximate, uncertified graph unlearning (UEU)**: We estimate the influence of the removed edges on the original model whose loss function is free of noise, and apply the one-shot update on the model parameters. More details of the influence analysis and unlearning for UEU can be found in Appendix A.7.
- **Approximate, certified graph unlearning:** We consider Certified Graph Unlearning (CGU) [10] as a baseline.[4]

Two retraining settings. As CEU adds noise to the loss function of the target model, we consider two different retraining settings (denoted as "Retrain" and "R+N" respectively). The original model is retrained under "Retrain" setting, while the model with perturbation on its loss function is retrained under "R+N" setting.

We also have additional results of the following studies: (1) the impacts of types of removed edges on unlearning performance; (2) the performance of sequential unlearning. These results and discussions can be found in Appendix C and D respectively.

6.2 Tightness of Bounds

As the certified guarantee is computed from the worst-case and data-dependent bounds of the gradient residual norm (Theorem 8 and 9), the tightness of these two bounds determines the strictness of our theoretical guarantee. Hence, we evaluate the tightness of both bounds. We consider the 1-layer GCN model and measure the real gradient residual norm values (as the ground truth) as well as the two bounds for tightness evaluation. Figure 2 reports the value of the two bounds as a function of the number of removed edges. We have two main observations. First, as expected, the worst-case bound is much looser than the data-dependent bound. It can be *several orders of magnitude* larger than the data-dependent bound. The looseness in the bound comes from $\frac{1}{\lambda^4}$ in the bound. Second, the data-dependent bound is close to the ground-truth gradient

[4]Implementation of CGU [10]: https://github.com/thupchnsky/sgc_unlearn

Table 1: Model accuracy of CEU, two retrained models (Retrain and R+N), and three baselines (BLPA, BEKM, UEU) for linear GCN and CS dataset.

Type	Method	Number of removed edges					
		0	2K	4K	6K	8K	10K
Retrain	Retrain	0.93	0.93	0.93	0.93	0.93	0.93
	R+N	0.91	0.91	0.91	0.91	0.90	0.90
Unlearn	BLPA	0.84	0.69	0.80	0.84	0.84	0.68
	BEKM	0.64	0.80	0.56	0.83	0.77	0.67
	UEU	0.93	0.93	0.93	0.93	0.93	0.93
	CEU	0.91	0.91	0.91	0.91	0.90	0.90

residual norm, regardless of the growth in the number of removed edges. Given the tightness of the data-dependent bounds, CEU is expected to handle batch removal of a large number of edges.

6.3 Performance of Linear GCN Models

In this section, we only consider linear GCN models (i.e., 1-layer GCN model), and evaluate the performance of CEU for this model on three graph datasets in terms of model accuracy, unlearning efficiency, and unlearning efficacy. The results of UEU show the impact of noise on model performance compared with CEU.

Model accuracy. Table 1 reports the results of GCN model accuracy on the CS dataset. The results on Cora and Citeseer datasets are similar and can be found in Appendix E.1. We have the following observations. First, the model accuracy obtained by CEU stays very close to that of the retrained model, regardless of the number of removed edges. The difference in model accuracy between the retrained and unlearned models remains negligible (in the range of [0.01%, 0.11%]). Second, in terms of comparison with both exact unlearning baselines (BEKM, BLPA), the model accuracy by CEU is significantly higher than these two baselines in all the settings. For example, when removing 4,000 edges, both BEKM and BLPA only can deliver model accuracy of around 0.56 and 0.80, while CEU can deliver a model accuracy of around 0.91 (63% and 14% improvement). This demonstrates the weakness of the exact unlearning through graph partitioning - breaking the graph structure can bring non-negligible model accuracy loss. Third, regarding the comparison with the approximate unlearning baseline (UEU), CEU has very similar model accuracy, although UEU does not add perturbation to the model loss function. This demonstrates that CEU addresses the trade-off between privacy and model accuracy—it can deliver a provable unlearning guarantee while requiring negligible sacrifice on model accuracy.

Unlearning efficiency. We report the time performance results of CEU in Figure 3. Our observations are followings. First, CEU is significantly faster than retraining from scratch. It speeds up by 11.4×, 6.4×, and 16.2× for Cora, CiteSeer, and CS datasets, respectively. Second, CEU is much faster than both BEKM and BLPA baselines, especially when training large graphs. For example, CEU is 3.7× faster than both BLPA and BEKM on the CS dataset when 4,000 edges and 10,000 edges were removed respectively (Figure 3 (c)). This demonstrates the advantage of the approximate unlearning methods. Third, for both approximate unlearning methods, CEU has comparable time performance as UEU although UEU is slightly faster than CEU.

Table 2: Unlearning efficacy of CEU, retraining, and the UEU baseline for linear GCN and Cora dataset.

$\lvert E_{\text{UL}}\rvert$	Original	Retrain	UEU	R+N	CEU
200	0.930	0.577	0.572	0.533	0.535
400	0.936	0.582	0.580	0.541	0.543
600	0.935	0.582	0.580	0.547	0.547
800	0.936	0.589	0.585	0.549	0.552
1,000	0.935	0.586	0.592	0.559	0.553

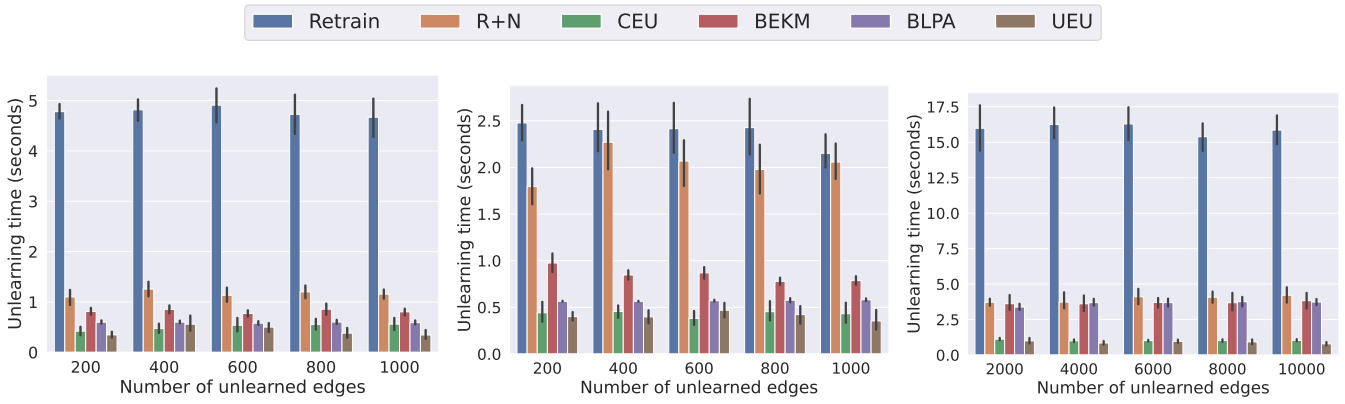
Unlearning efficacy. Table 2 reports the attack performance of attack accuracy of the removed edges E_{UL} against the original model, retraining model (with and without noise), UEU, and CEU on the Cora dataset. We observe the following phenomena. First, StealLink is highly effective to predict the existence of E_{UL} in the original graph ("Original" column), as the AUC of the attack against the original model is higher than 0.9 (much higher than 0.5). Second, the AUC of the attack is noticeably reduced to close to 0.5 for both retrained and unlearned models ("R+N" and "CEU" columns). This demonstrates that CEU has a similar ability to make the model forget the removed edges as retraining. Third, the AUC of both retraining and learning with noise ("R+N" and "CEU" columns) is lower than that without noise ("Retrain" and "UEU" columns). This demonstrates that the perturbation added to the loss function helps to reduce the privacy vulnerability of the removed edges.

Effects of ϵ on model accuracy. We study the effect of various ϵ values (for (ϵ, δ)-unlearning)) on unlearning performance. The noise b is determined by using the data-dependency bound (Theorem 9) as ϵ' and ϵ together. Figure 4 reports the model accuracy with various ϵ values. We observe that, unsurprisingly, the model accuracy degrades when ϵ grows (i.e., more noise is added). For instance, when ϵ changes from 0.1 to 10, we witness the model accuracy drops from 0.925 to 0.9 when removing 2,000 edges from the CS dataset (Figure 4 (c)). Such model accuracy drop is more significant on Cora and Citeseer datasets. The drop in model accuracy meets our expectation as higher ϵ allows a larger statistical distance between the retrained model and the unlearned model, and thus lowers the accuracy of the unlearned model.

Comparison with [10]. As the approach presented in [10] is specifically designed for Simple Graph Convolutional (SGC) models, we apply both CEU and CGU to the SGC model to ensure a fair comparison of their performance.

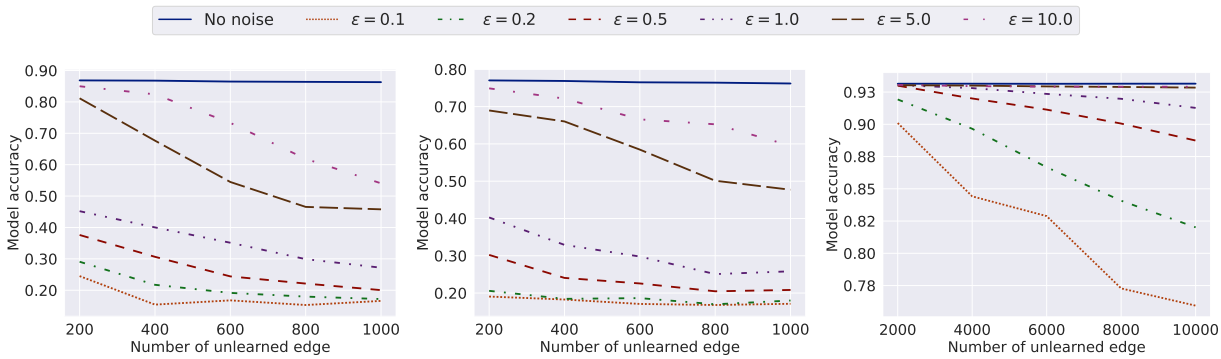
Figure 5 (a) reports the model accuracy of the retrained model and both CGU and CEU on the Cora dataset. The results on Citeseer and CS datasets are similar and can be found in Appendix E.2. We observe that the model accuracy of CEU stays close to CGU in all the settings. On the other hand, as shown in Figure 5 (b), CEU is much faster than CGU, with a speed-up by at least two orders of magnitude. Indeed, the speed-up is more compelling when more edges are removed. This shows the advantage of CEU for batch edge removal to CGU.

Besides model accuracy and unlearning efficiency, we also evaluated the unlearning efficacy of both CEU and CGU, and observed



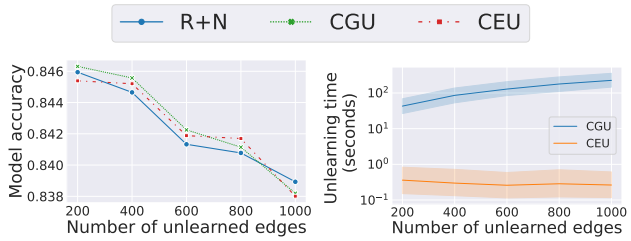
(a) Cora (b) CiteSeer (c) CS

Figure 3: Time performance of CEU, two retrained models (Retrain and R+U), and three baselines (BLPA, BEKM, UEU) for linear GCN model.



(a) Cora (b) CiteSeer (c) CS

Figure 4: Effect of unlearning parameter ϵ on model accuracy by CEU for linear GCN model.



(a) Model accuracy (b) Unlearning efficiency

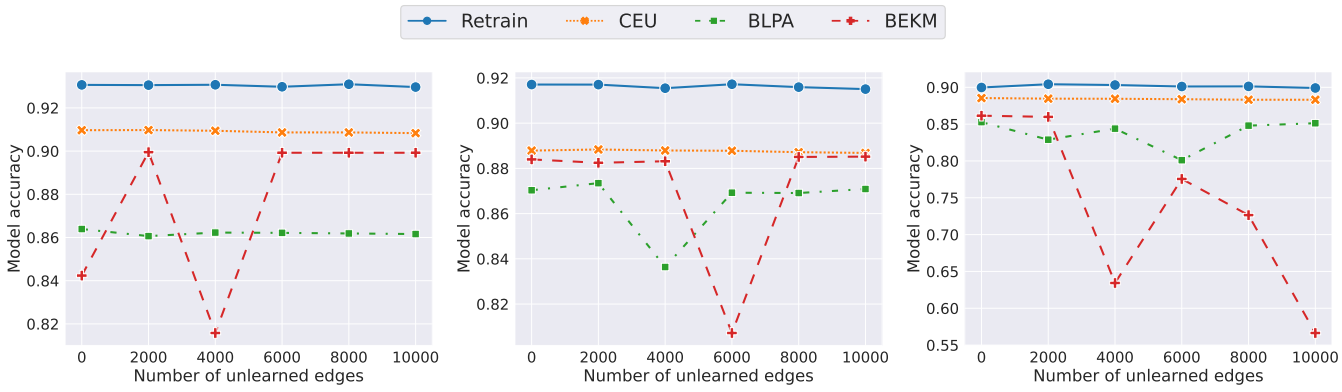
Figure 5: CGU [10] vs. CEU (SGC model and Cora dataset). Wendy:Enlarge font size on both x-axis and y-axis labels.Kun:done

that CGU and CEU have comparable unlearning efficacy. Due to the limited space, we include the results in Appendix E.2.

6.4 Performance of Deep GNN Models

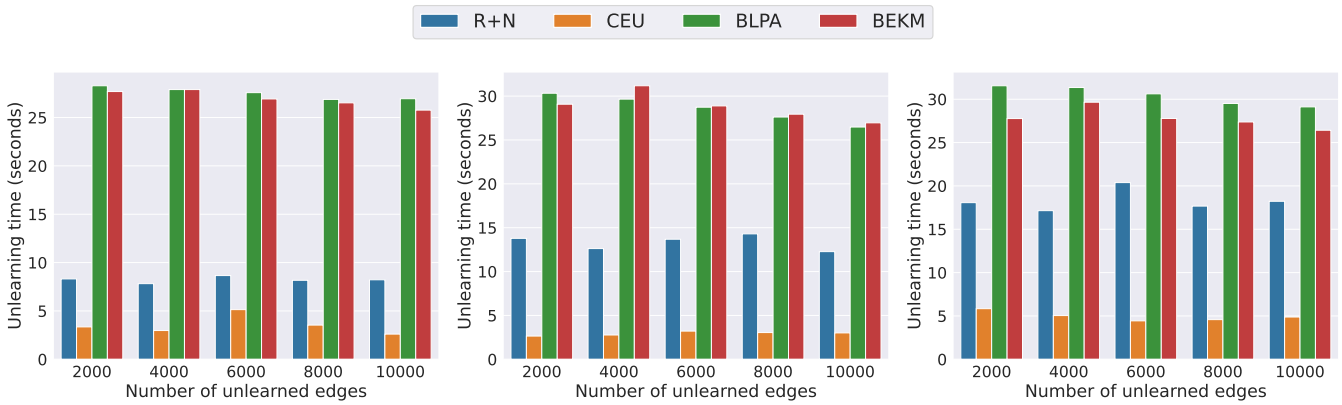
So far, we only consider the linear GCN model that meets Assumption 6. Next, we evaluate the performance of CEU on deep GNN models that do not meet Assumption 6. We consider GCN, GraphSAGE, and GIN models of various complexity (2-layer, 3-layer, and 4-layer) with ReLU as the activation function. We do not compare with the existing certified edge unlearning method [10] as it cannot be used on non-linear GNN models. Hence, we only compare CEU with the two baselines of exact edge unlearning (BLPA and BEKM).

Model accuracy. Figure 6 reports the model accuracy of the retrained model and CEU for the GCN model of various complexity. The results of GraphSAGE and GIN as well as the other two datasets are similar; they can be found in Appendix E.3. We observe two phenomena. First, although the model accuracy degrades for both retrained and unlearned GNN models of higher complexity, the model accuracy of the unlearned model remains close to that of the retrained model. The largest difference between model accuracy is only around 1.4% (Figure 6 (c)). Second, CEU outperforms two baselines (BLPA and BEKM) in terms of model accuracy for all



(a) 2-layer (b) 3-layer (c) 4-layer

Figure 6: Model accuracy of CEU for deep GCN models (CS dataset).



(a) 2-layer (b) 3-layer (c) 4-layer

Figure 7: Time performance of retraining and CEU for deep GCN models (CS dataset).

the settings. For example, the model accuracy of CEU on the 4-layer GCN is 30% higher than BEKM when removing 10,000 edges (Figure 6 (c)). This demonstrates the advantage of CEU to the exact graph unlearning. We also observe that the model accuracy of both retrained the unlearned models is insensitive to the number of edges. This is because the removed edges only takes a small portion (no more than 6%) of the original data.

Unlearning efficiency. Figure 7 shows the running time of retraining and CEU on GCN models with CS dataset. The time performance results of the other two datasets are included in Appendix E.3. We observe that, although the running time for both retraining and CEU grows with the increase in the complexity of GNN models, CEU is always significantly faster than retraining in all the settings, with the speedup factor as large as 5.2×. Furthermore, CEU is dominantly faster than the two baselines of exact unlearning (BLPA and BEKM), with a speedup as large as one-order magnitude.

Unlearning efficacy. Table 3 presents the attack performance of StealLink [20] when inferring the removed edges E_{UL} from the original model, the retrained model, as well as the unlearned model by CEU for CS dataset. The results of the other settings can be found in Appendix E.3. We have the following observations. First, while StealLink is highly effective in predicting the presence of E_{UL} from the original model ("Original" column), its attack accuracy is significantly reduced to close to 0.5 when being launched against both the retrained and unlearned models ("Retrain" and "CEU" columns). This indicates that CEU exhibits a similar capability as retraining to make deep GNN models forget the removed edges.
Wendy:Add summary of comparing with the baselines.

7 CONCLUSION

In this paper, we design CEU an efficient edge unlearning method that handles batch edge removal from GNNs. We prove that CEU can provide the theoretical guarantee of unlearning for GNN models under certain assumptions of convexity of the model's loss function. Our extensive set of experiments demonstrates that CEU can

Table 3: Unlearning efficacy for deep GCNs (CS dataset). Wendy:Should include at least one baseline.

$\lvert E_{\text{UL}}\rvert$	2-layer			3-layer			4-layer		
	Original	Retrain	CEU	Original	Retrain	CEU	Original	Retrain	CEU
2K	0.960	0.547	0.547	0.957	0.543	0.547	0.955	0.543	0.551
4K	0.960	0.545	0.552	0.956	0.545	0.549	0.956	0.547	0.553
6K	0.959	0.550	0.555	0.957	0.544	0.552	0.955	0.547	0.550
8K	0.959	0.553	0.554	0.956	0.549	0.550	0.956	0.547	0.553
10K	0.960	0.550	0.554	0.956	0.551	0.554	0.956	0.549	0.554

achieve significant speedup gains over retraining while delivering similar model accuracy for both linear and deep GNN models.

There are several research directions for future work. An interesting direction will be extending to handle the removal of nodes from graphs. It is seemly straightforward that node unlearning can be easily adapted from edge unlearning, as removing a node v from a graph is equivalent to removing all the edges that connect with v in the graph. However, node unlearning indeed is more challenging than edge unlearning, as removing a node entirely from the model requires removing not only the edges connected with the node but also its features and labels. We will explore how to design efficient and certified node learning methods for the future work. Another interesting direction is to add additional constraints on unlearning. A possible constraint is the *unlearning capacity*, i.e., the maximum number of edges that can be deleted while still ensuring good model accuracy.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. This project is supported by the National Science Foundation (NSF) under Grant No. 2029038, 2135988, 2212323, 2119331, 1951729, and 1953893. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agency.

REFERENCES

[1] D. P. Bertsekas. *Nonlinear Programming*. Massachusetts: Athena Scientific, 1999.

[2] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot. Machine unlearning. In *Symposium on Security and Privacy (SP)*, pages 141–159, 2021.

[3] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] J. Brophy and D. Lowd. Machine unlearning for random forests. In *International Conference on Machine Learning*, pages 1092–1104, 2021.

[5] Y. Cao and J. Yang. Towards making systems forget with machine unlearning. In *Symposium on Security and Privacy*, pages 463–480, 2015.

[6] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, 13, 2000.

[7] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

[8] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang. Graph unlearning. In *ACM Conference on Computer and Communications Security (CCS)*, 2022.

[9] J. Cheng, G. Dasoulas, H. He, C. Agarwal, and M. Zitnik. GNNDelete: A general strategy for unlearning in graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023.

[10] E. Chien, C. Pan, and O. Milenkovic. Certified graph unlearning. In *NeurIPS 2022 New Frontiers in Graph Learning Workshop (NeurIPS GLFrontiers 2022)*, 2022.

[11] W. Cong and M. Mahdavi. Grapheditor: An efficient graph representation learning and unlearning approach, 2023.

[12] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284, 2006.

[14] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[15] A. Golatkar, A. Achille, A. Ravichandran, M. Polito, and S. Soatto. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 792–801, 2021.

[16] A. Golatkar, A. Achille, and S. Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020.

[17] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten. Certified data removal from machine learning models. In *International Conference on Machine Learning*, pages 3832–3842, 2020.

[18] V. Gupta, C. Jung, S. Neel, A. Roth, S. Sharifi-Malvajerdi, and C. Waites. Adaptive machine unlearning. *Advances in Neural Information Processing Systems*, 34, 2021.

[19] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[20] X. He, J. Jia, M. Backes, N. Z. Gong, and Y. Zhang. Stealing links from graph neural networks. In *30th USENIX Security Symposium*, pages 2669–2686, 2021.

[21] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR*, 2017.

[22] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894, 2017.

[23] C. Kwak, J. Lee, K. Park, and H. Lee. Let machines unlearn–machine unlearning and the right to be forgotten. In *America's Conference on Information Systems (AMCIS)*, 2017.

[24] A. Mahadevan and M. Mathioudakis. Certifiable machine unlearning for linear models. *arXiv preprint arXiv:2106.15093*, 2021.

[25] J. Martens et al. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[26] S. Neel, A. Roth, and S. Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pages 931–962, 2021.

[27] C. Pan, E. Chien, and O. Milenkovic. Unlearning nonlinear graph classifiers in the limited training data regime. *arXiv preprint arXiv:2211.03216*, 2022.

[28] S. L. Pardau. The california consumer privacy act: Towards a european-style privacy regime in the united states. *J. Tech. L. & Pol'y*, 23:68, 2018.

[29] C. Parliament. Personal information protection and electronic documents act. *Consolidated Acts, SC 2000, c*, 5:13, 2000.

[30] B. A. Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[31] R. Pytlak. *Conjugate gradient algorithms in nonconvex optimization*, volume 89. Springer Science & Business Media, 2008.

[32] G. D. P. Regulation. General data protection regulation (gdpr). *Intersoft Consulting, Accessed in October*, 24(1), 2018.

[33] A. Sekhari, J. Acharya, G. Kamath, and A. T. Suresh. Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems*, 34, 2021.

[34] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

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

[36] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[37] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[38] P. Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

[39] P. Wolfe. Convergence conditions for ascent methods. ii: Some corrections. *SIAM review*, 13(2):185–188, 1971.

[40] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[41] Y. Wu, E. Dobriban, and S. Davidson. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*, pages 10355–10366, 2020.

[42] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[43] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48, 2016.

[44] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 793–803, 2019.

APPENDIX

In this appendix, we describe additional experimental results. The complete proofs of our theorems and lemmas can be found in our full paper [] **Wendy:add citation of the full paper**. Our code is available at https://github.com/kunwu522/certified_edge_unlearning **Wendy:Add the link to the codeKun:added**. Please note that the code is subjected to reorganization to improve readability.

A COMPLETE PROOF OF THEOREMS AND LEMMAS

A.1 Proof of $\zeta = \frac{1}{|V|}$ in Eqn. (11)

In this section, we prove the statement that Eqn. (11) is equivalent to retraining if $\zeta = \frac{1}{|V|}$.

Proof. Recall Eqn. (11) as defined below:

$$\theta_{\zeta, V_{E_{\text{UL}}}} = \arg\min_{\theta} \frac{1}{|V|} \sum_{v \in V} \mathcal{L}_b(\theta; v, E) + \zeta \Big(\sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E \backslash E_{\text{UL}}) - \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E) \Big).$$

The first term $\frac{1}{|V|} \sum_{v \in V} \mathcal{L}_b(\theta; v, E)$ can be split in the following way:

$$\begin{aligned} &\frac{1}{|V|} \sum_{v \in V} \mathcal{L}_b(\theta; v, E) \\ &= \frac{1}{|V|} \sum_{v \in V \backslash V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E) + \frac{1}{|V|} \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E) \end{aligned} \tag{17}$$

By seting $\zeta = \frac{1}{|V|}$ and plugging Eqn. (17) into Eqn. (11), we have the following:

$$\begin{aligned} \theta_{\zeta, V_{E_{\text{UL}}}} = \arg\min &\frac{1}{|V|} \sum_{v \in V \backslash V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E) \\ &+ \frac{1}{|V|} \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E \backslash E_{\text{UL}}). \end{aligned}$$

As $v \in V \backslash V_{E_{\text{UL}}}$ will not be affected by E_{UL}, we can use $E \backslash E_{\text{UL}}$ to replace E as

$$\begin{aligned} \theta_{\zeta, V_{E_{\text{UL}}}} &= \arg\min \frac{1}{|V|} \sum_{v \in V \backslash V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E \backslash E_{\text{UL}}) \\ &\quad + \frac{1}{|V|} \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E \backslash E_{\text{UL}}) \\ &= \arg\min \frac{1}{|V|} \sum_{v \in V} \mathcal{L}_b(\theta; v, E \backslash E_{\text{UL}}). \end{aligned}$$

Then the statement follows. □

A.2 Proof of Theorem 3

Proof. For simplicity, we first define

$$R_b(\theta, V, E) = \sum_{v \in V} \mathcal{L}_b(\theta, v, E).$$

Then, we formulate a GNN learning process as

$$\tilde{\theta}_{\text{OR}} = \arg\min_{\theta} \frac{1}{|V|} R_b(\theta, V, E). \tag{18}$$

Since removing edges can be considered as perturbing the input, we introduce Eqn. 11,

$$\begin{aligned} \tilde{\theta}_{\zeta} &= \arg\min_{\theta} \frac{1}{|V|} \sum_{v \in V} \mathcal{L}_b(\theta; v, E) + \zeta \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E \backslash E_{\text{UL}}) \\ &\quad - \zeta \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta; v, E) \\ &= \arg\min_{\theta} \frac{1}{|V|} R_b(\theta, V, E) + \zeta R_b(\theta, V_{E_{\text{UL}}}, E \backslash E_{\text{UL}}) - \zeta R_b(\theta, V_{E_{\text{UL}}}, E). \end{aligned} \tag{19}$$

We note a necessary condition is that the gradient of Eqn. 19 at $\tilde{\theta}_{\zeta}$ is zero. Then, we have

$$0 = \frac{1}{|V|} \nabla_{\theta} R(\tilde{\theta}_{\zeta}, V, E) + \zeta \nabla_{\theta} R_b(\tilde{\theta}_{\zeta}, V_{E_{\text{UL}}}, E \backslash E_{\text{UL}}) - \zeta \nabla_{\theta} R(\tilde{\theta}_{\zeta}, V_{E_{\text{UL}}}, E).$$

Next, we apply Taylor series at θ_{OR} and we get

$$\begin{aligned} 0 \approx &\frac{1}{|V|} \nabla_{\theta} R_b(\theta_{\text{OR}}, V, E) + \zeta \nabla_{\theta} R_b(\theta_{\text{OR}}, V_{E_{\text{UL}}}, E \backslash E_{\text{UL}}) \\ &- \zeta \nabla_{\theta} R_b(\theta_{\text{OR}}, V_{E_{\text{UL}}}, E) + \Big[\frac{1}{|V|} \nabla_{\theta}^2 R(\theta_{\text{OR}}, V, E) \\ &+ \zeta \nabla_{\theta}^2 R(\theta_{\text{OR}}, V_{E_{\text{UL}}}, E \backslash E_{\text{UL}}) - \zeta \nabla_{\theta}^2 R(\theta_{\text{OR}}, V_{E_{\text{UL}}}, E) \Big] (\theta_{\zeta} - \theta_{\text{OR}}), \end{aligned} \tag{20}$$

where we have dropped $o(\tilde{\theta}_{\text{OR}} - \tilde{\theta}_{\zeta})$ for approximation. Then Eqn. (20) is a linear system of E_{UL}, the influence of E_{UL}. Since $\tilde{\theta}_{\text{OR}}$ is the minimum of Eqn. (18), we have $\frac{1}{|V|} \nabla R_b(\tilde{\theta}_{\text{OR}}, V, E) = 0$. As ζ is a small value, we drop the two $o(\zeta)$ terms and have the following:

$$\begin{aligned} &\frac{1}{|V|} \nabla_{\theta}^2 R_b(\tilde{\theta}_{\text{OR}}, V, E)(\tilde{\theta}_{\zeta} - \tilde{\theta}_{\text{OR}}) \\ &+ \zeta \Big(\nabla_{\theta} R(\tilde{\theta}_{\text{OR}}, V_{E_{\text{UL}}}, E \backslash E_{\text{UL}}) - \nabla_{\theta} R(\tilde{\theta}_{\text{OR}}, V_{E_{\text{UL}}}, E) \Big) \approx 0. \end{aligned}$$

Suppose Eqn. (18) is convex, then

$$\begin{aligned} \tilde{\theta}_{\zeta} - \tilde{\theta}_{\text{OR}} \approx &- \frac{1}{|V|} \nabla_{\theta}^2 R_b(\tilde{\theta}_{\text{OR}}, V, E)^{-1} \\ &\times \Big(\nabla_{\theta} R_b(\tilde{\theta}_{\text{OR}}, V_{E_{\text{UL}}}, E \backslash E_{\text{UL}}) - \nabla_{\theta} R(\tilde{\theta}_{\text{OR}}, V_{E_{\text{UL}}}, E) \Big) \zeta \end{aligned}$$

We have the following:

$$\begin{aligned} I_{E_{\text{UL}}} &:= \frac{d(\tilde{\theta}_{\zeta} - \tilde{\theta}_{\text{OR}})}{d\zeta} \Big|_{\zeta = 0} \\ &= -\tilde{H}_{\tilde{\theta}_{\text{OR}}}^{-1} \Big(\nabla_{\theta} R_b(\tilde{\theta}_{\text{OR}}, V_{E_{\text{UL}}}, E \backslash E_{\text{UL}}) - \nabla_{\theta} R_b(\tilde{\theta}_{\text{OR}}, V_{E_{\text{UL}}}, E) \Big) \end{aligned}$$

where $\tilde{H}_{\tilde{\theta}_{\text{OR}}} := \nabla^2 \frac{1}{|V|} \sum_{v \in V} L(\tilde{\theta}_{\text{OR}}, v, E)$. □

A.3 Proof of Lemma 5

Proof. Recall that a key step of CG update is calculating the gradient of $f(x)$ as

$$\nabla f(x) = \tilde{H}_{\tilde{\theta}_{\text{OR}}} x - \Big(\nabla_{\theta} \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta_{\text{OR}}; v, E \backslash E_{\text{UL}}) - \nabla_{\theta} \sum_{v \in V_{E_{\text{UL}}}} \mathcal{L}_b(\theta_{\text{OR}}; v, E) \Big).$$

As $\tilde{H}_{\tilde{\theta}_{\text{OR}}} \in \mathbb{R}^{|\theta|\times|\theta|}$, we can not explicitly compute $\tilde{H}_{\tilde{\theta}_{\text{OR}}}$. Instead, we utilize Hessian-vector product [30] to approximately calculate $\tilde{H}_{\tilde{\theta}_{\text{OR}}} x$ for some very small step size $r > 0$ by

$$\tilde{H}_{\tilde{\theta}_{\text{OR}}} x \approx \frac{g(\tilde{\theta}_{\text{OR}} + rx) - g(\tilde{\theta}_{\text{OR}})}{r}$$
$$g(\theta) \coloneqq \nabla_\theta \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}}) - \nabla_\theta \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E).$$

As the memory cost of evaluating the function value of $g(\cdot)$ is $O(|\theta|)$, Lemma 5 follows. □

A.4 Proof of Theorem 8

Proof. Let $G(\theta) = \nabla_\theta \frac{1}{|V|} \sum_{v\in V} \mathcal{L}_b(\theta; v, E\backslash E_{\text{UL}})$, by Taylor's Theorem, there exists some $\eta \in [0, 1]$ such that,

$$\begin{aligned} G(\tilde{\theta}_{\text{UL}}) &\approx G(\tilde{\theta}_{\text{OR}} + \frac{1}{|V|}\tilde{I}_{E_{\text{UL}}}) \\ &= G(\tilde{\theta}_{\text{OR}}) + \nabla G(\tilde{\theta}_{\text{OR}} + \frac{\eta}{|V|}\tilde{I}_{E_{\text{UL}}}) \frac{1}{|V|}\tilde{I}_{E_{\text{UL}}}. \end{aligned}$$

Since $\nabla G(\tilde{\theta}_{\text{OR}} + \frac{\eta}{|V|}\tilde{I}_{E_{\text{UL}}})$ is the Hessian of $\mathcal{L}_b$ calculated at $\theta_\eta = \tilde{\theta}_{\text{OR}} + \frac{\eta}{|V|}\tilde{I}_{E_{\text{UL}}}$, we denote ∇G as H_η^{-1} and let

$\Delta = \nabla_\theta \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E) - \nabla_\theta \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}})$. Thus,

$$\begin{aligned} G(\tilde{\theta}_{\text{UL}}) &= G(\tilde{\theta}_{\text{OR}}) + H_{\theta_\eta} \frac{1}{|V|}\tilde{I}_{E_{\text{UL}}} \\ &= G(\tilde{\theta}_{\text{OR}}) + \frac{1}{|V|} H_{\theta_\eta} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta \\ &= (G(\tilde{\theta}_{\text{OR}}) + \frac{1}{|V|}\Delta) + (\frac{1}{|V|} H_{\theta_\eta} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta - \frac{1}{|V|}\Delta). \end{aligned}$$

Since $G(\tilde{\theta}_{\text{OR}}) = \frac{1}{|V|}\nabla \sum_{v\in V\backslash V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}})$ $+ \frac{1}{|V|}\nabla \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}})$, let us first look at $G(\tilde{\theta}_{\text{OR}}) + \frac{1}{|V|}\Delta$ as

$$\begin{aligned} G(\tilde{\theta}_{\text{OR}}) + \frac{1}{|V|}\Delta = \frac{1}{|V|}\nabla &\sum_{v\in V\backslash V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}}) \\ &+ \frac{1}{|V|}\nabla \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E). \end{aligned}$$

Since $v \in V\backslash V_{E_{\text{UL}}}$ is not in the set of infected nodes, therefore, $\sum_{v\in V\backslash V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}}) = \sum_{v\in V\backslash V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E)$, then we have

$$G(\tilde{\theta}_{\text{OR}}) + \frac{1}{|V|}\Delta = \nabla_\theta \frac{1}{|V|} \sum_{v\in V} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E) = 0.$$

Back to $G(\tilde{\theta}_{\text{UL}})$, we have

$$\begin{aligned} G(\tilde{\theta}_{\text{UL}}) &= \frac{1}{|V|} H_{\theta_\eta} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta - \frac{1}{|V|}\Delta \\ &= \frac{1}{|V|} H_{\theta_\eta} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta - \frac{1}{|V|} \tilde{H}_{\theta_{\text{OR}}} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta \\ &= \frac{1}{|V|} (H_{\theta_\eta} - \tilde{H}_{\theta_{\text{OR}}}) \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta. \end{aligned} \tag{21}$$

Next,

$$\begin{aligned} ||G(\tilde{\theta}_{\text{UL}})||_2 &= ||\frac{1}{|V|}(H_{\theta_\eta} - \tilde{H}_{\theta_{\text{OR}}}) \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta||_2 \\ &\le ||H_{\theta_\eta} - \tilde{H}_{\theta_{\text{OR}}}||_2 ||\frac{1}{|V|} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta||_2 \end{aligned}$$

Assume the Hessian of L is γ_1-Lipschitz, the first norm on the right-hand side can be bounded as

$$\begin{aligned} ||H_{\theta_\eta} - \tilde{H}_{\theta_{\text{OR}}}||_2 &= ||\nabla^2 \sum_{v\in V} \mathcal{L}_b(\theta_\eta; v, E\backslash E_{\text{UL}}) \\ &\quad - \nabla^2 \sum_{v\in V} \mathcal{L}(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}})||_2 \\ &\le \gamma_1 ||\theta_\eta - \tilde{\theta}_{\text{OR}}||_2 = \gamma_1 ||\frac{\eta}{|V|} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta||_2 \\ &\le \gamma_1 ||\frac{1}{|V|} \tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta||_2 \end{aligned} \tag{22}$$

where $\gamma_1 \ge 0$.

Then,

$$||G(\tilde{\theta}_{\text{UL}})||_2 \le \gamma_1 \frac{1}{|V|} ||\tilde{H}_{\theta_{\text{OR}}}^{-1} \Delta||_2^2.$$

Since $\mathcal{L}$ is λ-strongly convex, we have $||\tilde{H}_{\theta_{\text{OR}}}^{-1}||_2 \le \frac{1}{\lambda}$, we mainly focus on $||\Delta||_2$.

Observe

$$\begin{aligned} ||\Delta||_2 &= ||\nabla_\theta \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E) - \nabla_\theta \sum_{v\in V_{E_{\text{UL}}}} \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}})||_2 \\ &= \Big\| \sum_{v\in V_{E_{\text{UL}}}} \big[\nabla \mathcal{L}_b + (\tilde{\theta}_{\text{OR}}; v, E) - \nabla \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}})\big] \Big\|_2 \\ &\le \sum_{v\in V_{E_{\text{UL}}}} \Big\| \nabla \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E) - \nabla \mathcal{L}_b(\tilde{\theta}_{\text{OR}}; v, E\backslash E_{\text{UL}}) \Big\|_2 \end{aligned}$$

Consider Z denotes the input of the loss function without removing edges, such as $Z_v \triangleq (\tilde{\theta}_{\text{OR}}; E)$, and Z' denotes $(\tilde{\theta}_{\text{OR}}; E\backslash E_{\text{UL}})$, we have

$$\begin{aligned} ||\Delta||_2 &= \sum_{v\in V_{E_{\text{UL}}}} ||\nabla \mathcal{L}_b(Z_v) - \nabla \mathcal{L}_b(Z'_v)||_2 \\ &\overset{(a)}{\le} \gamma_2 \sum_{v\in V_{E_{\text{UL}}}} \Big\| Z_v - Z'_v \Big\|_2 \\ &= \gamma_2 \sum_{v\in V_{E_{\text{UL}}}} \Big\| \sum_{u\in N(v)\bigcup\{v\}} \tilde{\theta}_{\text{OR}}^T x_u - \sum_{u\in N'(v)\bigcup\{v\}} \tilde{\theta}_{\text{OR}}^T x_u \Big\|_2 \\ &\overset{(b)}{=} \gamma_2 \sum_{v\in V_{E_{\text{UL}}}} \Big\| \sum_{u\in \overline{N(v)}} \tilde{\theta}_{\text{OR}}^T x_u \Big\|_2 \\ &\le \gamma_2 \sum_{v\in V_{E_{\text{UL}}}} \sum_{u\in \overline{N(v)}} \Big\| \tilde{\theta}_{\text{OR}}^T x_u \Big\|_2 \\ &\le \gamma_2 \sum_{v\in V_{E_{\text{UL}}}} \sum_{u\in \overline{N(v)}} \Big\| \tilde{\theta}_{\text{OR}} \Big\|_2 \Big\| x_u \Big\|_2 \\ &\overset{(c)}{\le} \gamma_2 \sum_{v\in V_{E_{\text{UL}}}} \sum_{u\in \overline{N(v)}} \Big\| \tilde{\theta}_{\text{OR}} \Big\|_2 \end{aligned}$$

$$\leq \gamma_2 \sum_{v \in V_{E_{\text{UL}}}} \sum_{u \in \overline{N(v)}} \frac{c}{\lambda}$$
$$\overset{\text{(d)}}{=} \gamma_2 \sum_{v \in V_{E_{\text{UL}}}} \frac{n_v c}{\lambda}$$

where N and N' are the set of neighbors of v and the set of neighbors that after removing E_{UL}, respectively. In (a), we apply γ_2-Lipschitz, and we obtain (b) due to $N' \subset N$ and let $\overline{N(v)} = N(\cdot) - N'(\cdot)$. According to Eqn. 15 in [10], we replace $||\theta_{\text{OR}}|| \leq \frac{c_1}{\lambda}$ in (c). In (d) $n_v = |\overline{N(v)}|$ denotes the number of nodes in $\overline{N(v)}$.

Finally,

$$\begin{aligned}
||G(\theta_{\text{UL}})||_2 &\leq \gamma_1 \frac{1}{|V|} ||\tilde{H}^{-1}_{\theta_{\text{OR}}} \Delta||_2^2 \\
&\leq \gamma_1 \frac{1}{|V|} ||\tilde{H}^{-1}_{\theta_{\text{OR}}}||_2^2 ||\Delta||_2^2 \\
&\leq \gamma_1 \frac{1}{|V|} \cdot \big(\frac{1}{\lambda}\big)^2 \cdot \big(\frac{\gamma_2 c_1}{\lambda} \sum_{v \in V_{E_{\text{UL}}}} n_v\big)^2 \\
&\leq \frac{\gamma_1 \gamma_2^2 c_1^2}{\lambda^4 |V|} \Big(\sum_{v \in V_{E_{\text{UL}}}} n_v \Big)^2
\end{aligned}$$

□

A.5 Proof of Theorem 9

Proof. The noisy loss function with l_2-regularization is defined as

$$\mathcal{L}_b = \frac{1}{|V|} \sum_{v \in V} L(\theta; v, E) + \frac{\lambda}{2} ||\theta||_2^2 + b^T \theta.$$

Correspondingly, the gradient of the noisy loss is

$$\nabla \mathcal{L}_b = \frac{1}{|V|} \sum_{v \in V} \nabla L(\theta; v, E) + \lambda \theta + b,$$

and the Hessian

$$\tilde{H}_\theta = \nabla^2 \mathcal{L}_b = \frac{1}{|V|} \sum_{v \in V} \nabla^2 L(\theta; v, E) + \lambda I,$$

Recall Eqn. 21,

$$\begin{aligned}
||G(\tilde{\theta}_{\text{UL}})|| &= ||\frac{1}{|V|}(H_{\theta_\eta} - \tilde{H}_{\theta_{\text{OR}}}) \tilde{H}^{-1}_{\theta_{\text{OR}}} \Delta||_2 \\
&= ||\frac{1}{|V|^2} \sum_{v \in V} (\nabla^2 \mathcal{L}_b(\theta_\eta) - \nabla^2 \mathcal{L}_b(\tilde{\theta}_{\text{OR}})) \tilde{H}^{-1}_{\theta_{\text{OR}}} \Delta||_2 \\
&\leq ||\frac{1}{|V|^2}|| \sum_{v \in V} ||\nabla^2 \mathcal{L}(\theta_\eta) - \nabla^2 \mathcal{L}(\tilde{\theta}_{\text{OR}})|| ||\tilde{H}^{-1}_{\theta_{\text{OR}}} \Delta||_2.
\end{aligned}$$

Given the Lipschitz constant γ of the second derivative $\nabla^2 L$, we have

$$\begin{aligned}
||G(\theta_{\text{UL}})|| &\leq \gamma \frac{1}{|V|^2} \sum_{v \in V} ||\theta_\eta - \tilde{\theta}_{\text{OR}}|| ||\tilde{H}^{-1}_{\theta_{\text{OR}}} \Delta||_2 \\
&\overset{\text{(a)}}{\leq} \gamma \frac{1}{|V|^2} ||\tilde{H}^{-1}_{\theta_{\text{OR}}} \Delta||_2^2,
\end{aligned}$$

(a) follows Eqn. (22).

□

A.6 Proof of Convexity of 1-layer GCN

Proof. Given a 1-layer GCN without normalization and a graph $G = (V, E)$ and the corresponding adjacency matrix A, node features $X \in \mathbb{R}^{|V| \times d}$, the loss function of a multi-class classification task can be described as

$$L = -\sum_{i=1}^{k} y_i \log \frac{\exp(AXW)_i}{\sum_{j=1}^{k} \exp(AXW)_j}, \tag{23}$$

where k is the number of classes and W denotes the trainable parameters of the GCN model. Since $\mathbf{y}$ is an one-hot vector, Eqn. 23 can be rewritten as

$$L = -\log \frac{exp(AXW)_i}{\sum_{j=1}^{k} \exp(AXW)_j}, \tag{24}$$

where i indicates the index of the true class.

By simplifying 24, we have

$$\begin{aligned}
L &= -\Big(\log \exp(AXW) - \log \sum_{j=1}^{k} \exp(AXW)_j \Big) \\
&= \log \sum_{j=1}^{k} \exp(AXW)_j - AXW.
\end{aligned} \tag{25}$$

The first term of Eqn. 25 is convex due to log-sum-exp is convex [3]. Hence, its Hessian is positive semidefinite. Meanwhile, the Hessian of the second term in Eqn. 25 is 0. Therefore, the Hessian of the loss function L is positive semidefinite as $\nabla^2 L \succeq 0$. As we know a twice differentiable function f is convex if and only if its Hessian is positive semidefinite, hence, the loss function of a 1-layer GCN model is convex. □

A.7 Details of UEU: Efficient Unlearning without Certified Guarantee

UEU follows a similar idea as CEU. It aims to identify an update to θ_{OR} through an analogous *one-shot **unl**earning update*:

$$\theta_{\text{UL}} = \theta_{\text{OR}} + \frac{1}{|V|} I_{E_{\text{UL}}},$$

where $I_{E_{\text{UL}}}$ is the *influence* of E_{UL} on the target model, i.e., the change on the model parameters by E_{UL}, and θ_{OR} refers to the parameters of the original model whose loss function does not have noise.

First, UEU computes the new parameters $\theta_{\zeta, E_{\text{UL}}}$ after the removal of E_{UL} as follows:

$$\begin{aligned}
\theta_{\zeta, V_{E_{\text{UL}}}} = \arg\min_\theta \frac{1}{|V|} \sum_{v \in V} L(\theta; v, E) + \zeta \sum_{v \in V_{E_{\text{UL}}}} L(\theta; v, E \backslash E_{\text{UL}}) \\
- \zeta \sum_{v \in V_{E_{\text{UL}}}} L(\theta; v, E).
\end{aligned} \tag{26}$$

Intuitively, Eqn. (26) approximates the effects that moving ζ mass of perturbation on $V_{E_{\text{UL}}}$ with $E \backslash E_{\text{UL}}$ in place of E. Then we obtain the following theorem.

Theorem 10. *Given the parameters θ_{OR} obtained by $\mathcal{A}_{\text{UL}}$ on a graph G, and the loss function L, assume that L is twice-differentiable*

and convex in θ, then the influence of a set of edges E_{UL} is:

$$I_{E_{\mathrm{UL}}} = -H_{\theta_{\mathrm{OR}}}^{-1}\Big(\nabla_\theta \sum_{v\in V_{E_{\mathrm{UL}}}} L(\theta_{\mathrm{OR}}; v, E\backslash E_{\mathrm{UL}}) - \nabla_\theta \sum_{v\in V_{E_{\mathrm{UL}}}} L(\theta_{\mathrm{OR}}; v, E)\Big) \tag{27}$$

where $H_{OR} \coloneqq \nabla^2 \frac{1}{|V|}\sum_{v\in V} L(\theta_{OR}, v, E)$ is the Hessian matrix of L with respect to θ_{OR}.

Proof. For simplicity, we first define

$$R(\theta, V, E) = \sum_{v\in V} L(\theta, v, E).$$

Then, we formulate a GNN learning process as

$$\theta_{\mathrm{OR}} = \arg\min_\theta \frac{1}{|V|} R(\theta, V, E). \tag{28}$$

Since removing edges can be considered as perturbing the input, we introduce Eqn. 11,

$$\begin{aligned}\theta_\zeta =& \arg\min_\theta \frac{1}{|V|}\sum_{v\in V} L(\theta; v, E) + \zeta \sum_{v\in V_{E_{\mathrm{UL}}}} L(\theta; v, E\backslash E_{\mathrm{UL}}) \\ & - \zeta \sum_{v\in V_{E_{\mathrm{UL}}}} L(\theta; v, E) \\ =& \arg\min_\theta \frac{1}{|V|} R(\theta, V, E) + \zeta R(\theta, V_{E_{\mathrm{UL}}}, E\backslash E_{\mathrm{UL}}) - \zeta R(\theta, V_{E_{\mathrm{UL}}}, E).\end{aligned} \tag{29}$$

We note a necessary condition is that the gradient of Eqn. 29 at θ_ζ is zero. Then, we have

$$0 = \frac{1}{|V|}\nabla_\theta R(\theta_\zeta, V, E) + \zeta\nabla_\theta R(\theta_\zeta, V_{E_{\mathrm{UL}}}, E\backslash E_{\mathrm{UL}}) - \zeta\nabla_\theta R(\theta_\zeta, V_{E_{\mathrm{UL}}}, E). \tag{30}$$

Next, we apply Taylor series at θ_{OR} and we get

$$\begin{aligned}0 \approx& \frac{1}{|V|}\nabla_\theta R(\theta_{\mathrm{OR}}, V, E) + \zeta\nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E\backslash E_{\mathrm{UL}}) - \zeta\nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E) \\ & + \Big[\frac{1}{|V|}\nabla_\theta^2 R(\theta_{\mathrm{OR}}, V, E) + \zeta\nabla_\theta^2 R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E\backslash E_{\mathrm{UL}}) \\ & - \zeta\nabla_\theta^2 R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E)\Big](\theta_\zeta - \theta_{\mathrm{OR}}),\end{aligned} \tag{31}$$

where we have dropped $o(\theta_{\mathrm{OR}} - \theta_\zeta)$ for approximation. Then Eqn. (31) is a linear system of E_{UL}, the influence of E_{UL}. Since θ_{OR} is the minimum of Eqn. (28), we have $\frac{1}{|V|}\nabla R(\theta_{\mathrm{OR}}, V, E) = 0$. As ζ is a small value, we drop the two $o(\zeta)$ terms and have the following:

$$\begin{aligned}&\frac{1}{|V|}\nabla_\theta^2 R(\theta_{\mathrm{OR}}, V, E)(\theta_\zeta - \theta_{\mathrm{OR}}) \\ &+ \zeta\Big(\nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E\backslash E_{\mathrm{UL}}) - \nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E)\Big) \approx 0.\end{aligned}$$

Suppose Eqn. (28) is convex, then

$$\begin{aligned}&\theta_\zeta - \theta_{\mathrm{OR}} \\ &\approx -\frac{1}{|V|}\nabla_\theta^2 R(\theta_{\mathrm{OR}}, V, E)^{-1}\Big(\nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E\backslash E_{\mathrm{UL}}) - \nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E)\Big)\zeta\end{aligned}$$

Denote

$$\begin{aligned}I_{E_{\mathrm{UL}}} &\coloneqq \frac{d(\theta_\zeta - \theta_{\mathrm{OR}})}{d\zeta}\Big|_{\zeta=0} \\ &= -H_{\theta_{\mathrm{OR}}}^{-1}\Big(\nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E\backslash E_{\mathrm{UL}}) - \nabla_\theta R(\theta_{\mathrm{OR}}, V_{E_{\mathrm{UL}}}, E)\Big)\end{aligned}$$

where $H_{OR} \coloneqq \nabla^2 \frac{1}{|V|}\sum_{v\in V} L(\theta_{OR}, v, E)$. □

B MORE DETAILS OF EXPERIMENTAL SETUP

B.1 Details of Datasets

Table 4 summarizes the statistical information of the three graph datasets (Cora, Citeseer, and CS) we used in the experiments. Cora and Citeseer datasets are citation graphs, while CS dataset is a co-author graph.

B.2 Details of Model Setup

To ensure a fair comparison between the retrained and unlearned models, we use the same model size (i.e., the same number of layers and number of neurons) for both retraining and unlearned models. All GNN models are trained with a learning rate of 0.001. We train the models by 1,000 epochs, with the early-stopping condition so that the validation loss does not decrease for 20 epochs.

C IMPACT OF TYPE OF REMOVED EDGES ON UNLEARNING

To evaluate the impact of edge types on unlearning performance, we consider three different strategies to pick edges for removal.

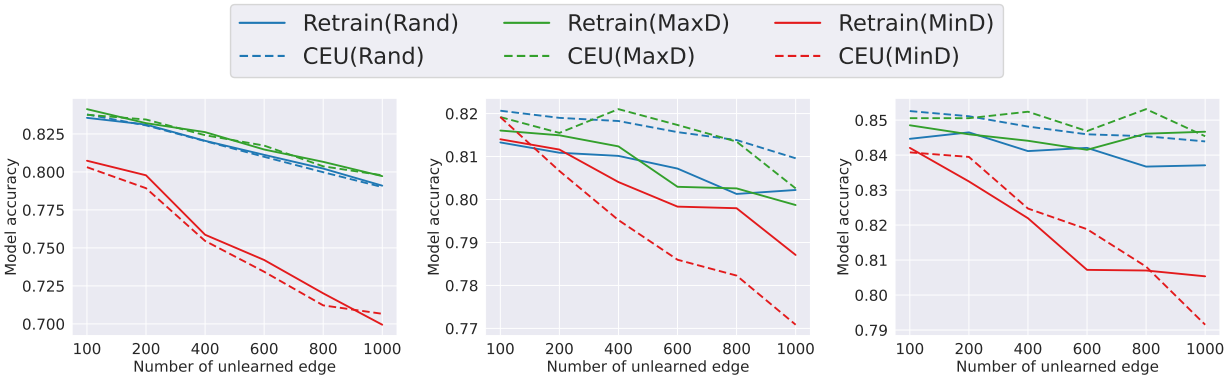
- Random sampling (**Rand**): we randomly sample k edges from the training graph.
- Max-degree & Min-degree sampling (**MaxD & MinD**): As GNN models aggregate information from the neighboring nodes when generating node embeddings, the size of node neighbors (i.e., node degree) directly affects the amounts of information of edges encoded in GNNs. Therefore, we measure the importance of an edge $e(v_i, v_j)$ as its *degree* defined as *EdgeDegree*(e) = degree(v_i)+degree(v_j). Then we rank edges by their degree in descending order and pick k edges of the largest edge degree (**MaxD**) as well as the k edges of the smallest edge degree (**MinD**) for removal.

Model accuracy. Figure 8 shows the model accuracy results for removing three types of removed edges. We observe non-negligible impact of the type of removed edges on both retrained and unlearned models. The retrained model witnesses the largest and smallest drop in model accuracy for the MinD and MaxD edges respectively (Figure 8 (a) & (b). Meanwhile, the unlearned model witnesses the same trend as their corresponding retrained models, where removing MinD edges and MaxD edges incur the largest and smallest model accuracy downgrade on the unlearned model.

Unlearning efficacy. Table 5 reports the unlearning efficacy results (AUC of StealLink attack) for removing three types of edges. We observe the following phenomena. First, StealLink is effective on predicting the existence of the three types of removed edges from the original model, with the edges of "Rand" and "MinD" types most vulnerable to the privacy attack. Second, the AUC of the attack noticeably reduces when inferring from either the retrained or the unlearned model. In particular, the AUC is reduced to around 0.6 for both "Rand" and "MinD" types of edges. This demonstrates the forgettablity power of CEU. We also observe that the edges of "MaxD" type always have the highest AUC before and after

Table 4: Description of datasets

Dataset	#. Features	#. Nodes	#. Edges	#. Classes	Min Degree	Max Degree	Avg. Degree
Cora	1,433	2,708	5,429	7	2	198	21.82
CiteSeer	3,703	3,327	4,552	6	2	126	13.83
CS	6,805	18,333	163,788	15	3	262	36.43



(a) GCN (b) GraphSAGE (c) GIN

Figure 8: Model accuracy of different edge types ($\sigma = 0.1$).

Table 5: Impact of edge types on unlearning efficacy of CEU.

$\lvert E_{\text{UL}}\rvert$	Edge Type	Original	R+N	CEU
200	MaxD	0.631	0.610	0.629
	Rand	0.930	0.595	0.609
	MinD	0.931	0.704	0.709
400	MaxD	0.571	0.676	0.679
	Rand	0.928	0.588	0.590
	MinD	0.932	0.690	0.687
600	MaxD	0.644	0.661	0.667
	Rand	0.927	0.592	0.588
	MinD	0.927	0.689	0.681
800	MaxD	0.702	0.686	0.688
	Rand	0.927	0.594	0.599
	MinD	0.923	0.666	0.663
1000	MaxD	0.759	0.688	0.694
	Rand	0.928	0.602	0.593
	MinD	0.923	0.659	0.653

retraining/unlearning. This suggests that "MaxD" type edges are most vulnerable to the attack.

D SEQUENTIAL UNLEARNING

So far we only considered deleting one batch of edges. In practice, there can be multiple batch deletion requests to forget the edges in a sequential fashion. Next, we focus on the scenario where multiple edge batches are removed sequentially. Specifically, we divide the to-be-removed E_{UL} into $k > 1$ disjoint batches $\{B_i\}_{i=1}^{k}$, with each batch consisting of the same number of edges. For each batch B_i ($1 \le i \le k-1$), we consider the target model obtained from retraining/unlearning of the previous batch B_{i-1} as the original model θ_{OR}, and update θ_{OR} by removing B_i (either by retraining or unlearning). We evaluate the target model accuracy under sequential unlearning and compare it with that under one-batch unlearning.

We consider $k = 4$, and report the target model accuracy for deleting E_{UL} in one batch and deleting E_{UL} in $k = 4$ batches in Table 6. We also report the target model accuracy of the retrained and

Table 6: Target model accuracy under single-batch and sequential unlearning

Dataset	Setting	Sequential				Batch
		B_1	B_2	B_3	B_4	
Cora	Retrain	0.875	0.874	0.873	0.874	0.872
	UEU	0.873	0.873	0.873	0.875	0.871
	R+N	0.818	0.816	0.818	0.819	0.815
	CEU	0.815	0.814	0.821	0.820	0.811
CiteSeer	Retrain	0.778	0.778	0.778	0.777	0.776
	UEU	0.777	0.778	0.778	0.777	0.774
	R+N	0.750	0.749	0.750	0.749	0.750
	CEU	0.750	0.755	0.751	0.752	0.753
CS	Retrain	0.937	0.937	0.937	0.937	0.937
	UEU	0.937	0.937	0.938	0.937	0.937
	R+N	0.930	0.930	0.930	0.930	0.930
	CEU	0.931	0.929	0.931	0.931	0.930

Table 7: Model accuracy of CEU, two retrained models (Retrain and R+N), and three baselines (BLPA, BEKM, UEU) for Linear GCN model on Cora and CiteSeer datasets.

Dataset	Type	Method	Number of removed edges					
			0	200	400	600	800	1000
Cora	Retrain	Retrain	0.87	0.87	0.87	0.86	0.86	0.86
		R+N	0.84	0.82	0.82	0.81	0.80	0.79
	Unlearn	BLPA	0.58	0.54	0.58	0.58	0.59	0.58
		BEKM	0.65	0.64	0.70	0.70	0.70	0.70
		UEU	0.87	0.87	0.87	0.86	0.86	0.86
		CEU	0.84	0.83	0.82	0.81	0.80	0.79
CiteSeer	Retrain	Retrain	0.77	0.77	0.77	0.76	0.76	0.76
		R+N	0.75	0.75	0.75	0.75	0.75	0.75
	Unlearn	BLPA	0.69	0.69	0.69	0.69	0.69	0.69
		BEKM	0.72	0.72	0.72	0.72	0.72	0.72
		UEU	0.77	0.77	0.77	0.77	0.76	0.76
		CEU	0.75	0.75	0.75	0.74	0.75	0.75

unlearned models at each batch. We observe that, first, the accuracy of the unlearned model remains close to the retrained model at each batch during sequential removals. Second, the performance of the unlearned model after removing k batches stays close to that of the model after single-batch unlearning. These results demonstrate that CEU can handle sequential deletion of multiple batches of edges.

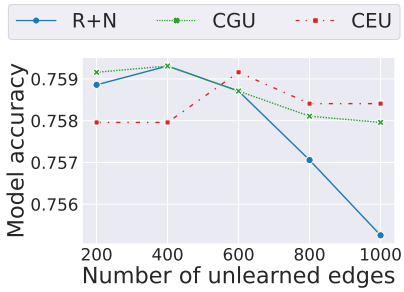
E ADDITIONAL EXPERIMENTAL RESULTS

E.1 Additional Results of Performance for Linear GCN Model

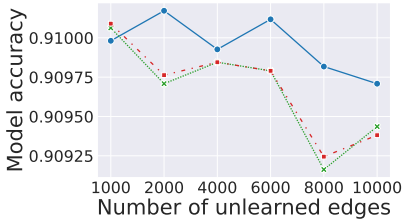
Table 7 reports the model accuracy of GCN model on Cora and CiteSeer datasets. Similar to Table 1, we first observe the model accuracy obtained by CEU stays very close to that of the retrained model, regardless of the number of removed edges. The difference in model accuracy between the retrained and unlearned models remains negligible (in the range of [0.06%, 0.57%] and [0.02%, 0.15%]] on Cora and CiteSeer, respectively). Second, in terms of comparison with both exact unlearning baselines (BEKM, BLPA), the model accuracy by CEU is significantly higher than these two baselines in all the settings. For example, when removing 200 edges from the Cora dataset with GCN as the target model, both BEKM and BLPA only can deliver model accuracy of around 0.53 and 0.64, while CEU can deliver model accuracy of around 0.81. This demonstrates the weakness of the exact unlearning through graph partitioning - breaking the graph structure can bring non-negligible model accuracy loss. Third, regarding the comparison with the approximate unlearning baseline (UEU), CEU has very similar model accuracy, although UEU does not add perturbation to the model loss function. This demonstrates that CEU addresses the trade-off between privacy and model accuracy—it can deliver a provable unlearning guarantee while requiring negligible sacrifice on model accuracy.

E.2 Additional Results: CGU versus CEU

Model accuracy. Figure 9 presents the model accuracy of both CGU [10] and CEU for Citeseer and CS datasets. The observations are similar to those in Figure 5 (a); thus we omit the details.



(a) CiteSeer



(b) CS

Figure 9: Model accuracy: CGU [10] vs. CEU for SGC model.

Table 8: Unlearning efficacy: CGU versus CEU.

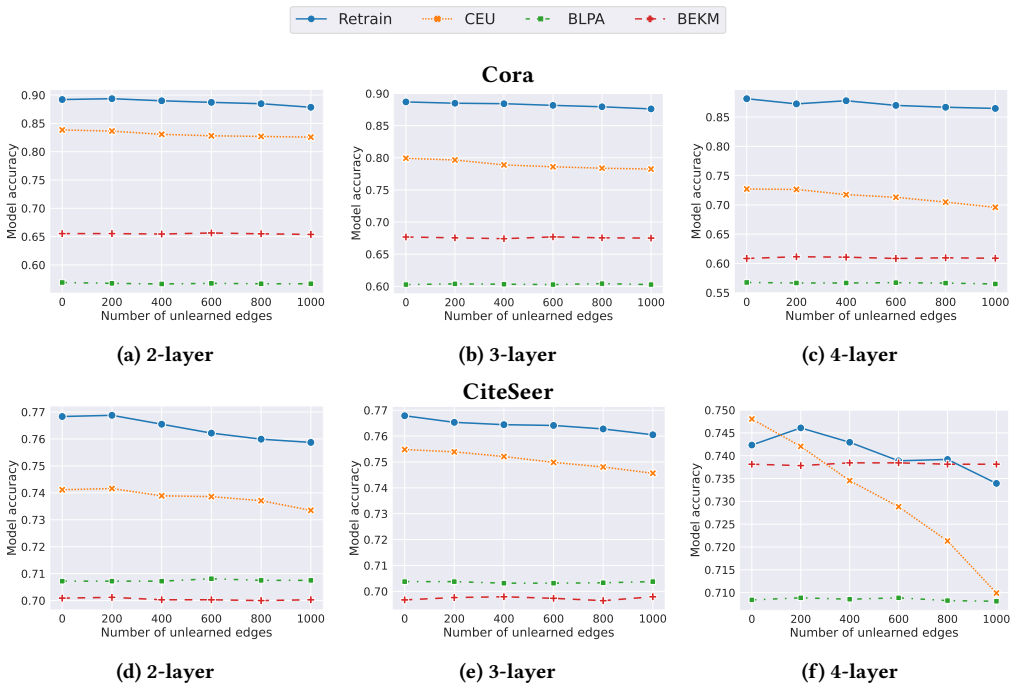
$\lvert E_{\text{UL}}\rvert$	Original	R+N	CGU	CEU
200	0.952	0.622	0.605	0.598
400	0.951	0.623	0.618	0.620
600	0.952	0.616	0.617	0.622
800	0.951	0.627	0.625	0.624
1000	0.950	0.623	0.625	0.618

Unlearning efficacy. Table 8 shows the unlearning efficacy results by both CGU and CEU, where unlearning efficacy is measured as the accuracy (AUC) of the membership inference attack (StealLink [20]). We observe that CGU and CEU have comparable unlearning efficacy. This demonstrates empirically that CEU provides similar unlearning efficacy as CGU.

E.3 Additional Results of Unlearning Performance for Deep GNN Models

Figures 10, 11, and 12 show the model accuracy of GCN, GraphSAGE, and GIN for various complexity (2-, 3-, and 4-layer) respectively. Similar to the observations in Figure 6, despite the model accuracy drops for both retrained and unlearned models of higher complexity, the model accuracy of the unlearned model remains close to that of the retrained model. The largest difference between model accuracy is only around 5% (Figure 10 (f)). Secondly, CEU outperforms two baselines (BLPA and BEKM) in terms of model accuracy for all the settings. For example, the model accuracy of CEU on the 4-layer GIN is 29% higher than BLPA when removing 1000 edges.

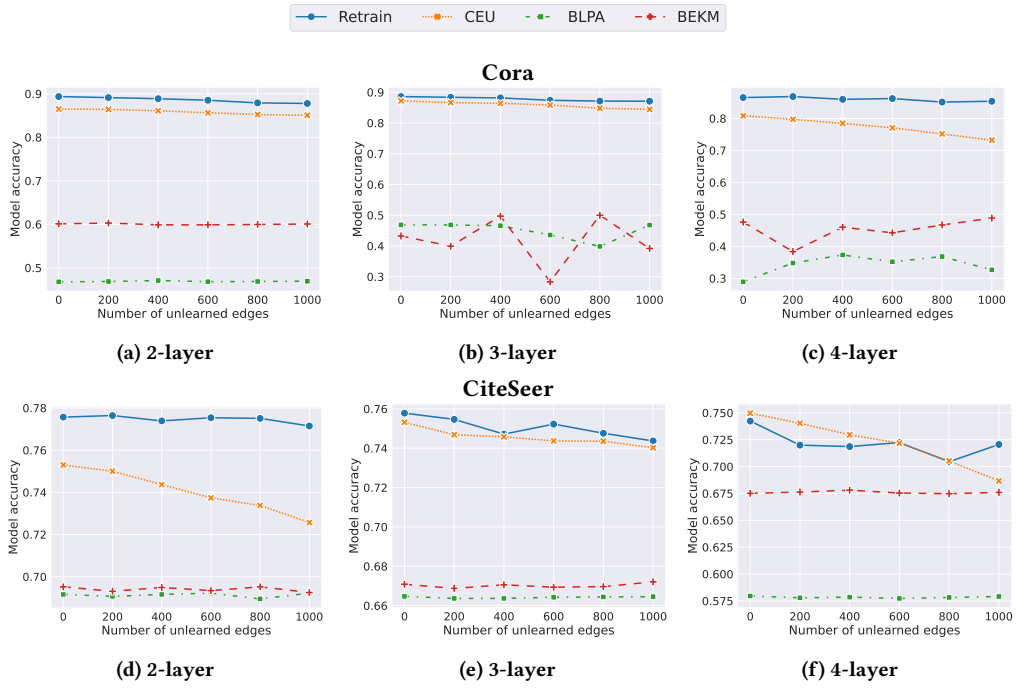
Figure 13 reports the running time of retraining and CEU on GCN models with Cora and CiteSeer datasets. We have similar observations as in Figure 7. First, although the running time for



(a) 2-layer (b) 3-layer (c) 4-layer

(d) 2-layer (e) 3-layer (f) 4-layer

Figure 10: Model accuracy of CEU and retraining on deep GCN models.

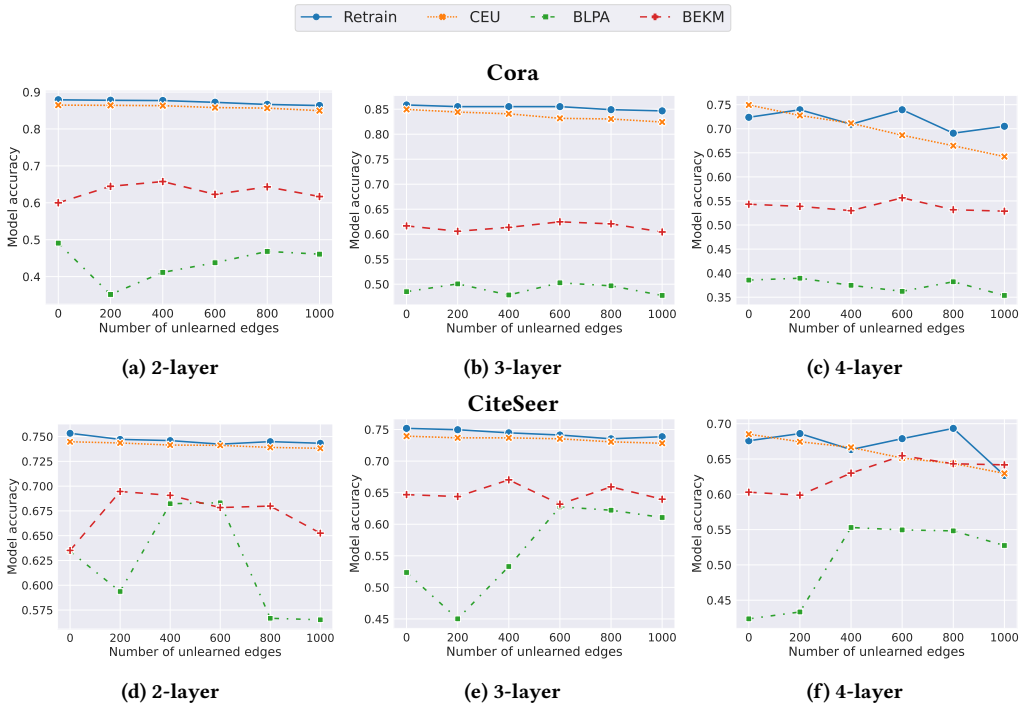


(a) 2-layer (b) 3-layer (c) 4-layer

(d) 2-layer (e) 3-layer (f) 4-layer

Figure 11: Model accuracy of CEU and retraining on GraphSAGE.

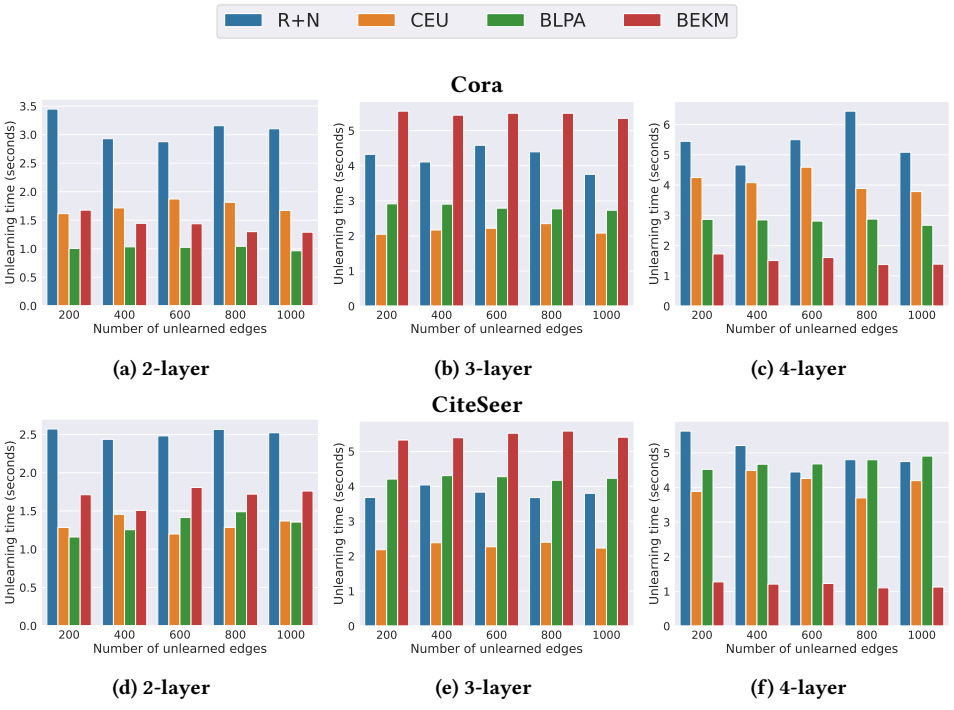
both retraining and CEU grows with the increase in the complexity of GNN models, CEU is always significantly faster than retraining in all the settings, with the speedup factor as large as 2.1X. Furthermore, we observe that CEU is faster than the two baselines of



(a) 2-layer (b) 3-layer (c) 4-layer

(d) 2-layer (e) 3-layer (f) 4-layer

Figure 12: Model accuracy of CEU and retraining on GIN.



(a) 2-layer (b) 3-layer (c) 4-layer

(d) 2-layer (e) 3-layer (f) 4-layer

Figure 13: Time performance of retraining and CEU on deep GCN models (Cora and CiteSeer datasets).

Table 9: Unlearning efficacy for deep GCN on Cora.

Setting	$\|E_{UL}\|$	2-layer			3-layer			4-layer		
		Original	Retrain	CEU	Original	Retrain	CEU	Original	Retrain	CEU
GCN + Cora	200	0.936	0.604	0.587	0.942	0.568	0.570	0.937	0.558	0.562
	400	0.940	0.597	0.599	0.937	0.578	0.569	0.930	0.578	0.559
	600	0.942	0.599	0.589	0.937	0.578	0.571	0.939	0.572	0.563
	800	0.940	0.599	0.594	0.938	0.583	0.574	0.934	0.563	0.560
	1000	0.943	0.602	0.601	0.938	0.573	0.583	0.932	0.573	0.560
GCN + CiteSeer	200	0.954	0.641	0.632	0.953	0.627	0.620	0.951	0.608	0.625
	400	0.954	0.640	0.634	0.952	0.625	0.622	0.951	0.618	0.620
	600	0.958	0.643	0.639	0.953	0.632	0.622	0.957	0.625	0.623
	800	0.955	0.655	0.647	0.954	0.641	0.631	0.953	0.631	0.630
	1000	0.956	0.660	0.650	0.953	0.643	0.638	0.952	0.635	0.635
GraphSAGE + Cora	200	0.938	0.653	0.657	0.949	0.667	0.651	0.952	0.663	0.640
	400	0.946	0.666	0.649	0.948	0.671	0.669	0.954	0.672	0.643
	600	0.948	0.660	0.652	0.947	0.672	0.662	0.951	0.678	0.659
	800	0.945	0.660	0.657	0.947	0.685	0.672	0.952	0.693	0.663
	1000	0.942	0.667	0.657	0.950	0.697	0.681	0.950	0.699	0.676
GraphSAGE + CiteSeer	200	0.960	0.652	0.690	0.963	0.709	0.728	0.970	0.739	0.728
	400	0.959	0.663	0.707	0.966	0.720	0.737	0.968	0.729	0.734
	600	0.960	0.670	0.712	0.965	0.732	0.745	0.965	0.740	0.738
	800	0.955	0.677	0.722	0.964	0.737	0.754	0.968	0.747	0.751
	1000	0.957	0.683	0.724	0.965	0.743	0.757	0.969	0.755	0.759
GIN + Cora	200	0.928	0.596	0.587	0.906	0.592	0.592	0.889	0.576	0.563
	400	0.925	0.603	0.592	0.908	0.600	0.586	0.899	0.592	0.560
	600	0.920	0.612	0.596	0.910	0.606	0.588	0.895	0.598	0.560
	800	0.923	0.613	0.597	0.910	0.603	0.583	0.896	0.586	0.559
	1000	0.924	0.622	0.597	0.912	0.608	0.588	0.894	0.602	0.564
GIN + CiteSeer	200	0.941	0.645	0.613	0.917	0.629	0.606	0.904	0.596	0.597
	400	0.937	0.643	0.628	0.919	0.622	0.610	0.909	0.598	0.599
	600	0.934	0.655	0.639	0.919	0.624	0.609	0.905	0.617	0.597
	800	0.938	0.656	0.637	0.916	0.635	0.616	0.907	0.623	0.599
	1000	0.938	0.660	0.640	0.915	0.641	0.617	0.904	0.629	0.608

exact unlearning (BLPA and BEKM) for most settings on Cora and CiteSeer datasets.

Table 9 presents the unlearning efficacy of the original model, retraining model, and CEU for the three GNN models on Cora and CiteSeer datasets respectively. We have similar observations as in Table 3. We first observe StealLink is highly effective to predict the existence of E_{UL} in the original graph ("Original" column), as the AUC of the attack against the original model is higher than 0.9. Second, the AUC of the attack is noticeably reduced to close to 0.5 for both retrained and unlearned models ("Retrain", and "CEU" columns). This demonstrates that CEU has a similar ability to make the deep GNN models forget the removed edges as retraining.

E.4 Results of Unlearning Performance of Close and Far-Away Edges.

Wendy:This appendix is not referred in the paper. In this section, we present the unlearning performance in terms of model accuracy when removing edges at various distances from the test set. To determine the distance between an edge $e(u, v)$ and a node u', denoted as $dis(e, u')$, we calculate the minimum number of hops required to reach both u and v from u'. Subsequently, for a given edge $e(u, v)$ and a test set G_S, we define the distance between $e(u, v)$ and G_S as $dis(e, G_S)$, representing the minimum distance between e and any node in G_S. We classify the edges in the training graph based on their distance from G_S and select the top-k edges with the highest distance as the "far-away" edges, and the top-k edges with the lowest distance as the "close" edges. Table 10 presents the average results obtained from randomly sampling 10 test sets from the Cora graph. The key observation is that removing close edges leads to a greater impact on the target model accuracy compared to removing far-away edges. This aligns with our expectations since, for GNNs that rely on neighborhood aggregation, a node influences the nodes within its immediate vicinity. Consequently, the removal of far-away edges has a comparatively limited impact on model accuracy compared to the removal of close edges. **Wendy:What**

Table 10: Model accuracy of "close" and "far-away" edges.

	0	100	200	400	600	800	1000	1500	2000
close	0.824	0.822	0.821	0.821	0.820	0.818	0.815	0.815	0.814
far-away	0.806	0.805	0.806	0.806	0.805	0.805	0.805	0.801	0.801

does this last sentence mean? Are you still conducting any experiements?Kun:I removed the sentence.