Efficiently Forgetting What You Have Learned in Graph Representation Learning via Projection

Weilin Cong
The Pennsylvania State University

Mehrdad Mahdavi
The Pennsylvania State University

Abstract

As privacy protection receives much attention, unlearning the effect of a specific node from a pre-trained graph learning model has become equally important. However, due to the *node dependency* in the graph-structured data, representation unlearning in Graph Neural Networks (GNNs) is challenging and less well explored. In this paper, we fill in this gap by first studying the unlearning problem in linear-GNNs, and then introducing its extension to non-linear structures. Given a set of nodes to unlearn, we propose PROJECTOR that unlearns by projecting the weight parameters of the pre-trained model onto a subspace that is irrelevant to features of the nodes to be forgotten. PROJECTOR could overcome the challenges caused by node dependency and enjoys a perfect data removal, i.e., the unlearned model parameters do not contain any information about the unlearned node features which is guaranteed by algorithmic construction. Empirical results on real-world datasets illustrate the effectiveness and efficiency of PROJECTOR. [Code].

1 Introduction

As graph representation learning has achieved great success in real-world applications (e.g., social networks Kipf and Welling (2017); Hamilton et al. (2017), knowledge graphs Wang et al. (2019a,b), and recommender system Berg et al. (2017)), privacy protection in graph representation learning has become equally important. Recently, as "*Right to be forgotten*" gradually implemented in multiple jurisdictions, users are empowered with the right to request any organization or company to remove the effect of their private data from a machine learning model, which is known as "*machine unlearning*". For example, when a Twitter user deletes a post, the user not only may require Twitter to permanently remove the post from their database, but also might require Twitter to eliminate its impact on any machine learning models pre-trained on the deleted post, so as to prevent the private information in the deleted post be inferred by any malicious third party.

Existing unlearning approaches can be roughly classified into exact unlearning and approximate unlearning. The goal of "*exact unlearning*" is to exactly produce the model parameters trained without the deleted data. The most straightforward unlearning approach is to retrain the model from scratch using the remaining data, which could be computationally prohibitive when the dataset is large or even infeasible if not all the data are available to retrain. To avoid re-training on large data, SISA Bourtoule et al. (2021) proposes to split the original dataset into multiple shards and train a model on each data shard, then aggregate their prediction during inference. Upon receiving unlearning requirements, they only need to re-train the specific shard model that the unlearned data belongs to. While being more efficient compared to retraining from scratch, the model performance suffers because each model has fewer data to be trained on and data heterogeneity also deteriorates the performance. To further reduce the computation overhead, "*approximate unlearning*" is proposed to trade-off between the unlearning efficiency and the data removal effectiveness. For example, INFLUENCE Guo et al. (2020) proposes to approximate the unlearned model using first-order Taylor approximation and FISHER Golatkar et al. (2020) proposes to directly fine-tune with Newton's method on the remaining data. Since approximate unlearning methods lack guarantee on whether all information associated with the deleted data is eliminated, it is necessary to inject random noise to model parameters or objective functions to amplify privacy, which could significantly hurt the performance of unlearned model. Employing these methods in graph-structured data is even more challenging due to the dependency among nodes. Motivated by the importance of unlearning graph-structured data, we aim at answering the following questions in the context of GNNs:

Q1. Can existing machine unlearning methods be uti-

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

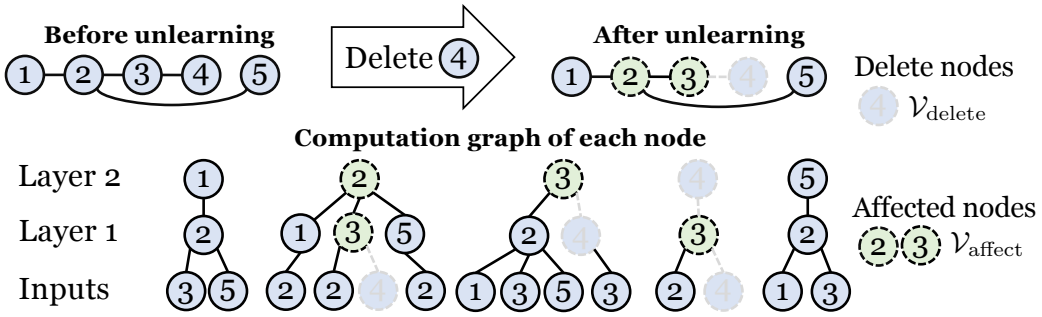


Figure 1: An illustration of how the node representations of a 2-layer GNN (with neighbor average aggregation) are affected after deleting the node v_4 from the graph. After removing node v_4, the node representation of nodes $\{v_2, v_3\}$ are also affected since these nodes require node v_4 to compute their representations. Such dependency grows exponentially with respect to the number of GNN layers.

lized to solve graph unlearning problem? Most of the existing methods are designed for settings where the loss function can be decomposed over individual training samples, and the node dependency in graph-structured data render these methods inapplicable to GNNs and makes them sub-optimal. For example, exact graph unlearning method GRAPHERASER Chen et al. (2021) extends Bourtoule et al. (2021) by partitioning the original graph into multiple subgraphs. However, graph partitioning will result in loosing part of the structure information due to ignorance of the edges that span subgraphs, which could further hurt the model performance. Moreover, the data heterogeneity issue on homophily graphs is more severe because nodes with similar properties/categories are more likely to be partitioned into the same subgraph. Applying approximate unlearning for graph structured data is also non-trivial. For example, most of these methods require "*the objective function before data deletion*" could be formulated as a summation of "*the objective after data deletion*" and "*the loss on deleted data*". However, this is not the case on graph-structured data because the representation of the deleted nodes' multi-hop neighbors $\mathcal{V}_{\text{affect}}$ are also affected after node deletion. Please refer to Figure 1 on how node dependency would affect the GNN models' output after deleting a single node from graph, refer to Appendix C for a detailed mathematical explanation on node dependency. To overcome this issue, we need to update all affected nodes $\mathcal{V}_{\text{affect}}$ in parallel, which results in massive computation overhead because $|\mathcal{V}_{\text{affect}}|$ grows exponentially with the number of layers.

Q2. If not, can we effectively unlearn representations in GNNs in a computationally efficient manner? We propose a projection-based unlearning approach for linear-GNNs that not only "*bypasses the node dependency issue*" but also "*enjoys a perfect data removal guarantee*". More specifically, we propose to unlearn node features by orthogonal projecting linear-GNN's weight parameters to a subspace that is irrelevant to the unlearned node features (Section 3). The projection step guarantees our weight parameters do not carry any information about the deleted node features, please refer to Figure 2 for an illustration of our main idea. PROJECTOR could bypass the node dependency issue because the graph convolutions in linear-GNN can be re-formulated as a linear combination of the input node features and the projection-step is directly applied to the node features (Section 3.2). Notice that this is different from most approximate unlearning approaches because their gradient and Hessian are computed on the output of GNN models, therefore they are affected by the node dependency.

Q3. How to assess the effectiveness of unlearning in GNNs? We consider two criteria to evaluate the effectiveness of unlearning. Our first criterion is "*the distance between the unlearned weights to the exactly retrained weights*". We evaluate this criterion by theoretically upper bound the distance of two models. We show that PROJECTOR enjoys a tighter upper bound than approximate unlearning methods Guo et al. (2020); Golatkar et al. (2020) (Section 3.3). Although this criterion has become the de facto way to measure the success of unlearning for approximate unlearning methods, it has been pointed out by Thudi et al. (2021) that we cannot infer "whether the data have been deleted" solely from it. Our theoretical explanation on this point is deferred to the Appendix E. Therefore, we introduce our second criterion by checking "*whether unlearned weights contain the deleted node features*". To achieve this, we introduce "feature injection test" in the experiment section to rigorously verify this criterion.

Contributions. The main contributions of the present paper are summarized as follows:

- We propose an efficient graph representation unlearning method PROJECTOR, which could overcome the node dependency issue and is guaranteed to remove the trace of the deleted node features (Section 3.2).
- We theoretically show that *unlearned model* of PROJECTOR is closer to the *model retrained from scratch* than other approximate unlearning methods, which indicates that PROJECTOR is more preferred if only approximate unlearning is required (Section 3.3).

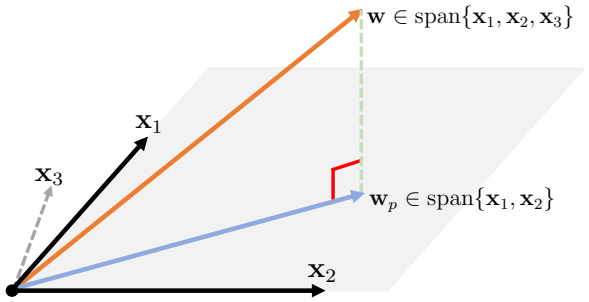


Figure 2: The orthogonal projection unlearning in PROJECTOR. The original weight $\mathbf{w}$ exists inside the subspace defined by node feature vectors $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$. We can unlearn $\mathbf{x}_3$ and obtain the new weight $\mathbf{w}_p$ by projecting $\mathbf{w}$ onto the subspace defined without $\mathbf{x}_3$.

- To improve the expressive of the linear-GNN used with PROJECTOR, we introduce two unlearning-favorable extension, i.e., non-linearity extension and adaptive diffusion graph convolution (Section 3.4).
- We introduce the "feature injection test" to rigorously verify whether an unlearning method could perfectly remove the trace of the deleted node features. Our results show that PROJECTOR could perfectly remove the trace of the deleted node features, however, other approximate unlearning methods cannot, which emphasizes the importance of PROJECTOR (Section 4).
- Empirical results on large-scale real-world datasets of different sizes illustrate the effectiveness, efficiency, and robustness of PROJECTOR (Section 4 and Appendix A).

2 Related work and backgrounds

Exact unlearning. The most straightforward way is to retrain the model from scratch, which is computationally demanding, except for some model-specific problems such as SVM Cauwenberghs and Poggio (2000), K-means Ginart et al. (2019), and decision tree Brophy and Lowd (2021). To reduce the computation cost, Bourtoule et al. (2021) proposes to split the dataset into multiple shards and train an independent model on each data shard, then aggregate their prediction during inference. A similar idea is explored in Aldaghri et al. (2021); He et al. (2021). GRAPHERASER Chen et al. (2021) extends Bourtoule et al. (2021) to graph-structured data by proposing a graph partition method that can preserve the structural information as much as possible and weighted prediction aggregation for evaluation. Chen et al. (2022) further generalize Chen et al. (2021) to the recommender system. Although the data partition schema allows for a more efficient retrain of models on a smaller fragment of data, the model performance suffers because each model has fewer data to be trained on and data heterogeneity can also deteriorate the performance. Moreover, if a large set of deleted nodes are selected at random, it could still result in massive retraining efforts. Ullah et al. (2021) proposes to retrain at the iteration that deleted data the first time appears, which is not suitable if it requires iterating the full dataset multiple rounds. Neel et al. (2020); Ullah et al. (2021); Sekhari et al. (2021) study the unlearning from the generalization theory perspective, Fu et al. (2022); Nguyen et al. (2022) study unlearning for Bayesian inference, which is orthogonal to the main focus of this paper.

Approximate unlearning. The main idea is to approximate the model trained without the deleted data in the parameter space. For example, Guo et al. (2020) proposes to unlearn by removing the influence of the deleted data on the model parameters by first-order Taylor approximation, where the Hessian is computed on the remaining data and gradient is computed on the deleted data. Chien et al. (2022) generalize the analysis in Guo et al. (2020) to graph. A similar idea has been explored in Wu et al. (2022) but requires an objective function as a finite-sum formulation, which is non-trivial to extend onto graph-structured data. Golatkar et al. (2020) performs Fisher forgetting by taking a single step of Newton's method on the remaining training data. Golatkar et al. (2021) generalizes the idea to deep neural networks by assuming a subset of training samples are never forgotten, which can be used to pre-train a neural network as a feature extractor and only unlearn the last layer. Izzo et al. (2021) speeds up Guo et al. (2020) by using the leave-one-out residuals for the linear model update, which reduces the time complexity to linear in the dimension of the deleted data and is independent of the size of the dataset. Wu et al. (2020a) proposes to first save all the intermediate weight parameters and gradients during training, then utilize such information to efficiently estimate the optimization path. Similar idea have been explored in Wu et al. (2020b) for logistic regression. Notice that due to the nature of approximate unlearning, these methods only approximately unlearn the information of deleted data, require adding random noise, and lack of perfect data removal guarantee in practice Thudi et al. (2021).

Linearity requirement in unlearning. Linearity is required in most unlearning methods Guo et al. (2020); Golatkar et al. (2020); Wu et al. (2020a) to verify whether the trace of deleted data has been perfectly unlearned. Unless re-training from scratch, it is still an open problem to theoretically or rigorously empirically verify this in the non-linear models Thudi et al. (2021); Guo et al. (2020). Therefore, we initiate our study on linear-GNNs in Section 3.2 and provide its non-linearity extension in Section 3.4. We will rigorously test whether the information is perfectly unlearned on linear-GNNs and demonstrate the application of using PROJECTOR with non-linear GNNs.

Relation between unlearning and differential privacy. Unlearning and differential privacy (DP) are two concepts that could be used in parallel. More specifically, *DP* aims to prevent the privacy leakage issue, while *unlearning* seeks to remove some data points' effect on the pre-trained model.

Recently, a number of approximate unlearning methods Guo et al. (2020); Golatkar et al. (2020); Chien et al. (2022) are inspired by DP to unlearn by injecting random noises and derive an approximate unlearning DP-like upper bound. However, not all unlearning methods require using random noises and could be evaluated under a DP-like framework. For example, Ullah et al. (2021); Chen et al. (2021) unlearn by re-training from scratch and PROJECTOR unlearns by orthogonal projection, therefore adding random noise is not required. Please refer to Appendix D for more details. In this paper, we only consider fully removing the trace of data from the model by unlearning, but do not consider preventing the privacy leakage issue with DP.

3 Graph representation unlearning

We first introduce backgrounds on graph learning and unlearning in Section 3.1. Then, we introduce our graph representation unlearning approach PROJECTOR on linear-GNN in Section 3.2 and theoretically analyzing its effectiveness in Section 3.3. Finally, we introduce PROJECTOR's non-linearity extension in Section 3.4.

3.1 Backgrounds

We consider solving semi-supervised binary node classification using the linear-GNN, which could be easily extended to multi-class classification. More specifically, given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges, let us suppose each node $v_i \in \mathcal{V}$ is associated with a node feature vector $\mathbf{x}_i \in \mathbb{R}^d$. Let $\mathbf{A}, \mathbf{D} \in \mathbb{R}^{n\times n}$ denote the adjacency matrix and its associated degree matrix. Then, an L-layer linear-GNN[1] computes the node representation $\mathbf{H} = \mathbf{P}^L\mathbf{X} \in \mathbb{R}^{n\times d}$ by applying L propagation matrices $\mathbf{P} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ to the node features matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$. During training, only training set nodes $\mathcal{V}_{\text{train}} \subset \mathcal{V}$ are labeled by a binary label $y_i \in \{-1, +1\}$, our goal is to estimate the label of the unlabeled nodes $\mathcal{V}_{\text{eval}} = \mathcal{V} \setminus \mathcal{V}_{\text{train}}$. More specifically, we want to find the weight parameters $\mathbf{w} \in \mathbb{R}^d$ that minimize

$$F(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{\mathcal{V}_{\text{train}}}\sum_{v_i\in\mathcal{V}_{\text{train}}} f_i(\mathbf{w}),$$
$$f_i(\mathbf{w}) = \log\left(1 + \exp(-y_i\mathbf{w}^\top\mathbf{h}_i)\right), \mathbf{h}_i = [\mathbf{P}^L\mathbf{X}]_i. \tag{1}$$

For graph representation unlearning, let $\mathcal{V}_{\text{delete}} \subset \mathcal{V}_{\text{train}}$ denote the set of deleted nodes and $\mathcal{V}_{\text{remain}} = \mathcal{V}_{\text{train}} \setminus \mathcal{V}_{\text{delete}}$ denote the remaining nodes. Our goal is to unlearn the node feature information $\{\mathbf{x}_i \mid v_i \in \mathcal{V}_{\text{delete}}\}$ of the deleted nodes $\mathcal{V}_{\text{delete}}$. In terms of the notations, we denote $\mathbf{w}$ as the solution before unlearning, $\mathbf{w}_p$ as the solution obtained by PROJECTOR, and $\mathbf{w}_u$ as the solution obtained by re-training from scratch on the dataset without the deleted nodes.

[1]Non-linear GNNs usually add activation function and weight matrix after each graph convolution. For example, the GCN's hidden representation is computed by $\mathbf{H}^{(\ell)} = \sigma(\mathbf{P}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)})$.

3.2 Graph representation unlearning via PROJECTOR

The main idea behind PROJECTOR is as follows: "*If the weight parameters of linear-GNN are located inside the linear span of all node features (precondition), then we can unlearn a set of node features by projecting the weight parameters onto a subspace that is irrelevant to the node features that we want to unlearn (how to unlearn).*" In the following, we will first explain why the precondition holds in linear-GNNs, then introduce how to unlearn, and explain why PROJECTOR can bypass the node dependency.

Why precondition holds in linear-GNN? The precondition holds because the graph convolution in linear-GNN is a linear operator on node features. As a result, all gradients are inside the linear span of all node features. Therefore, if we optimizing linear-GNN (Eq. 1) using SGD with weight initialization satisfying $\mathbf{w}_{\text{init}} \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, regardless of how many steps of gradient updates, we still have $\mathbf{w} \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ holds. To see this, let us first recall that the gradient of Eq. 1 with respect to any $\mathbf{w}$ is

$$\nabla F(\mathbf{w}) = \lambda\mathbf{w} + \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{j\in\mathcal{V}_{\text{train}}} \nu_j\mathbf{x}_j,$$
$$\nu_j \underset{(a)}{=} \sum_{i\in\mathcal{V}_{\text{train}}} \mu_i[\mathbf{P}^L]_{ij},\ \mu_i = -y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i), \tag{2}$$

where $[\mathbf{P}^L]_{ij}$ denotes the i-th row j-th column of $\mathbf{P}^L$ and $\sigma(\cdot)$ is the Sigmoid function. Then, Eq. 2 implies that the gradient $F(\mathbf{w})$ is inside the linear span of all node features, i.e., $\nabla F(\mathbf{w}) \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Therefore, when using gradient update rule $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\nabla F(\mathbf{w}_t)$, the weight after gradient updates still stays inside the same subspace defined by the linear span of all node features.

How to unlearn? Recall that our goal is to unlearn node features $\mathbf{X}_{\text{delete}} = \{\mathbf{x}_i \mid v_i \in \mathcal{V}_{\text{delete}}\}$ of size $m = |\mathcal{V}_{\text{delete}}|$ by making sure the unlearned solution does not carry any information about $\mathbf{X}_{\text{delete}}$. This can be achieved by finding an alternative solution $\mathbf{w}_p$ from a subspace that is irrelevant to $\mathbf{X}_{\text{delete}}$. Meanwhile, we hope $\mathbf{w}_p$ is close to $\mathbf{w}$ because small changes in the input data are expected to lead to small changes in the optimal solutions. Formally, let us define $\mathcal{U} = \text{span}\{\mathbf{x}_i \mid v_i \in \mathcal{V}_{\text{remain}}\}$ as the linear subspace spanned by all remaining samples and our goal is to find $\mathbf{w}_p = \arg\min_{\mathbf{v}\in\mathcal{U}} \|\mathbf{v} - \mathbf{w}\|_2^2$. Because the vertical distance is the shortest, we can obtain $\mathbf{w}_p$ by orthogonal projecting $\mathbf{w}$ onto the subspace $\mathcal{U}$. Knowing that any projection $\Pi_{\mathcal{U}}(\mathbf{w})$ onto $\mathcal{U}$ is necessarily an element of $\mathcal{U}$, i.e., $\Pi_{\mathcal{U}}(\mathbf{w}) \in \mathcal{U}$, the results after orthogonal projection can be represented as a weighted combination of all remaining node features $\mathbf{w}_p = \Pi_{\mathcal{U}}(\mathbf{w}) = \sum_{v_i\in\mathcal{V}_{\text{remain}}} \alpha_i\mathbf{x}_i$, where the coefficients of the orthogonal projection $\boldsymbol{\alpha}$ is derived in Proposition 1. An

Algorithm 1 PROJECTOR to unlearning linear-GNN

Require: The pre-trained parameters $\mathbf{w}$, (*Option 1*) remain nodes' features $\mathbf{X}_{\text{remain}}$, (*Option 2*) deleted node features $\mathbf{X}_{\text{delete}}$, pre-computed $\mathbf{M} = \mathbf{X}^\top\mathbf{X}$ and $\mathbf{M}^\dagger = (\mathbf{X}^\top\mathbf{X})^\dagger$
Ensure: Unlearned weight parameters $\mathbf{w}_p$
if (*Option 1*) $\mathbf{X}_{\text{remain}}$ is available **then**
Compute $\mathbf{M}_{\text{remain}}$ by

$$\mathbf{M}_{\text{remain}} = \mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}}$$

Compute $\mathbf{M}_{\text{remain}}^\dagger$ by

$$\mathbf{M}_{\text{remain}}^\dagger = (\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger$$

else if (*Option 2*) $\mathbf{X}_{\text{delete}}, \mathbf{M}, \mathbf{M}^\dagger$ are available **then**
Compute $\mathbf{M}_{\text{remain}}$ by

$$\mathbf{M}_{\text{remain}} = \mathbf{M} - \mathbf{X}_{\text{delete}}^\top\mathbf{X}_{\text{delete}}$$

Compute $\mathbf{M}_{\text{remain}}^\dagger$ by

$$\mathbf{S} = \mathbf{X}_{\text{delete}}^\top[\mathbf{I} - \mathbf{X}_{\text{delete}}\mathbf{X}_{\text{delete}}^\top]\mathbf{X}_{\text{delete}}$$
$$\mathbf{M}_{\text{remain}}^\dagger = \mathbf{M}^\dagger + \mathbf{M}^\dagger\mathbf{S}\mathbf{M}^\dagger$$

end if
Compute $\mathbf{w}_p = \mathbf{M}_{\text{remain}}\mathbf{M}_{\text{remain}}^\dagger\mathbf{w}$ as final output

illustration of the projection-based unlearning is shown in Figure 2 and the proof is provided in Appendix F.

Proposition 1 *The coefficients of the orthogonal projection is computed as* $\boldsymbol{\alpha} = \mathbf{X}_{remain}(\mathbf{X}_{remain}^\top\mathbf{X}_{remain})^\dagger\mathbf{w}$*, where* $\mathbf{X}_{remain} = \{\mathbf{x}_j \mid v_j \in \mathcal{V}_{remain}\}$ *is the remaining node features and* $\dagger$ *is the pseudo-inverse operator.*

The significant computation required in Proposition 1 includes computing $\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}} \in \mathbb{R}^{d\times d}$ and its inverse with $\mathcal{O}(rd^2)$ and $\mathcal{O}(d^3)$ computation complexity, where $r = |\mathcal{V}_{\text{remain}}|$ is the size of remaining nodes and d is node feature dimension. However, if we could pre-computed $\mathbf{X}^\top\mathbf{X}$ before the unlearning requests arrive, then we could efficiently compute $\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}} = \mathbf{X}^\top\mathbf{X} - \mathbf{X}_{\text{delete}}^\top\mathbf{X}_{\text{delete}}$ and compute $(\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger$ by applying the Woodbury identity Golub and Van Loan (2013) on $\mathbf{X}_{\text{delete}}^\top\mathbf{X}_{\text{delete}}$, $\mathbf{X}^\top\mathbf{X}$, which leads to a lower computation complexity of $\mathcal{O}(\max\{m^3, md^2\})$ if $m < \max\{r, d\}$. After obtaining $\boldsymbol{\alpha}$, PROJECTOR computes the unlearned weight parameters by $\mathbf{w}_p = \mathbf{X}_{\text{remain}}^\top\boldsymbol{\alpha}$. Intuitively, the projection step in PROJECTOR could be thought of as a re-weighting on the remaining nodes, which allows our model to behave as close to the model before unlearning as possible, but without carrying any information about the deleted node features. Therefore, the output of PROJECTOR could be interpreted as re-training on the remaining graph under some unknown importance sampling distribution.

To this end, we summarize PROJECTOR in Algorithm 1, where two different types of input options are available that lead to identical results. More specifically, we can use *option 1* if only remaining node features are available, otherwise we can use *option 2* if only the features of deleted nodes are available but pre-computing is feasible. Besides, due to the similarity between logistic regression and SVM, PROJECTOR could also be used in primal-based SVM unlearning Chu et al. (2015) to alleviate the high computation cost of the dual-based SVM unlearning approach Cauwenberghs and Poggio (2000). Readers could refer to Appendix J for more details on its application to SVM.

Why node dependency is bypassed? From Eq. 2 (a), we could tell that node dependencies in $\mathbf{P}$ are included inside the finite sum weight μ_j, which is a constant that is multiplied with its features $\mathbf{x}_j$. PROJECTOR could bypass the node dependency because our projection-step is directly applied to the input node features, instead of the final outputs of GNNs. This is not the case for most approximate unlearning methods, e.g., Guo et al. (2020); Golatkar et al. (2020); Wu et al. (2020a), because their unlearning requires computing the gradient or Hessian on the final layer outputs.

Extension to multi-class classification. Please notice that PROJECTOR also works with cross-entropy loss for multi-class classification. To see this, let us consider C categories and N data but without considering the node dependency for simplicity, i.e., optimizing $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_C] \in \mathbb{R}^{C\times d}$ on $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ where $\mathbf{w}_c$ is the c-th row of $\mathbf{W}$. Then, the softmax's c-th class probability computed on $\mathbf{x}_n$ is

$$p_{n,c} = \frac{\exp(a_{n,c})}{\sum_{i=1}^C \exp(a_{n,i})},\ a_{n,c} = \mathbf{w}_c^\top\mathbf{x}_n.$$

We define the objective function as

$$L_{\mathbf{W}} = -\sum_{n=1}^N\sum_{c=1}^C y_{n,c}\log(p_{n,c}),$$

then its gradient is

$$\frac{\partial L_{\mathbf{W}}}{\partial \mathbf{w}_c} = \sum_{n=1}^N\sum_{i=1}^C \frac{\partial L_{\mathbf{W}}}{\partial a_{n,i}}\frac{\partial a_{n,i}}{\partial \mathbf{w}_c} = \sum_{n=1}^N (p_{n,c} - y_{n,c})\mathbf{x}_n$$

because

$$\frac{\partial L_{\mathbf{W}}}{\partial a_{n,i}} = p_{n,i} - y_{n,i} \text{ and } \frac{\partial a_{n,i}}{\partial \mathbf{w}_c} = \begin{cases}\mathbf{x}_n & \text{if } i = c\\ \mathbf{0} & \text{if } i \neq c.\end{cases}$$

As a result, for any $j \in [C]$ we have

$$\frac{\partial L_{\mathbf{W}}}{\partial \mathbf{w}_j} \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_N\},$$

which means each row of the $\mathbf{W}$ is in the span of all node features, and we can apply PROJECTOR on each row of $\mathbf{W}$ independently to unlearn.

3.3 On the effectiveness of PROJECTOR

In this section, we study the effectiveness of PROJECTOR by measuring the ℓ_2-norm on the difference between PROJECTOR's unlearned solution $\mathbf{w}_p$ to the solution obtained by retraining from scratch $\mathbf{w}_u$ on the dataset without the deleted nodes, and we are expecting $\|\mathbf{w}_p - \mathbf{w}_u\|_2$ to be small for good unlearning methods. For unlearning, we suppose a random subset of nodes $\mathcal{V}_{\text{delete}} \subset \mathcal{V}_{\text{train}}$ are selected and the remaining nodes are denoted as $\mathcal{V}_{\text{remain}} = \mathcal{V}_{\text{train}} \setminus \mathcal{V}_{\text{delete}}$. Since removing the nodes $\mathcal{V}_{\text{delete}}$ is the same as updating the propagation matrix from $\mathbf{P}$ to $\mathbf{P}_u$, where all edges that are connected to node $v_i \in \mathcal{V}_{\text{delete}}$ are removed in $\mathbf{P}_u$, we can write down the objective after data deletion $F^u(\mathbf{w}_u)$ as

$$
\begin{aligned}
F^u(\mathbf{w}_u) &= \frac{1}{|\mathcal{V}_{\text{remain}}|} \sum_{v_i \in \mathcal{V}_{\text{remain}}} f_i^u(\mathbf{w}_u), \\
f_i^u(\mathbf{w}_u) &= \log\left(1 + \exp(-y_i \mathbf{w}_u^\top \mathbf{h}_i^u)\right) + \lambda \|\mathbf{w}_u\|_2,
\end{aligned} \tag{3}
$$

where $\mathbf{h}_i^u = [\mathbf{P}_u^L \mathbf{X}]_i$.

Before proceeding to our result, we make the following customery assumptions on graph propagation matrices, node features, and weight parameters in Assumption 1, on the variance of stochastic gradients in Assumption 2, and on the correlation between node feature in Assumption 3. Please notice that Assumption 1, 2 are standard assumptions in GNN's theoretical analysis Cong et al. (2021); Ramezani et al. (2022) and Assumption 3 is a mild assumption that could be empirically verified in Table 3 on real-world dataset, where δ could be think of as a measurement on the closeness of the subspace defined with and without the deleted node features. In practice, δ is small if only a small amount of nodes are removed from the original graph.

Assumption 1 *We assume each row of the propagation matrices before and after node deletion is bounded by $P_s \geq 0$, i.e., $\max_j \left\|[\mathbf{P}^L]_j\right\|_2 \leq P_s$, $\max_j \left\|[\mathbf{P}_u^L]_j\right\|_2 \leq P_s$. Besides, we assume each row of the difference of the propagation matrices before and after data deletion is bounded by $P_d \geq 0$, i.e., $\max_j \left\|[\mathbf{P}_u^L - \mathbf{P}^L]_j\right\|_2 \leq P_d$. Furthermore, we assume the norm of any node features $\mathbf{x}_i$, $v_i \in \mathcal{V}$ and weight parameters $\mathbf{w}$ are bounded by $B_x, B_w \geq 0$, i.e., $\|\mathbf{x}_i\|_2 \leq B_x$, $\|\mathbf{w}\|_2 \leq B_w$.*

Assumption 2 *For any deleted nodes $\mathcal{V}_{delete}$, the gradient variance computed on the remaining nodes $\mathcal{V}_{remain} = \mathcal{V} \setminus \mathcal{V}_{delete}$ can be upper bounded by $G \geq 0$, i.e., we have $\mathbb{E}_{\mathcal{V}_{delete}}[\|\mathbf{g} - \tilde{\mathbf{g}}\|_2] \leq G$, where $\mathbf{g} = \frac{1}{|\mathcal{V}_{train}|} \sum_{v_i \in \mathcal{V}_{train}} \nabla f_i^u(\mathbf{w})$ and $\tilde{\mathbf{g}} = \frac{1}{|\mathcal{V}_{remain}|} \sum_{v_i \in \mathcal{V}_{remain}} \nabla f_i^u(\mathbf{w})$ for any $\mathbf{w}$.*

Assumption 3 *For any node $v_j \in \mathcal{V}_{delete}$, its node feature $\mathbf{x}_j$ can be approximated by the linear combination of all node features in the remaining node set $\{\mathbf{x}_i \mid v_i \in \mathcal{V}_{remain}\}$ up to an error $\delta \geq 0$. Formally, we have $\max_{v_j \in \mathcal{V}_{delete}} \min_{\boldsymbol{\alpha}} \left\|\sum_{i \in \mathcal{V}_{remain}} \alpha_i \mathbf{x}_i - \mathbf{x}_j\right\|_2 \leq \delta$.*

To this end, let us introduce our main results. From Theorem 1, we know that $\|\mathbf{w}_p - \mathbf{w}_u\|_2$ is mainly controlled by three key factors: ① the difference between the propagation matrices before and after data deletion, which is captured by P_d in Assumption 1; ② the variance of stochastic gradient computed on the remaining nodes, which is captured by G in Assumption 2; ③ the closeness of any deleted node features that could be approximated by weighted combination of all node features in the remaining node sets, which is captured by δ in Assumption 3. By reducing the number of nodes in $\mathcal{V}_{\text{delete}}$, all P_d, δ, G are expected to decrease. At an extreme case with $|\mathcal{V}_{\text{delete}}| = 0$, we have $P_d = \delta = G = 0$ and $\mathbf{w}_p = \mathbf{w} = \mathbf{w}_u$. The proof is deferred to Appendix G.

Theorem 1 *Let us suppose Assumptions 1,2,3 hold. Let us define $\mathbf{w}_p$ as the solution obtained by* PROJECTOR*, $\mathbf{w}_u$ is the solution obtained by re-training from scratch with objective function $F^u(\mathbf{w})$, and we assume $\mathbf{w}_u$ is well trained such that $\mathbf{w}_u \approx \arg\min_{\mathbf{w}} F^u(\mathbf{w})$. Then, the closeness of $\mathbf{w}_p$ to the weight parameters $\mathbf{w}_u$ can be bounded by*

$$
\begin{aligned}
&\mathbb{E}_{\mathcal{V}_{delete}}[\|\mathbf{w}_u - \mathbf{w}_p\|_2] \leq \Delta = \\
&Q \sum_{t=1}^{T} \left(1 + \eta(\lambda + B_x^2 P_s^2)\right)^{t-1} + \delta \eta T \times |\mathcal{V}_{delete}|,
\end{aligned} \tag{4}
$$

where $Q = \eta\big((1 + B_x B_w P_s) B_x P_d + G\big)$ and η is the learning rate used to pre-train the weight $\mathbf{w}$ for T steps of gradient descent updates. After projection, we can fine-tune $\mathbf{w}_p$ for K iterations with learning rate $(\lambda + B_x^2 P_s^2)^{-1}$ to obtain $\tilde{\mathbf{w}}_p$ that has an error $F^u(\tilde{\mathbf{w}}_p) - \min_{\mathbf{w}} F^u(\mathbf{w}) \leq \mathcal{O}((\lambda + B_x^2 P_s^2)\Delta / K)$.

Besides, we know the solution of PROJECTOR is probably closer to the model retrained from scratch compared to Guo et al. (2020); Golatkar et al. (2020) if δ satisfies the condition in Proposition 2. In practice, the condition is very likely to be satisfied because learning rate η, regularization term λ, and the ratio of deleted nodes $|\mathcal{V}_{\text{delete}}|/|\mathcal{V}|$ are usually very small. For example, a common choice of learning rate and regularization is $\eta = 0.01, \lambda = 10^{-6}$ for most model training. Moreover, we empirically validate the difference between the weight before and after unlearning in the experiment section to validate our theoretical results. The proof of Proposition 2 is deferred to Appendix H.

Proposition 2 *If the approximation error in Assumption 3 satisfying $\delta < \left((\lambda \eta T)^{-1} + 1\right) B_x \times \frac{|\mathcal{V}|}{|\mathcal{V}_{delete}|}$, then* PROJECTOR*'s output is provably closer to re-training from scratch then using approximate unlearning* INFLUENCE *Guo et al. (2020) and* FISHER *Golatkar et al. (2020).*

3.4 Toward a more powerful structure

To boost the model performance PROJECTOR, we first introduce an unlearning-favorable non-linearity extension to help PROJECTOR better leverage node feature information,

then we introduce an unlearning favorable adaptive diffusion graph convolution to help PROJECTOR better leverage the graph structure information.

An extension from linear to non-linear. Recall that the geometric view of solving logistic regression is finding a hyperplane to linearly separate the node representations $\mathbf{H}$ computed by linear-GNN. However, node representations computed by linear-GNNs might not be linearly separable. To overcome this issue, we propose to first apply a *MLP* on all node features, then apply *linear-GNN* onto the output of the MLP before classification, i.e., $\mathbf{Z} = \sigma(\sigma(\mathbf{X}\mathbf{W}^{(1)}_{\text{mlp}})\mathbf{W}^{(2)}_{\text{mlp}})$, $\mathbf{H} = \mathbf{P}^L\mathbf{Z}\mathbf{W}_{\text{gnn}}$. The above extension can be interpreted as finding a non-linear separation in the input space. During training, we could first pre-train on a public dataset with training samples that do not need to be forgotten, then we only need to unlearn the linear-GNN model by applying PROJECTOR onto the output of the MLP. By doing so, PROJECTOR enjoys both the separation power brought by the non-linearity of MLP and the efficiency brought by the projection-based unlearning.

Adaptive diffusion graph convolution. To help the linear-GNN fully take advantages of the graph structure, we propose an unlearning favorable adaptive diffusion graph convolution operation that take the similarity of both node feature and node label category information into consideration. To achieve this, let us first initialize the node features as $\mathbf{h}_i^{(0)} = \mathbf{x}_i$, initialize node labels as $\mathbf{z}_i^{(0)} = \mathbf{y}_i$ if $i \notin \mathcal{V}_{\text{test}}$ and $\mathbf{z}_i^{(0)} = \mathbf{0}$ if $i \in \mathcal{V}_{\text{test}}$. Then, the forward propagation of the adaptive diffusion graph convolution operation is computed as

$$[\mathbf{H}^{(\ell+1)}, \mathbf{Z}^{(\ell+1)}] = \big((1-\gamma)\mathbf{I} + \gamma\mathbf{D}^{(\ell)}_{\mathcal{G}}\big)[\mathbf{H}^{(\ell)}, \mathbf{Z}^{(\ell)}],$$

where we denote $[\cdot,\cdot]$ as the feature channel concatenation operation and the i-th row j-th column of the ℓ-th diffusion operator is defined by

$$[\mathbf{D}^{(\ell)}(\mathcal{G})]_{i,j} = \frac{1}{Z}\exp(-\sigma_h^2\|\mathbf{h}_i^{(\ell)} - \mathbf{h}_j^{(\ell)}\|_2^2 - \sigma_z^2\|\mathbf{z}_i^{(\ell)} - \mathbf{z}_j^{(\ell)}\|_2^2),$$

where $\sigma_h, \sigma_z \in \mathbb{R}$ are learned during training. Intuitively, our diffusion operator assign a higher neighbor aggregation weight to a node if it has a similar node feature and label information. Then, we set $\mathbf{H} = [\mathbf{H}^{(1)}, \mathbf{Z}^{(1)}, \ldots, \mathbf{H}^{(L)}, \mathbf{Z}^{(L)}]$ as the final node representation for prediction. During unlearning, we do not have to modify σ_h, σ_z since these scalars will not leak the node feature information.

To this end, we conclude this section by showing in Proposition 3 that under mild conditions on $\mathbf{X}$ and $\mathbf{P}$, the linear-GNN used in PROJECTOR could approximate any function defined on the graph. Since non-linearity extension and adaptive diffusion graph convolution could potentially alleviate the conditions on $\mathbf{X}$ and $\mathbf{P}$, these extensions could improve the expressive power of linear-GNN.

Proposition 3 *Let us define $\mathbf{U}, \boldsymbol{\lambda}$ as the eigenvectors and eigenvalues of graph propagation matrix $\mathbf{P}$, $g_{\mathbf{w}}(\mathbf{L}, \mathbf{X}) = \sum_{\ell=1}^{n}(\mathbf{P}^{\ell-1}\mathbf{X})\mathbf{w}_\ell$ as the linear-GNN, and $f(\mathbf{P}, \mathbf{X}) \in \mathbb{R}^{n\times 1}$ as the target function we want to approximate by linear-GNN. If no elements in $\boldsymbol{\lambda}$ are identical and no rows of $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{X}$ are zero vectors, then there is always exists a set of $\mathbf{w}_\ell^\star \in \mathbb{R}^d$ such that $g_{\mathbf{w}^\star}(\mathbf{P}, \mathbf{X}) = f(\mathbf{P}, \mathbf{X})$. Replacing $\mathbf{P}$ with adaptive diffusion graph convolution and replace $\mathbf{X}$ as the output of MLP model could potentially alleviate our requirement on the $\boldsymbol{\lambda}$ and $\tilde{\mathbf{X}}$ since their values are learned by training, therefore improving its expressiveness.*

The intuition behind above proposition is that the expressive power of the linear-GNN $g_{\mathbf{w}}(\mathbf{L}, \mathbf{X})$ mainly comes from its graph convolution. Given a dataset with n nodes, using graph convolutions with polynomial from 0 to $n-1$ allows us map each node feature to its desired value with n different weight parameters, therefore it could approximate any function defined on graph. Proof deferred to Appendix I.

4 Experiments

We consider GRAPHERASER as our exact graph unlearning baseline. For approximate graph unlearning baselines, we extend INFLUENCE and FISHER to graph structured data by taking the node dependency into consideration and rename them as INFLUENCE+ and FISHER+. The details on the baselines are introduced in Appendix B.1. Moreover, since each experiment is designed to evaluate different aspect of unlearning, the setup of each experiment could be slightly different (e.g., linear or non-linear, different deleted node size, different datasets, etc). Therefore, we choose to provide a brief introduction on the experiment design and setup at the beginning of each experiment paragraph, but defer the detailed descriptions to Appendix B.2.

4.1 Experiment results

Feature injection test. This experiment is designed to verify whether PROJECTOR and baselines could perfectly unlearn the trace of deleted node features from the weight parameters. To achieve this goal, we append an extra binary feature to all nodes and set the extra binary feature as 1 for the deleted nodes and as 0 for other nodes. To make sure this extra binary feature is an important feature and is heavily used during training, we add an extra category and change all deleted nodes to this extra category, then pre-train on the modified dataset. We measure the effectiveness of unlearning by checking ① whether unlearning method can *fully unlearn* by comparing weight norm of the injected channel before and after unlearning[2]; ② whether unlearning

[2]Since the weight parameters of logistic regression are weighted combination of all input features used during training, the weight norm of the injected channel before unlearning is expected to be positive if $\mathcal{V}_{\text{delete}}$ are used before unlearning. However, if an unlearning method could perfectly remove the trace of $\mathcal{V}_{\text{delete}}$, the weight norm of the injected channel after unlearning should be

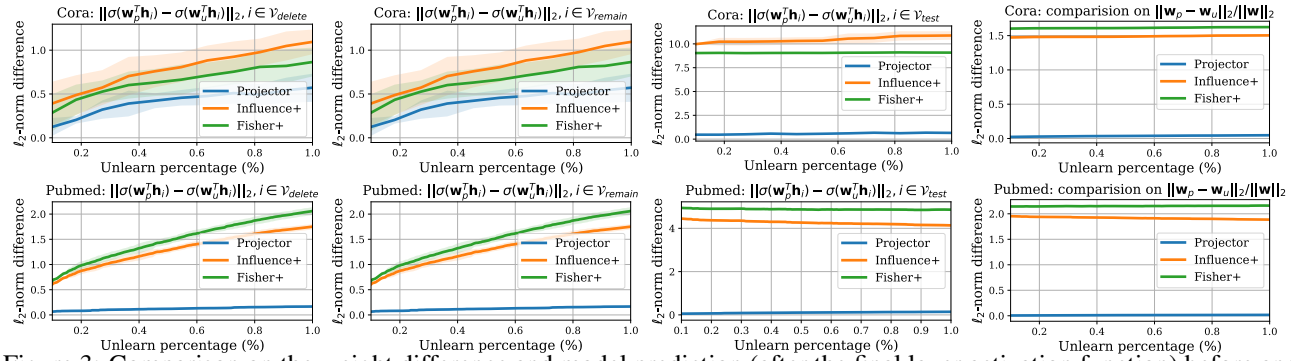


Figure 3: Comparison on the weight difference and model prediction (after the final layer activation function) before and after the unlearning process.

method *hurt the model performance* by comparing the accuracy before and after unlearning; ③ the *computation cost* by comparing the time required for unlearning. We randomly select $5\%, 10\%$ of the nodes from the training set as deleted nodes. We have the following observations from Table 1: ① By comparing the weight norm of the injected channel, we observe that GRAPHERASER and PROJECTOR can perfectly unlearn the deleted nodes and setting the extra-feature channel as zero. However, INFLUENCE+, FISHER+ cannot fully unlearn the correlation because they are approximate unlearning methods; ② By comparing the wall-clock time, PROJECTOR requires less time to unlearn because it is one-shot unlearning approach with the least computation cost, whereas baselines either require re-training for multiple iterations (e.g., GRAPHERASER) or require a larger computation cost to compute Hessian inversion (e.g., INFLUENCE and FISHER); ③ By comparing the accuracy before and after unlearning, INFLUENCE+ and FISHER+ have around $2\%/7\%$ performance degradation on OGB-Arxiv/Products dataset than re-training because a stronger regularization is required to stabilize the unlearning process (to make sure the Hessian inverse is bounded), and GRAPHERASER have around $4\%/9\%$ performance degradation on OGB-Arxiv/Products dataset due to graph partitioning; ④ By comparing the performance of PROJECTOR with and without adaptive diffusion, we know that adaptive diffusion provides consistent performance boosting to linear-GNN models; ⑤ When comparing with re-training from scratch, PROJECTOR is around $0.04 \sim 0.2\%$ slightly better than re-training because PROJECTOR could be thought of as a re-weighting on the remaining nodes, which allows our model to behave similar to the model before unlearning, but without carrying information about the deleted nodes.

Closeness to retraining from scratch. We compare the closeness of the unlearned solution $\mathbf{w}_p$ to the retrained model $\mathbf{w}_u$ to verify our conclusion in Theorem 1 and Proposition 2. We measure the difference between normalized weight parameters $\|\mathbf{w}_u - \mathbf{w}_p\|_2 / \|\mathbf{w}\|_2$ and distance between the final activations $\mathbb{E}_{v_i \in \mathcal{B}}[\|\sigma(\mathbf{w}_p^\top \mathbf{h}_i) - \sigma(\mathbf{w}_u^\top \mathbf{h}_i)\|_2]$ where

zero because the features of $\mathcal{V}_{\text{delete}}$ does not belong to the support vectors of weight parameters.

Table 1: Comparison on the *F1-score accuracy* (Acc), and the norm of extra-feature weight channel (WN) before unlearning and after unlearning (denoted as *before* → *after*), and wall-clock time (T) using linear GNN.

	Method	Metrics	Delete 5% nodes	Delete 10% nodes
OGB-Arxiv	PROJECTOR	Acc (%)	73.33 → 73.39	73.25 → 73.39
		WN (T)	21.7 → 0 (0.07 s)	56.8 → 0 (0.07 s)
	PROJECTOR (+ adapt diff)	Acc (%)	73.42 → 73.48	73.34 → 73.44
		WN (T)	24.3 → 0 (0.07 s)	25.6 → 0 (0.07 s)
	GRAPHERASER (×8 subgraphs)	Acc (%)	70.59 → 70.56	70.55 → 70.23
		WN (T)	22.3 → 0 (1,866 s)	30.6 → 0 (1,866 s)
	INFLUENCE+	Acc (%)	71.90 → 72.73	70.40 → 72.65
		WN (T)	29.2 → 14.1 (1.1 s)	21.1 → 12.1 (1.1 s)
	FISHER+	Acc (%)	72.29 → 72.73	71.71 → 72.65
		WN (T)	29.2 → 14.1 (0.4 s)	35.4 → 15.6 (0.3 s)
	RE-TRAINING (+ adapt diff)	Acc (%)	73.42 → 73.42	73.34 → 73.40
		WN (T)	24.3 → 0 (1,973 s)	25.6 → 0 (1,973 s)
OGB-Products	PROJECTOR	Acc (%)	79.21 → 79.22	79.18 → 79.11
		WN (T)	27.6 → 0 (0.06 s)	30.8 → 0 (0.06 s)
	PROJECTOR (+ adapt diff)	Acc (%)	79.95 → 79.93	79.96 → 79.91
		WN (T)	16.4 → 0 (0.06 s)	18.6 → 0 (0.06 s)
	GRAPHERASER (×8 subgraphs)	Acc (%)	70.80 → 70.78	70.80 → 70.78
		WN (T)	25.4 → 0 (598 s)	28.9 → 0 (598 s)
	INFLUENCE+	Acc (%)	72.23 → 72.78	72.08 → 72.51
		WN (T)	8.9 → 3.1 (1.7 s)	14.3 → 4.2 (1.9 s)
	FISHER+	Acc (%)	72.23 → 72.78	72.08 → 72.51
		WN (T)	8.9 → 3.1 (1.3 s)	14.3 → 4.2 (1.1 s)
	RE-TRAINING (+ adapt diff)	Acc (%)	79.95 → 79.74	79.96 → 79.71
		WN (T)	16.4 → 0 (661 s)	18.6 → 0 (661 s)

$\mathcal{B} \in \{\mathcal{V}_{\text{delete}}, \mathcal{V}_{\text{remain}}, \mathcal{V}_{\text{test}}\}$. Ideally, a powerful unlearning algorithm is expected to generate similar final weight parameters and activations to the retrained model. We randomly select 1% of the nodes from the training set as the deleted nodes $\mathcal{V}_{\text{delete}} \subset \mathcal{V}_{\text{train}}$ and the rest as remain nodes $\mathcal{V}_{\text{remain}} = \mathcal{V}_{\text{train}} \setminus \mathcal{V}_{\text{delete}}$. As shown in Figure 3, both the final activation (column 1, 2, 3) and the output parameters (column 4) of PROJECTOR (blue curve) is closer to the weight obtained by retraining from scratch compared to baseline methods, which could reflect our result in Proposition 2. Besides, we can observe that lower unlearning percentage leads to a smaller difference on the output weight parameters of PROJECTOR (blue curve in column 4), which could reflect our theoretical result in Theorem 1.

Compare to non-linear models. We compare the perfor-

Table 2: Comparison on the performance of linear GNN and its non-linear extension with ordinary GNNs.

		Method	Accuracy
OGB-Arxiv	①	Linear GNN + Adap diff	73.35 ± 0.12
		Linear GNN + Adap diff + MLP	73.41 ± 0.31
	②	GCN	71.74 ± 0.29
		GraphSAGE	71.49 ± 0.27
	③	GCN + GRAPHERASER	66.52 ± 0.31
		GraphSAGE + GRAPHERASER	62.96 ± 0.26
OGB-Product	①	Linear GNN + Adap diff	80.25 ± 0.09
		Linear GNN + Adap diff + MLP	80.30 ± 0.40
	②	GAT	79.45 ± 0.59
		GraphSAGE	78.70 ± 0.36
		GraphSaint	79.08 ± 0.24
	③	GAT + GRAPHERASER	60.23 ± 0.71
		GraphSAGE + GRAPHERASER	58.99 ± 0.40
		GraphSaint + GRAPHERASER	59.54 ± 0.41

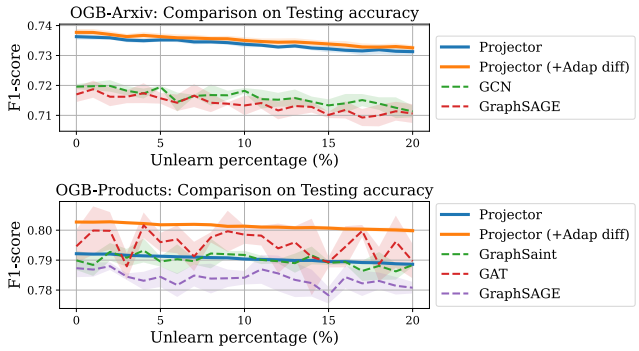


Figure 4: Comparison on the test performance with different number of node to unlearn.

mance of non-linear GNNs (introduced in Section 3.4) and linear GNNs, where the MLP extractor in non-linear PROJECTOR is pre-trained by supervised learning on the features of all training set nodes but except the deleted ones. We have the following observations from Table 2: ① By comparing results in block 1, we know that using MLP as a feature extractor can improve the average F1-score accuracy, but it also increases the variance of the model performance; ② By comparing the results in block 1 and 2, we know that linear-GNN could achieve better performance than ordinary GNNs; ③ By comparing results in block 2 and 3, we know that employing GRAPHERASER with non-linear GNNs will significantly hurt the performance of the original GNN models, which is due to the data heterogeneously and the lack of training data for each subgraph model.

Robustness of PROJECTOR. We study the change of testing accuracy as we progressively increase the unlearning ratio from 1% to 20%, where a more stable model performance is preferred in real-world scenarios. As shown in Figure 4, the change of testing accuracy in PROJECTOR is smaller (e.g., on the OGB-Arvix dataset the test accuracy of PROJECTOR changes around 0.5% while the GNNs change around $0.8\% \sim 1\%$), more stable (i.e., the test accuracy fluctuate less when the fraction of unlearning nodes increases), and with accuracy even better than re-training ordinary GNNs.

Evaluation on the δ term in Assumption 3.

The performance of PROJECTOR's unlearned solution is highly dependent on the correlation between node features, which is captured by the δ term in Assumption 3. Therefore, we report the δ by computing

$$\delta = \max_{v_i \in \mathcal{V}} \left\| \mathbf{x}_i - \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}} (\mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger \mathbf{x}_i \right\|_2 , \tag{5}$$

where $\mathbf{X}_{\text{remain}} = [\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n]$ is the stack of all remaining node features. As shown in Table 3, the δ value is relatively small compared to the norm of average node features, which indicates the realism of our assumption and guarantees the performance of PROJECTOR's unlearned solution (even without finetuning). Besides, we can observe that the δ value on the Cora dataset is larger than other datasets, this is because the feature of the Cora dataset is a binary-valued vector of size 1433 which is very close to the total number of nodes in the graph (2708 nodes). When the node feature dimension is large and all values are either 0 or 1, representing any vectors with others becomes difficult, therefore resulting in a larger δ.

Table 3: Evaluation δ on real-world datasets.

	OGB-Arvix	OGB-Product	Cora	Pubmed
δ	0.3815	0.0915	0.2984	0.0049
$\|\frac{1}{n}\sum_{i=1}^n \mathbf{x}_i\|_2$	9.6369	161.4997	0.6923	0.5546

More experiment results. More experiment results are deferred to the appendix. We compare PROJECTOR with re-training non-linear GNNs under different node deletion schemes in Appendix A.1. We evaluate unlearning with membership inference attack in Appendix A.2. We ablation study the effectiveness of fine-tuning on PROJECTOR in Appendix A.3.

5 Conclusion

In this paper, we study graph representation unlearning by proposing a projection-based unlearning approach PROJECTOR. PROJECTOR unlearns the deleted node features by projecting the weight parameters of a pre-trained model onto a subspace that is irrelevant to the deleted node features. Empirical results on real-world dataset illustrate its effectiveness, efficiency, and robustness.

Acknowledgements

This work was supported in part by NSF grant 2008398.

References

Aldaghri, N., Mahdavifar, H., and Beirami, A. (2021). Coded machine unlearning. *IEEE Access*.

Berg, R. v. d., Kipf, T. N., and Welling, M. (2017). Graph convolutional matrix completion. In *International Conference on Knowledge Discovery & Data Mining*.

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

Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Brophy, J. and Lowd, D. (2021). Machine unlearning for random forests. In *International Conference on Machine Learning*.

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

Chen, C., Sun, F., Zhang, M., and Ding, B. (2022). Recommendation unlearning. *arXiv preprint arXiv:2201.06820*.

Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. (2021). Graph unlearning. *arXiv preprint arXiv:2103.14991*.

Chien, E., Pan, C., and Milenkovic, O. (2022). Certified graph unlearning. *arXiv preprint arXiv:2206.09140*.

Chu, B.-Y., Ho, C.-H., Tsai, C.-H., Lin, C.-Y., and Lin, C.-J. (2015). Warm start for parameter selection of linear classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 149–158.

Cong, W., Ramezani, M., and Mahdavi, M. (2021). On provable benefits of depth in training graph convolutional networks. *Advances in Neural Information Processing Systems*.

Diehl, C. P. and Cauwenberghs, G. (2003). Svm incremental learning, adaptation and optimization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 4, pages 2685–2690. IEEE.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fu, S., He, F., and Tao, D. (2022). Knowledge removal in sampling-based bayesian inference. In *International Conference on Learning Representations*.

Gâlmeanu, H. and Andonie, R. (2008). Implementation issues of an incremental and decremental svm. In *International Conference on Artificial Neural Networks*, pages 325–335. Springer.

Ginart, A., Guan, M. Y., Valiant, G., and Zou, J. (2019). Making ai forget you: Data deletion in machine learning. *arXiv:1907.05012*.

Golatkar, A., Achille, A., Ravichandran, A., Polito, M., and Soatto, S. (2021). Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Golatkar, A., Achille, A., and Soatto, S. (2020). Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Golub, G. H. and Van Loan, C. F. (2013). *Matrix computations*. JHU press.

Guo, C., Goldstein, T., Hannun, A., and Van Der Maaten, L. (2020). Certified data removal from machine learning models. In *International Conference on Machine Learning*.

Hamilton, W. L., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*.

He, Y., Meng, G., Chen, K., He, J., and Hu, X. (2021). Deepobliviate: A powerful charm for erasing data residual memory in deep neural networks. *arXiv preprint arXiv:2105.06209*.

Izzo, Z., Smart, M. A., Chaudhuri, K., and Zou, J. (2021). Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*.

Karasuyama, M. and Takeuchi, I. (2009). Multiple incremental decremental learning of support vector machines. *Advances in neural information processing systems*, 22.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Klicpera, J., Bojchevski, A., and Günnemann, S. (2018). Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*.

Laskov, P., Gehl, C., Krüger, S., Müller, K.-R., Bennett, K. P., and Parrado-Hernández, E. (2006). Incremental support vector learning: Analysis, implementation and applications. *Journal of machine learning research*, 7(9).

Neel, S., Roth, A., and Sharifi-Malvajerdi, S. (2020). Descent-to-delete: Gradient-based methods for machine unlearning. *arXiv preprint arXiv:2007.02923*.

Nguyen, Q. P., Oikawa, R., Divakaran, D. M., Chan, M. C., and Low, B. K. H. (2022). Markov chain monte carlo-based machine unlearning: Unlearning what needs to be forgotten. *arXiv preprint arXiv:2202.13585*.

Olatunji, I. E., Nejdl, W., and Khosla, M. (2021). Membership inference attack on graph neural networks. In *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 11–20. IEEE.

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines.

Ramezani, M., Cong, W., Mahdavi, M., Kandemir, M. T., and Sivasubramaniam, A. (2022). Learn locally, correct globally: A distributed algorithm for training graph neural networks. *ICLR*.

Sekhari, A., Acharya, J., Kamath, G., and Suresh, A. T. (2021). Remember what you want to forget: Algorithms for machine unlearning.

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30.

Thudi, A., Jia, H., Shumailov, I., and Papernot, N. (2021). On the necessity of auditable algorithmic definitions for machine unlearning. *arXiv preprint arXiv:2110.11891*.

Tsai, C.-H., Lin, C.-Y., and Lin, C.-J. (2014). Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 343–352.

Ullah, E., Mai, T., Rao, A., Rossi, R., and Arora, R. (2021). Machine unlearning via algorithmic stability.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, H., Zhang, F., Zhang, M., Leskovec, J., Zhao, M., Li, W., and Wang, Z. (2019a). Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *International Conference on Knowledge Discovery & Data Mining*.

Wang, X., He, X., Cao, Y., Liu, M., and Chua, T. (2019b). KGAT: knowledge graph attention network for recommendation. In *International Conference on Knowledge Discovery & Data Mining*.

Wang, X. and Zhang, M. (2022). How powerful are spectral graph neural networks. *arXiv preprint arXiv:2205.11172*.

Wang, Y., Jin, J., Zhang, W., Yu, Y., Zhang, Z., and Wipf, D. (2021). Bag of tricks for node classification with graph neural networks. *arXiv preprint arXiv:2103.13355*.

Wu, G., Hashemi, M., and Srinivasa, C. (2022). Puma: Performance unchanged model augmentation for training data removal. *arXiv preprint arXiv:2203.00846*.

Wu, Y., Dobriban, E., and Davidson, S. (2020a). Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*.

Wu, Y., Tannen, V., and Davidson, S. B. (2020b). Priu: A provenance-based approach for incrementally updating regression models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Kannan, R., Prasanna, V., Jin, L., Malevich, A., and Chen, R. (2020). Deep graph neural networks with shallow subgraph samplers.

Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Malevich, A., Kannan, R., Prasanna, V., Jin, L., and Chen, R. (2021). Decoupling the depth and scope of graph neural networks. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Table of contents

A More experiment results

In this section, we provide more empirical evaluation results.

- Appendix A.1: We compare PROJECTOR with re-training non-linear GNNs under different node deletion schemes.
- Appendix A.2: We conduct experiments using membership inference attack framework.
- Appendix A.3: We ablation study the effectiveness of fine-tuning on PROJECTOR's unlearned model.

Code to reproduce the experiment results can be found at [Repository].

A.1 Linear vs non-linear GNN under different deleted nodes selection schemes

In this section, we compare the model performance of PROJECTOR against re-training 2-layer GNNs from scratch under different unlearning settings. We consider GCN Kipf and Welling (2017), GraphSAGE Hamilton et al. (2017), APPNP Klicpera et al. (2018), and GAT Veličković et al. (2017) as the baseline 2-layer GNNs. Notice that using a 2-layer is a common choice in graph representation learning to balance between computation cost, training accuracy, and generalization. To scale for large-graph training, we utilize the K-hop shadow sampler Zeng et al. (2021) implemented in Pytorch Geometric Fey and Lenssen (2019). We consider two different unlearning settings, i.e., unlearning 10% training set nodes with the largest node degree (i.e., "delete dense nodes" in Table 4) and unlearning 10% training set nodes with the smallest node degree (i.e., "delete sparse nodes" in Table 4), to simulate the potential real-world node deletion scenario.

Table 4: Compare the model performance PROJECTOR with re-training 2-layer GNNs from scratch under different deleted nodes selection schemes.

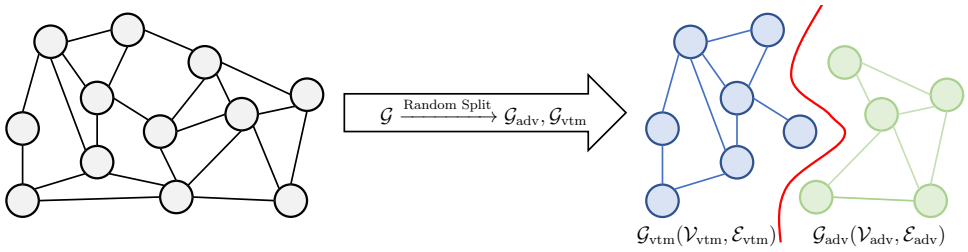
			Before node deletion	Delete dense nodes	Delete sparse nodes
Flickr (avg node degree 10)	GCN	Re-train	50.01 ± 0.13 (241.08s)	49.35 ± 0.10 (194.75s)	49.79 ± 0.18 (234.93s)
	GraphSAGE	Re-train	51.34 ± 0.14 (243.00s)	50.23 ± 0.22 (195.49s)	50.93 ± 0.20 (236.77s)
	APPNP	Re-train	50.04 ± 0.09 (244.53s)	49.03 ± 0.17 (200.34s)	49.60 ± 0.06 (239.71s)
	GAT	Re-train	51.01 ± 0.11 (409.51s)	49.78 ± 0.23 (313.75s)	50.76 ± 0.22 (392.77s)
	Linear-GNN	Re-train	$\mathbf{52.71} \pm 0.14$ (374.50s)	49.21 ± 0.23 (268.09s)	51.84 ± 0.13 (316.76s)
		Projector		$\mathbf{50.63} \pm 0.19$ (0.06s)	$\mathbf{51.90} \pm 0.11$ (0.05s)
Reddit (avg node degree 50)	GCN	Re-train	93.04 ± 0.09 (1174.12s)	92.95 ± 0.06 (910.51s)	91.47 ± 0.09 (1129.89s)
	GraphSAGE	Re-train	94.68 ± 0.06 (1005.13s)	94.90 ± 0.07 (775.80s)	94.45 ± 0.06 (917.80s)
	APPNP	Re-train	93.73 ± 0.06 (1010.02s)	94.08 ± 0.10 (780.10s)	93.09 ± 0.08 (925.56s)
	GAT	Re-train	92.82 ± 0.10 (1431.56s)	93.12 ± 0.05 (1228.95s)	92.41 ± 0.10 (1297.23s)
	Linear-GNN	Re-train	$\mathbf{94.72} \pm 0.08$ (1630.53s)	95.03 ± 0.03 (1290.16s)	94.58 ± 0.02 (1532.51s)
		Projector		$\mathbf{95.09} \pm 0.03$ (0.24s)	$\mathbf{94.66} \pm 0.07$ (0.32s)

We have the following observations From Table 4:

- Linear-GNNs could achieve even better performance than non-linear GNNs. For example, linear-GNN is around $0.1 \sim 1\%$ better than 2-layer GNNs on Flickr and Reddit dataset. This also verifies the arguments in Proposition 3 that the expressive power of linear-GNNs mainly comes from its weight combination of multi-hop graph convolution operators. Besides, since linear-GNN has lower model complexity, it could generalize better than multi-layer GNNs.
- Interestingly, according to the second column of our results, we found that removing dense nodes on the sparser graph (e.g., Flickr) hurt the model performance to around $1 \sim 2\%$, however, removing dense nodes on the denser graph (e.g., Reddit) it will improves the model performance to around 1%. This is potentially because those dense nodes provide too much redundant information on the denser graph than on a sparser graph or due to the over-smoothing Xu et al. (2018) issue caused by aggregating too many neighbors in the original graph.
- The performance of PROJECTOR is around $0.06 \sim 0.1\%$ better than re-training on the graph without the deleted nodes. This is potential because the output of PROJECTOR could be interpreted as re-training on the remaining graph under some unknown importance sampling distribution, while this importance distribution help PROJECTOR learn better from the remaining data.

A.2 Evaluation by Membership Inference Attack

In this section, we conduct membership inference attack Olatunji et al. (2021) to test whether GNN models could potentially leak information about the deleted nodes' membership information and whether PROJECTOR could alleviate the information leakage issue. In the following, we will first give a brief introduction on the GNN membership inference attack settings used in Olatunji et al. (2021) then provide details on our experiment results.



Step 1. Train adversary: $\mathcal{G}_{\text{adv}} \xrightarrow{\text{Train GNN}_{\text{adv}}} \mathbf{H}_{\text{adv}} = \text{GNN}_{\text{adv}}(\mathcal{G}_{\text{adv}}) \xrightarrow{\text{Train MLP}_{\text{adv}}} \text{MLP}_{\text{adv}}(\mathbf{H}_{\text{adv}})$

Step 2. Train victim: $\mathcal{G}_{\text{vtm}} \xrightarrow{\text{Train GNN}_{\text{vtm}}} \mathbf{H}_{\text{vtm}} = \text{GNN}_{\text{vtm}}(\mathcal{G}_{\text{vtm}})$

Step 3. Apply adversary model on victim model: $\text{MLP}_{\text{adv}}(\mathbf{H}_{\text{vtm}}) \xrightarrow{\text{Is trained?}}$

Figure 5: An overview on the workflow of membership inference attack.

Membership inference attack setting. An overview of the membership inference attack is introduced in Figure 5, which follows the implementation of the graph membership inference attack that is proposed in Olatunji et al. (2021). Let us denote $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as the full graph. In the membership inference attack, a necessary assumption is the graphs used by adversaries and victims are from the same distribution. To achieve this goal, we randomly split all nodes into adversary's node sets $\mathcal{V}_{\text{adv}} \subset \mathcal{V}$ and victim's node sets $\mathcal{V}_{\text{vtm}} = \mathcal{V} \setminus \mathcal{V}_{\text{adv}}$. Then, we define the subgraph induced by the two node sets as adversary subgraph $\mathcal{G}_{\text{adv}}$ and victim subgraph $\mathcal{G}_{\text{vtm}}$. This process is illustrated at the top of Figure 5. On the adversary side, the adversary first pre-trains a GNN model GNN_{adv} on $\mathcal{G}_{\text{adv}}$ by only using a subset of nodes as training sets and extract the node representation as $\mathbf{H}_{\text{adv}}$. Then, a binary classifier MLP_{adv} is trained on $\mathbf{H}_{\text{adv}}$ to classify whether a node has been used for training for not. On the victim side, the victim only need to train a GNN model GNN_{vtm} on $\mathcal{G}_{\text{vtm}}$ by only use a subset of nodes as training sets and extract the node representation as $\mathbf{H}_{\text{vtm}}$. During the membership inference attack, the adversary applies MLP_{adv} onto $\mathbf{H}_{\text{vtm}}$ to distinguish if a node is used for training. Membership inference attack is more challenging on graph data because the adversary classifier MLP_{adv} is applied to the node representation, and the node representation of some nodes might be very similar if they share the same neighborhood information.

Table 5: Comparison on the membership inference attack accuracy before and after unlearning.

	Method / Phase	$\mathcal{V}_{\text{delete}}$ as untrained	$\mathcal{V}_{\text{train}}^{\text{after}}$ as trained	$\mathcal{V} \setminus \mathcal{V}_{\text{train}}^{\text{before}}$ as untrained	$\mathcal{V}_{\text{train}}^{\text{before}}$
Cora	BEFORE-UNLEARN	46.35 ± 16.99	52.31 ± 16.31	63.17 ± 12.12	57.79 ± 2.84
	RE-TRAINING	59.37 ± 11.82	56.30 ± 16.56	62.98 ± 14.01	58.79 ± 1.83
	PROJECTOR	58.38 ± 11.75	51.64 ± 16.23	63.84 ± 12.14	57.76 ± 2.15
Citeseer	BEFORE-UNLEARN	47.33 ± 31.21	38.52 ± 32.73	68.30 ± 29.97	54.97 ± 2.32
	RE-TRAINING	66.33 ± 25.60	54.04 ± 30.86	55.53 ± 28.94	55.20 ± 3.08
	PROJECTOR	67.33 ± 27.50	38.22 ± 32.68	68.97 ± 29.56	55.05 ± 2.17

Results. We compare the accuracy of MLP_{adv} classifies each node representation $\mathbf{H}_{\text{vtm}}$ as training or non-training set nodes before and after unlearning. Let us denote $\mathcal{V}_{\text{train}}^{\text{before}} \subset \mathcal{V}_{\text{victim}}$ as the subset of nodes used for training before node deletion, denote $\mathcal{V}_{\text{train}}^{\text{after}} \subset \mathcal{V}_{\text{train}}^{\text{before}}$ as the subset of nodes used for training after node deletion, and denote $\mathcal{V}_{\text{delete}} = \mathcal{V}_{\text{train}}^{\text{before}} \setminus \mathcal{V}_{\text{train}}^{\text{after}}$ as the nodes for deletion. Two victim models are trained on training set nodes $\mathcal{V}_{\text{train}}^{\text{before}}, \mathcal{V}_{\text{train}}^{\text{after}}$. We report the accuracy on $\mathcal{V}_{\text{train}}^{\text{after}}, \mathcal{V}_{\text{delete}}, \mathcal{V}_{\text{vtm}} \setminus \mathcal{V}_{\text{train}}^{\text{before}}$ before and after unlearning. We are using the re-trained model as a baseline. In terms of the size of node sets, we set $|\mathcal{V}_{\text{train}}^{\text{before}}| = 0.5 \times |\mathcal{V}_{\text{victim}}|$ and $|\mathcal{V}_{\text{delete}}| = 0.9 \times |\mathcal{V}_{\text{train}}^{\text{before}}|$. We have the following observations from Table 5:

- By comparing the BEFORE-UNLEARN with RE-TRAINING and PROJECTOR, we know that both re-training and our proposal could increase the probability that MLP_{adv} classify $\mathcal{V}_{\text{delete}}$ at "untrained". More specifically, when applying MLP_{adv} on the model before-unlearning, since $\mathcal{V}_{\text{delete}}$ are used before unlearning, the probability of classifying $\mathcal{V}_{\text{delete}}$

as "untrained" should be lower than 50%. However, after re-training or using our unlearning approach, the probability of classifying $\mathcal{V}_{\text{delete}}$ as "untrained" increases as the information on $\mathcal{V}_{\text{delete}}$ are removed during the unlearning process.

- By comparing the BEFORE-UNLEARN with PROJECTOR at each column, we can observe that the prediction of MLP_{adv} on $\mathcal{V}_{\text{train}}^{\text{after}}$ and $\mathcal{V} \setminus \mathcal{V}_{\text{train}}^{\text{before}}$ are almost unchanged. This is also expected as our projection step only remove the trace of the deleted nodes and will preserve its model performance/behavior as much as possible according to our method design in Section 3.

A.3 Performance Before and After Finetuning

The experiment results in Section 4 are reported without the fine-tuning process as mentioned in Theorem 1. For the completeness of our discussion, we provide further results on the comparison of the training, validation, and testing accuracy of the unlearned model both with and without the fine-tuning process.

Setup. In this experiment, we randomly select 1% of the nodes from the training set to unlearn. During both training and fine-tunings, we early stop if the validation accuracy does not increase within 10 iterations on the OGB-Arvix dataset and $1,000$ iterations on the OGB-Products dataset. We repeat the experiment 5 times. Other setup remains the same as introduced in Section B.2.

Results. According to our result in Figure 6, we have the following observations: ① By looking at the blue curve, we know that both the training and validation accuracy dropped after unlearning, which is expected as part of the information related to the deleted nodes are removed; ② By looking at the orange curve, we can observe that the fine-tuning training accuracy indeed improves progressively but the improvement is relatively small, this is because the solution after PROJECTOR unlearning is already close to the optimal solution, which could be partially explained by the hypothesis that *small changes on the dataset will not results in massive changes on the optimal solution*. ③ By comparing the orange curve and green curve, we know that fine-tuning on the unlearned solution (orange curve) could save a lot of time comparing to re-training from scratch (green curve). Furthermore, we compare the F1-score on the test set in Table 6 and have the following observations: ① When without the adaptive diffusion operation, the generalization performance on the testing set between fine-tuning to re-training are relatively close; ② However, if using the adaptive diffusion operation, the unlearning solution (no matter with or without the fine-tuning step) always outperform re-training from stretch. This is potentially because more data are used to tune the scatter parameters in the adaptive diffusion, which leads to a better generalization ability; ③ The performance before and after the fine-tuning is relatively close, which indicates the impressive generalizability of our unlearning solution.

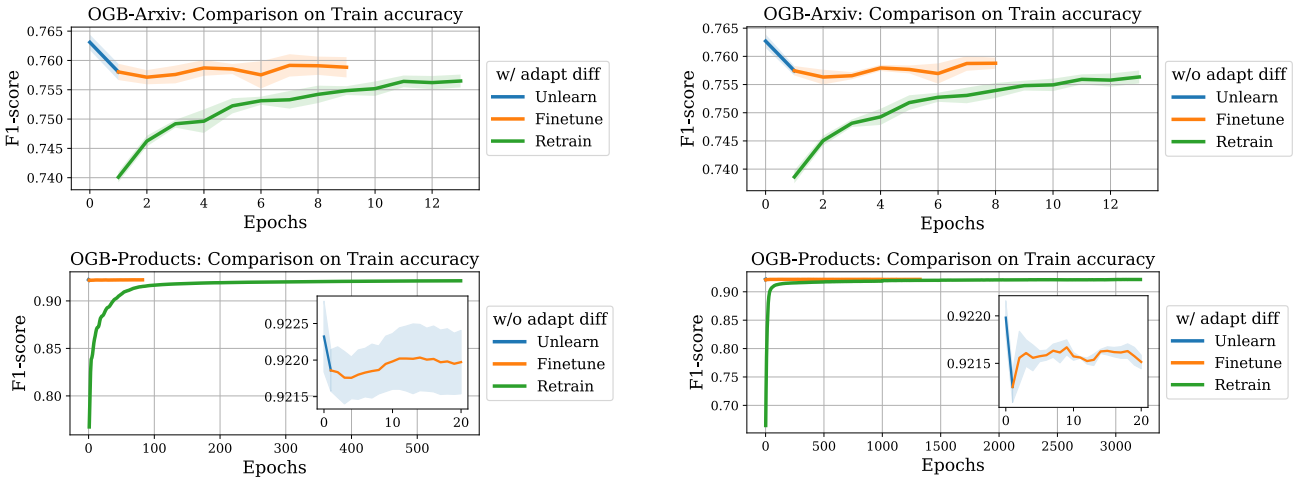


Figure 6: Evaluation on effect of finetuning on the training accuracy before and after 1% of the node from training set deleted.

Table 6: Comparison of F1-score on the testing set before unleanring, after unlearning, after fine-tuning, and re-training with 1% of the node from training set deleted on OGB datasets.

Arxiv	Status	Test F1-score (%)	**Products**	Status	Test F1-score (%)
PROJECTOR	Before unlearning	73.25 ± 00.23	PROJECTOR	Before unlearning	79.11 ± 00.08
	After unlearning	72.97 ± 00.24		After unlearning	78.96 ± 00.06
	After fine-tuning	73.03 ± 00.11		After fine-tuning	79.06 ± 00.06
	Re-training	73.02 ± 00.11		Re-training	78.78 ± 00.14
PROJECTOR (+ adapt diff)	Before unlearning	73.35 ± 00.12	PROJECTOR (+ adapt diff)	Before unlearning	80.25 ± 00.09
	After unlearning	73.09 ± 00.12		After unlearning	79.95 ± 00.12
	After fine-tuning	73.13 ± 00.12		After fine-tuning	80.02 ± 00.37
	Re-training	73.00 ± 00.12		Re-training	79.87 ± 00.47

B Missing details from Section 4 (experiment section)

B.1 Details on baseline methods

In this paper, we consider exact unlearning method GRAPHERASER Chen et al. (2021), approximate unlearning method INFLUENCE+ Guo et al. (2020) and FISHER+ Golatkar et al. (2020) as baseline methods.

Details on GRAPHERASER. GRAPHERASER is an exact unlearning method. GRAPHERASER proposes to split the original graph into multiple shards (i.e., subgraphs) and train an independent model on each data shard. During inference, GRAPHERASER averages the prediction of each shard model as the final prediction. Upon receiving unlearning requests, GRAPHERASER only needs to re-train the specific shard model where the deleted data belongs to. In the experiment, we split all nodes into 8 shards using graph partition algorithm `METIS` and use mean average for model aggregation. Each shard model is trained with enough epochs and we return the epoch model with the highest validation score. Our implementation is based on their official implementation[3] and is general enough to captured the main spirit of GRAPHERASER, i.e., split data into multiple shards and train a shard model on each shard. METIS allows us to split the original graph into multiple subgraphs while preserving the original graph structure as much as possible. We also test their official implementation with different model aggregation and graph partition strategies, but their performance is not as good as the `METIS` partitioning and mean aggregation on the more challenging OGB datasets.

Details on INFLUENCE+. INFLUENCE+ is approximate unlearning method and is implemented based on its official code[4]. INFLUENCE+ proposes to unlearn by removing the influence of the deleted data on the model parameters. Formally, let $\mathcal{D}_d \subset \mathcal{D}$ denote the deleted subset of training data, $\mathcal{D}_r = \mathcal{D} \setminus \mathcal{D}_d$ denote the remaining data, $\mathcal{L}(\mathbf{w})$ is the objective function, and $\mathbf{w}$ is the model parameters before unlearning. Then, INFLUENCE+ unlearn by $\mathbf{w}^u = \mathbf{w} + \mathbf{H}_r^{-1}\mathbf{g}_d$, which is derived from the first-order Taylor approximation on gradient, where $\mathbf{w}^u$ is the parameters after unlearning, $\mathbf{H}_r = \nabla^2\mathcal{L}(\mathbf{w}, \mathcal{D}_r)$ is the Hessian computed on the remaining data, and $\mathbf{g}_d = \nabla\mathcal{L}(\mathbf{w}, \mathcal{D}_d)$ is the gradient computed on the deleted data. To mitigate the potential information leakage, INFLUENCE+ utilizes a perturbed objective function $\mathbf{L}(\mathbf{w}) + \mathbf{b}^\top\mathbf{w}$, where $\mathbf{b}$ is the random noise. INFLUENCE+ requires the loss function as logistic regression, we use the one-vs-rest strategy splits the multi-class classification into one binary classification problem per class and train with logistic regression. Besides, INFLUENCE+ requires the *i.i.d.* data and cannot handle graph structured data, we opt to update both the deleted and affected nodes in parallel. A reader who is interesting the mathematically details could refer to Section C.

Details on FISHER+. FISHER+ is approximate unlearning method and is edited based on their official code[5]. FISHER+ performs Fisher forgetting by taking a single step of Newton's method on the remaining training data, then performing noise injection to model parameters to mitigate the potential information leaking. The model parameters after unlearning is given by $\mathbf{w}^u = \mathbf{w} - \mathbf{H}_r^{-1}\mathbf{g}_r + \mathbf{H}_r^{-1/4}\mathbf{b}$, where $\mathbf{H}_r = \nabla^2\mathcal{L}(\mathbf{w}, \mathcal{D}_r)$ is Hessian and $\mathbf{g}_r = \nabla\mathcal{L}(\mathbf{w}, \mathcal{D}_r)$ is gradient computed on the remaining data $\mathcal{D}_r$, and $\mathbf{b}$ is the random noise.

Details on multi-layer GNNs. For experiments on OGB datasets, we take their code from the Open Graph Benchmark's online public implementation and use the same hyper-parameters as originally provided. For example, the implementations on OGB-Arxiv is based on the code at here and the implementation on OGB-Products is based on code at here. For experiments on other datasets, we take the example code from PyTorch Geometric at here and use the same hyper-parameters

[3] `https://github.com/MinChen00/Graph-Unlearning`
[4] `https://github.com/facebookresearch/certified-removal`
[5] `https://github.com/AdityaGolatkar/SelectiveForgetting`

as originally provided.

B.2 Details on experiment setups

Experiment environment. We conduct experiments on a single machine with Intel i9 CPU, Nvidia RTX 3090 GPU, and 64GB RAM memory. The code is written in Python 3.7 and we use PyTorch 1.4 on CUDA 10.1 for model training. We repeat the experiment 5 times and report the average results (for all experiments) and its standard deviation (for all experiment results except the Table 1 due to space limit).

Model configuration. For fair comparision, the same linear-GNN is used for PROJECTOR and baseline methods is used: we use 3-layer linear-GNN with shallow-subgraph sampler Zeng et al. (2020) for OGB-Arxiv and OGB-Products dataset, use 2-layer linear-GNN with full-batch training for Cora and Pubmed dataset. During training, label reuse tricks in Wang et al. (2021) are used that leverage the training set node label information for inference. In terms of the linear GNN model we used in PROJECTORand all other baselines, we train the linear-GNN using SGD with momentum with learning rate selected from $\{0.1, 1.0\}$, momentum as 0.9, adaptive aggregation step size $\gamma = 1$, and regularization as 10^{-6}. Besides, we choose the regularization term λ in INFLUENCE+ and FISHER+ to balance the performance before and after unlearning: when λ is small, we are facing the gradient exploding issue where the gradient norm is an order of magnitude larger than the weight norm, such that the unlearned model cannot generate meaningful predictions. However, a larger λ will hurt model's learning ability and results in a poor performance before unlearning.

Details on dataset. We summarize the datasets that are used for experiments in Table 7.

Table 7: Statistics of the datasets used in our experiments.

	# Nodes	# Edges	# Features	# Classes
OGB-Arxiv	169,343	1,166,243	128	40
OGB-Products	2,449,029	61,859,140	100	47
Cora	2,708	10,556	1,433	7
Pubmed	19,717	88,648	500	3
Flickr	89,250	899,756	500	7
Reddit	232,965	114,615,892	602	41

C Dependency issue in applying existing unlearning approaches

Most unlearning approaches Wu et al. (2020a); Guo et al. (2020); Izzo et al. (2021) are designed for the settings where the loss function can be decomposed over individual training samples. Directly generalizing the aforementioned general machine unlearning methods to graph structured data is infeasible due to the node dependency. In other word, one cannot directly unlearn a specific node v_i, but have to remove the effect of all its multi-hop neighbors in parallel if using these methods.

In the following, we use Guo et al. (2020) as an example to illustrate the key issue. The discussion also applied to other machine unlearning methods that require the loss function to be decomposed over individual training samples. In the following, we first recall how the influence function is used to update the weight parameters in Guo et al. (2020), then highlight why node dependency makes applying Guo et al. (2020) to graph-structured data challenging and introduce a solution to alleviate this issue.

Influence function in Guo et al. (2020). The influence function used in Guo et al. (2020) captures the change in model parameters due to removing a data point from the training set. Let $L(\mathbf{w})$ denote the finite-sum objective function computed on the full training set $\{\mathbf{x}_i\}_{i=1}^n$ with optimal solution

$$\mathbf{w}_\star = \arg\min_{\mathbf{w}} L(\mathbf{w}), \text{where } L(\mathbf{w}) = \sum_{i=1}^{n} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) \tag{6}$$

and $L_{\setminus n}(\mathbf{w})$ denote the objective function without data point $(\mathbf{x}_n, y_n)$ with optimal solution

$$\mathbf{w}_{\setminus n} = \arg\min_{\mathbf{w}} L_{\setminus_n}(\mathbf{w}), \text{where } L_{\setminus_n}(\mathbf{w}) = \sum_{i=1}^{n-1} \ell(\mathbf{w}^\top \mathbf{x}_i, y_i) = L(\mathbf{w}) - \ell(\mathbf{w}^\top \mathbf{x}_n, y_n). \tag{7}$$

From $\mathbf{w}_{\setminus n} = \arg\min_{\mathbf{w}} L_{\setminus_n}(\mathbf{w})$ and the convexity of the objective function $L_{\setminus_n}$, we know that $\nabla L_{\setminus n}(\mathbf{w}_{\setminus n}) = \mathbf{0}$. Therefore, we have

$$\begin{aligned} 0 &= \nabla L(\mathbf{w}_{\setminus n}) - \nabla \ell(\mathbf{w}_{\setminus n}^\top \mathbf{x}_n, y_n) \\ &\underset{(a)}{\approx} \left[\nabla L(\mathbf{w}_\star) + \nabla^2 L(\mathbf{w}_\star)(\mathbf{w}_{\setminus n} - \mathbf{w}_\star)\right] - \left[\nabla \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n) + \nabla^2 \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n)(\mathbf{w}_{\setminus n} - \mathbf{w}_\star)\right] \\ &= \left[\nabla L(\mathbf{w}_\star) - \nabla \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n)\right] + \left[\nabla^2 L(\mathbf{w}_\star) - \nabla^2 \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n)\right](\mathbf{w}_{\setminus n} - \mathbf{w}_\star) \\ &\underset{(b)}{=} \left[-\nabla \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n)\right] + \left[\nabla^2 L(\mathbf{w}_\star) - \nabla^2 \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n)\right](\mathbf{w}_{\setminus n} - \mathbf{w}_\star), \end{aligned} \tag{8}$$

where (a) is the first-order Taylor expansion and (b) due to $\nabla L(\mathbf{w}_\star) = \mathbf{0}$ for $\mathbf{w}_\star = \arg\min_{\mathbf{w}} L(\mathbf{w})$. Re-arranging the above equation we have

$$\mathbf{w}_{\setminus n} \approx \mathbf{w}_\star + \underbrace{\left[\nabla^2 L(\mathbf{w}_\star) - \nabla^2 \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n)\right] \nabla \ell(\mathbf{w}_\star^\top \mathbf{x}_n, y_n)}_{\text{influence function}}, \tag{9}$$

where the second term on the right hand side is the so called influence function.

Challenges due to dependency in graph. Please notice that the objective function in Eq. 6 and Eq. 7 are finite-sum formulation. In the following, we will show that directly using the second-order method in Guo et al. (2020) is not allowed due to the node dependency in graph. Before getting started, let me first introduce some notations:

- Let us denote the graph before node deletion as $\mathcal{G}$, where the graph structure is captured by adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$ and node feature matrix is $\mathbf{X}$. The row normalized propagation matrix us computed as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$.

- Let us denote the graph after node deletion as $\mathcal{G}_{\setminus n}$, where the graph structure is captured by adjacency matrix $\mathbf{A}_{\setminus n} \in \{0,1\}^{(n-1)\times(n-1)}$ and node feature matrix is $\mathbf{X}_{\setminus n} \in \mathbb{R}^{(n-1)\times d}$. The row normalized propagation matrix us computed as $\mathbf{P}_{\setminus n} = \mathbf{D}_{\setminus n}^{-1}\mathbf{A}_{\setminus n}$.

For simplicity, let us only consider 1-hop SGC, which is already enough to illustrate why node dependency makes applying machine unlearning methods to graph structured data challenging. In graph structured data, let $F(\mathbf{w})$ denote the objective function computed on the full training graph $\mathcal{G}$ with optimal solution

$$\mathbf{w}_\star = \arg\min_{\mathbf{w}} L(\mathbf{w}), \text{where } L(\mathbf{w}) = \sum_{i=1}^{n} \ell(\mathbf{w}^\top [\mathbf{P}\mathbf{X}]_i, y_i) \tag{10}$$

and $L_{\setminus n}(\mathbf{w})$ denote the objective function computed on graph $\mathcal{G}_{\setminus n}$ without node n , with optimal solution

$$\begin{aligned} \mathbf{w}_{\setminus n} = \arg\min_{\mathbf{w}} L_{\setminus_n}(\mathbf{w}), \text{where } L_{\setminus_n}(\mathbf{w}) &= \sum_{i=1}^{n-1} \ell(\mathbf{w}^\top [\mathbf{P}_{\setminus n}\mathbf{X}]_i, y_i) \\ &\underset{(a)}{\neq} L(\mathbf{w}) - \ell(\mathbf{w}^\top [\mathbf{P}\mathbf{X}]_n, y_n). \end{aligned} \tag{11}$$

Due to the inequality of (a), we cannot directly use the second-order method in Guo et al. (2020) to approximate $\mathbf{w}_{\setminus n}$ from $\mathbf{w}_\star$. Please notice that this equality is important in Eq. 8 before using first-order Taylor expansion.

Get around this issue by deleting more nodes. One way to alleviate this issue is to update all the affected nodes $\mathcal{V}_{\text{affect}} = \{v_n\} \cup \mathcal{N}(v_n)$ in parallel. To see this, according to the definition of $\mathcal{V}_{\text{affect}}$, we know $[\mathbf{P}\mathbf{X}]_i = [\mathbf{P}_{\setminus n}\mathbf{X}_{\setminus n}]_i,\ \forall v_i \in \mathcal{V} \setminus \mathcal{V}_{\text{affect}}$ because all the final-layer output of any node in $\mathcal{V} \setminus \mathcal{V}_{\text{affect}}$ are remaining the same after node deletion. Then, we can define

the new objective function $L_{\setminus \mathcal{V}_{\text{affect}}}(\mathbf{w})$ on node set $\mathcal{V} \setminus \mathcal{V}_{\text{affect}}$

$$\begin{aligned} L_{\setminus \mathcal{V}_{\text{affect}}}(\mathbf{w}) &= \sum_{i \in \mathcal{V} \setminus \mathcal{V}_{\text{affect}}} \ell(\mathbf{w}^\top [\mathbf{P}_{\setminus n} \mathbf{X}_{\setminus n}]_i, y_i) \\ &= \sum_{i \in \mathcal{V} \setminus \mathcal{V}_{\text{affect}}} \ell(\mathbf{w}^\top [\mathbf{P}\mathbf{X}]_i, y_i) \\ &\underset{(a)}{=} L(\mathbf{w}) - \sum_{i \in \mathcal{V}_{\text{affect}}} \ell(\mathbf{w}^\top [\mathbf{P}\mathbf{X}]_i, y_i) + \sum_{i \in \mathcal{V}_{\text{affect}} \setminus \{n\}} \ell(\mathbf{w}^\top [\mathbf{P}_{\setminus n} \mathbf{X}_{\setminus n}]_i, y_i), \end{aligned} \tag{12}$$

where the equality in (a) is what we are looking for and is similar to the last term in Eq. 7. To this end, let us define $\mathbf{w}_{\setminus \mathcal{V}_{\text{affect}}} = \arg\min_{\mathbf{w}} L_{\setminus \mathcal{V}_{\text{affect}}}(\mathbf{w})$, then we have

$$\begin{aligned} 0 &= \nabla L(\mathbf{w}_{\setminus \mathcal{V}_{\text{affect}}}) - \sum_{i \in \mathcal{V}_{\text{affect}}} \nabla \ell(\mathbf{w}_{\setminus \mathcal{V}_{\text{affect}}}^\top [\mathbf{P}\mathbf{X}]_i, y_i) + \sum_{i \in \mathcal{V}_{\text{affect}} \setminus \{n\}} \nabla \ell(\mathbf{w}^\top [\mathbf{P}_{\setminus n} \mathbf{X}_{\setminus n}]_i, y_i) \\ &\approx \left[\nabla L(\mathbf{w}_\star) - \sum_{i \in \mathcal{V}_{\text{affect}}} \nabla \ell(\mathbf{w}_\star^\top [\mathbf{P}\mathbf{X}]_i, y_i) + \sum_{i \in \mathcal{V}_{\text{affect}} \setminus \{n\}} \nabla \ell(\mathbf{w}_\star^\top [\mathbf{P}_{\setminus n} \mathbf{X}_{\setminus n}]_i, y_i) \right] \\ &\quad + \left[\nabla^2 L(\mathbf{w}_\star) - \sum_{i \in \mathcal{V}_{\text{affect}}} \nabla^2 \ell(\mathbf{w}_\star^\top [\mathbf{P}\mathbf{X}]_i, y_i) + \sum_{i \in \mathcal{V}_{\text{affect}} \setminus \{n\}} \nabla^2 \ell(\mathbf{w}_\star^\top [\mathbf{P}_{\setminus n} \mathbf{X}_{\setminus n}]_i, y_i) \right] (\mathbf{w}_{\setminus \mathcal{V}_{\text{affect}}} - \mathbf{w}_\star) \\ &\underset{(a)}{=} \underbrace{\left[- \sum_{i \in \mathcal{V}_{\text{affect}}} \nabla \ell(\mathbf{w}_\star^\top [\mathbf{P}\mathbf{X}]_i, y_i) + \sum_{i \in \mathcal{V}_{\text{affect}} \setminus \{n\}} \nabla \ell(\mathbf{w}_\star^\top [\mathbf{P}_{\setminus n} \mathbf{X}_{\setminus n}]_i, y_i) \right]}_{\mathbf{v}} \\ &\quad + \underbrace{\left[\nabla^2 L(\mathbf{w}_\star) - \sum_{i \in \mathcal{V}_{\text{affect}}} \nabla^2 \ell(\mathbf{w}_\star^\top [\mathbf{P}\mathbf{X}]_i, y_i) + \sum_{i \in \mathcal{V}_{\text{affect}} \setminus \{n\}} \nabla^2 \ell(\mathbf{w}_\star^\top [\mathbf{P}_{\setminus n} \mathbf{X}_{\setminus n}]_i, y_i) \right]}_{\mathbf{H}} (\mathbf{w}_{\setminus \mathcal{V}_{\text{affect}}} - \mathbf{w}_\star). \end{aligned} \tag{13}$$

As a result, we can approximate $\mathbf{w}_{\setminus \mathcal{V}_{\text{affect}}}$ by

$$\mathbf{w}_{\setminus \mathcal{V}_{\text{affect}}} = \mathbf{w}_\star + \mathbf{H}^{-1} \mathbf{v}, \tag{14}$$

where both Hessian $\mathbf{H}$ and gradient $\mathbf{v}$ are defined in Eq. 13, which might induced massive computation cost as the number of affected nodes $|\mathcal{V}_{\text{affect}}|$ goes exponentially with respect to the number of layers.

D Connections between differential privacy and machine unlearning

The biggest difference between differential privacy and unlearning is whether the effect of data is removed from the model parameters. Please notice that the two methods can be used in parallel. For example, when using approximate unlearning Guo et al. (2020); Golatkar et al. (2020, 2021), since these methods could not guarantee a perfect data removal but just approximately removed, they propose to use differential privacy with their approximation unlearning method to further protect information leakage. More specifically,

- *Differential privacy* is designed to protect against the privacy leakage issue. In particular, they want to make a model trained on two different datasets behave similarly. The most widely accepted method is to control how much a model learned from each training example by adding random noise.
- *Machine unlearning* is designed to remove the effect of a data point on the pre-trained. For example in approximate unlearning Guo et al. (2020); Golatkar et al. (2020); Wu et al. (2020a), we want to make the re-training from scratch model $\mathbf{w}_u$ behaves similar to the model after unlearning $\mathbf{w}_p$, but whether data are perfectly removed are not guaranteed; in exact unlearning Chen et al. (2021), we want to make the re-training from scratch model $\mathbf{w}_u$ behaves similar to the model after unlearning $\mathbf{w}_p$, but guarantee the data are perfectly removed.

In the following, we will answer two questions related to differential privacy and unlearning:

Q1: Do we need to unlearning if a model is (ϵ, δ)-differential privacy?

If we looking for approximately remove the trace of private data, then a differential privacy method is enough (but may not be as efficient as approximate unlearning methods Guo et al. (2020)). However, if we are looking for perfectly remove the trace of private data, then differential privacy is not enough. To see this, let us first recall the definition of (ϵ, δ)-differential privacy.

Definition 1 ((ϵ, δ)-differential privacy) *Let $\epsilon > 0$ be a positive real number, $\mathcal{D}, \mathcal{D}'$ denote any two datasets that differ one a single element, $\mathcal{A}$ be a randomized algorithm that takes a dataset $\mathcal{D}, \mathcal{D}'$ as input, $\mathcal{S}$ denote any subset of the image of $\mathcal{A}$. Then, we say $\mathcal{A}$ is (ϵ, δ)-differential privacy if*

$$P[\mathcal{A}(\mathcal{D}) \in \mathcal{S}] \leq \exp(\epsilon) \cdot P[\mathcal{A}(\mathcal{D}') \in \mathcal{S}] + \delta.$$

To generalize the differential privacy definition to machine unlearning, let us think of $\mathcal{D}$ be the original dataset and $\mathcal{D}' = \mathcal{D} \setminus \{(\mathbf{x}_i, y_i)\}$ be the remaining dataset after delete data $(\mathbf{x}_i, y_i)$. Then, we can think of $\mathcal{A}$ as a composition of learning and unlearning algorithm:

$$\mathcal{A}(\mathcal{D}) = \text{Unlearn}\Big(\text{Learn}(\mathcal{D}), (\mathbf{x}_i, y_i)\Big), \mathcal{A}(\mathcal{D}') = \text{Unlearn}\Big(\text{Learn}(\mathcal{D}'), (\mathbf{x}_i, y_i)\Big) \tag{15}$$

More specifically, $\mathcal{A}(\mathcal{D})$ can be think of as first training a model on $\mathcal{D}$ then unlearn to produce the unlearned solution $\mathbf{w}_p$; $\mathcal{A}(\mathcal{D}')$ can be think of as re-training from scratch on $\mathcal{D}'$ with $\mathbf{w}_u = \mathcal{A}(\mathcal{D}')$ since $\mathcal{D}'$ does not contain $(\mathbf{x}_i, y_i)$. In other word, differential privacy algorithm can also guarantee the behavior of the the unlearning solution $\mathbf{w}_p$ similar to re-training from scratch solution $\mathbf{w}_u$, which leads to our approximate unlearning goal. However, $\mathbf{w}_p \approx \mathbf{w}_u$ not necessarily means $\mathbf{w}_p$ perfectly unlearn all the information related to the deleted data $(\mathbf{x}_i, y_i)$. Therefore, we cannot say a model is perfectly unlearned by using differential privacy method.

Q2: Can we have a differential privacy-like bounds for PROJECTOR*?* A reader familiar with differential privacy might expect a differential privacy-like bound similar to Definition 1. To understand why it is infeasible, let us first recall that ϵ term is the private budget that is related to the random noise distribution. Since our method is exact unlearning which can be shown from the algorithm level (i.e., "the support of the unlearned weight does not include the deleted node feature" could be guaranteed by the algorithm design), we don't need random noise and therefore we cannot show this kind of inequality. The reason why the approximate unlearning method Guo et al. (2020); Chien et al. (2022) has such a similar guarantee to differential privacy is because they are approximate unlearning method that use differential privacy type of random noise to unlearn.

E Why checking the closeness to re-trained solution along is not enough for unlearning?

Most approximate unlearning algorithms aim to generate the approximate unlearned model that is close to an exactly retrained model Wu et al. (2020a); Aldaghri et al. (2021); Izzo et al. (2021). However, as pointed out by Thudi et al. (2021); Guo et al. (2020), one cannot infer *"whether the data have been deleted"* solely from *"the closeness of the approximately unlearned and exactly retrained model"*. In fact, Thudi et al. (2021) empirically shows that one can even unlearn the data without modifying the parameters, which highlights the importance of showing the data removal guarantee from the algorithmic-level. In Theorem 2 below, we provide theoretical justification for the empirical observation in Thudi et al. (2021) under the binary classification setting and the proof is deferred to Appendix E.

Theorem 2 *Consider a general binary classification problem using logistic regression*

$$\min_{\mathbf{w} \in \mathbb{R}^d} f_{LR}(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \log\big(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)\big).$$

Upon receiving any request to unlearn $\mathbf{x}_i$, if it is misclassified by the optimal weight $\mathbf{w}_\star = \arg\min_{\mathbf{w}} f_{LR}(\mathbf{w})$, i.e., $y_i \mathbf{w}_\star^\top \mathbf{x}_i < 0$, we can unlearn without modifying the optimal weight $\mathbf{w}_\star$ as the optimally conditions are still satisfied.

An immediate implication of above theorem is that one could exactly unlearn a misclassified data point even without changing the model parameters. However, approximate unlearning methods Guo et al. (2020); Golatkar et al. (2020) cannot realize this from their algorithmic-level, which will output an estimated solution that is not only different from the exact unlearning solution but also could not fully unlearn the sensitive information. Although these methods borrow ideas from differential privacy (unlearning is different from the differential privacy, please refer to Appendix D for details) to add a

noise to unlearned model to avoid information leakage, this could also potentially deteriorate the accuracy of the unlearned model. Such observations highlights the importance of an algorithmic-level data removal guarantee over approximate unlearning.

The proof follows from standard optimality conditions. Let us consider a binary classification problem with N training samples using regularized logistic regression with the following empirical risk:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \; f(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \log\Big(1+\exp(-y_i\mathbf{w}^\top \mathbf{x}_i)\Big). \tag{16}$$

By utilizing the following inequality

$$\log(1+\exp(-z)) \geq \big(-\alpha\log\alpha - (1-\alpha)\log(1-\alpha)\big) - \alpha z, \; \forall \alpha\in[0,1], \tag{17}$$

we can lower bound the objective in Eq. 16 by

$$\begin{aligned} f(\mathbf{w}) &= \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \log\Big(1+\exp(-y_i\mathbf{w}^\top \mathbf{x}_i)\Big) \\ &\geq \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \big(-\alpha_i\log\alpha_i - (1-\alpha_i)\log(1-\alpha_i) - \alpha_i y_i \mathbf{w}^\top \mathbf{x}_i\big) \\ &= L(\mathbf{w},\boldsymbol{\alpha}), \end{aligned} \tag{18}$$

where the right hand size of inequality can be think of as a function of parameters $\mathbf{w}$ and $\boldsymbol{\alpha}$.

Optimal solution lies in linear span of input features. From the stationarity condition of the KKT conditions Boyd et al. (2004), we know that the gradient of $L(\mathbf{w},\boldsymbol{\alpha})$ with respect to $\mathbf{w}$ at the optimal point $\mathbf{w}^\star$ equals to zero, i.e.,

$$\frac{\partial L(\mathbf{w}_\star,\boldsymbol{\alpha})}{\partial \mathbf{w}} = \lambda\mathbf{w} - \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i = 0 \;\Rightarrow \mathbf{w}_\star = \frac{1}{\lambda}\sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i, \tag{19}$$

which implies that the optimal solution, regardless of the initialization, is a linear combination of all training samples $\{\mathbf{x}_1,\ldots,\mathbf{x}_N\}$.

Unlearning without modifying the parameters. By using the complementary slackness conditions of the KKT conditions, we know that for any $i\in\{1,\ldots,N\}$, we have

$$\alpha_i = 0 \;\vee\; y_i\mathbf{w}_\star^\top \mathbf{x}_i = -\log\alpha_i - \left(-1+\frac{1}{\alpha_i}\right)\log(1-\alpha_i) > 0. \tag{20}$$

From Eq. 20, we know that $\alpha_i \neq 0$ if and only if $\mathbf{x}_i$ can be correctly classified by optimal weight $\mathbf{w}_\star$, i.e., $y_i\mathbf{w}_\star^\top\mathbf{x}_i > 0$; otherwise, we have $\alpha_i = 0$ if $\mathbf{x}_i$ cannot be correctly classified by optimal weight $\mathbf{w}_\star$, which can be directly unlearned without requiring to modify the weight parameter $\mathbf{w}_\star$. When removing data $\mathbf{x}_i$ with dual variable as $\alpha_i = 0$, the KKT optimality condition still hold, similar to the main idea of SVM unlearning Cauwenberghs and Poggio (2000), therefore further updating the weight parameters is not required.

F Proof of Proposition 1

To find the coefficients of the orthogonal projection, let us write down the following $r = |\mathcal{V}_{\text{remain}}|$ simultaneous conditions on linear equations:

$$\langle \mathbf{x}_i, \mathbf{w} - \Pi_{\mathcal{U}}(\mathbf{w})\rangle = 0, \; \forall i \in \mathcal{V}_{\text{remain}}. \tag{21}$$

For notation simplicity, let $\mathbf{X}_{\text{remain}} = [\mathbf{x}_{r+1},\ldots,\mathbf{x}_N] \in \mathbb{R}^{r\times d}$ denote the remaining node features and $\boldsymbol{\alpha} = [\alpha_{r+1},\ldots,\alpha_N]\in\mathbb{R}^R$ denote the vectorized coordinates, then the Eq. 21 can be formulated as $\mathbf{X}_{\text{remain}}(\mathbf{w} - \mathbf{X}_{\text{remain}}^\top\boldsymbol{\alpha}) = \mathbf{0}$. As a result, we have

$$\mathbf{X}_{\text{remain}}(\mathbf{w} - \mathbf{X}_{\text{remain}}^\top\boldsymbol{\alpha}) = \mathbf{0} \Rightarrow \boldsymbol{\alpha} = (\mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger \mathbf{X}_{\text{remain}}\mathbf{w}. \tag{22}$$

However, notice that $\mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top$ is an $r \times r$ matrix and the computation of its inverse requires $\mathcal{O}(r^3)$ computation cost, which is computational prohibitive. As an alternative, we reconsider Eq. 22 from a different perspective by viewing it as the limit of the Ridge estimator with the Ridge parameter going to zero, i.e.,

$$(\mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger \mathbf{X}_{\text{remain}}\mathbf{w} = \lim_{\epsilon \to 0}(\epsilon \mathbf{I}_N + \mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger \mathbf{X}_{\text{remain}}\mathbf{w}. \tag{23}$$

Then, by using the Woodbury identity Golub and Van Loan (2013) we have

$$\begin{aligned}
&(\epsilon \mathbf{I}_N + \mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger \mathbf{X}_{\text{remain}}\mathbf{w} \\
&= \Big(\frac{1}{\epsilon}\mathbf{I}_N - \mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger \mathbf{X}_{\text{remain}}^\top\Big)\mathbf{X}_{\text{remain}}\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}}\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger (\epsilon \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger (\mathbf{I}_N + \epsilon \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}} - \mathbf{I}_N)\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} + \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger \mathbf{w} \\
&= \mathbf{X}_{\text{remain}}(\epsilon \mathbf{I}_d + \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger \mathbf{w}.
\end{aligned} \tag{24}$$

By taking the limit on both side, we have

$$\boldsymbol{\alpha} = \mathbf{X}_{\text{remain}}(\mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger \mathbf{w}, \tag{25}$$

where $\mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}}$ is an $d \times d$ matrix and its inverse requires $\mathcal{O}(d^3)$, which is much cheaper.

G Proof of Theorem 1

Let us define $\mathbf{w}(t),\ \mathbf{w}_u(t)$ as the weight parameters obtained by using full-batch GD training from scratch on $\mathcal{V}_{\text{train}},\ \mathcal{V}_{\text{remain}}$ for t epochs with the same initialization $\mathbf{w}(0) = \mathbf{w}_u(0)$, and $\mathbf{w}_p(t)$ denote the weight parameters obtained by applying PROJECTOR on $\mathbf{w}(t)$. To help reader better understand the proof strategy used in this section, we provide an overview on our proof strategy of Theorem 1. As shown in Figure 7, we derive the upper bound of $\|\mathbf{w}_u(T) - \mathbf{w}_p\|_2$ by first expanding the formula into two terms

$$\|\mathbf{w}_u(T) - \mathbf{w}_p\|_2 \leq \|\mathbf{w}_u(T) - \mathbf{w}(T)\|_2 + \|\mathbf{w}(T) - \mathbf{w}_p\|_2. \tag{26}$$

Then, we derive the upper bound of the first term in Appendix G.1 and the second term in Appendix G.2, and obtain the upper bound of $\|\mathbf{w}_u(T) - \mathbf{w}_p\|_2$. Then, by following the standard convergence analysis of smooth convex function, we obtain the upper bound training error $F^u(\tilde{\mathbf{w}}_p) - \min_{\mathbf{w}} F^u(\mathbf{w})$ in Appendix G.3.

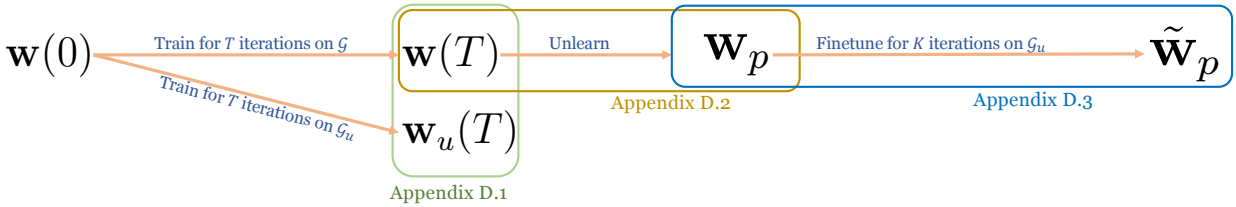


Figure 7: An illustration on the proof strategy of Theorem 1, where $\mathcal{G}$ stands for the graph used before node deletion and $\mathcal{G}_u$ stands for the graph after the node deletion.

G.1 Upper bound on $\|\mathbf{w}_u(T) - \mathbf{w}(T)\|_2$

Let first recall that the gradient of Eq. 1 and Eq. 3 are computed as

$$\begin{aligned}
\nabla F(\mathbf{w}) &= \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{i \in \mathcal{V}_{\text{train}}} \nabla f_i(\mathbf{w}),\ \nabla f_i(\mathbf{w}) = -y_i \sigma(-y_i \mathbf{w}^\top \mathbf{h}_i)\mathbf{h}_i + \lambda \mathbf{w}, \\
\nabla F^u(\mathbf{w}) &= \frac{1}{|\mathcal{V}_{\text{remain}}|} \sum_{i \in \mathcal{V}_{\text{remain}}} \nabla f_i^u(\mathbf{w}),\ \nabla f_i^u(\mathbf{w}) = -y_i \sigma(-y_i \mathbf{w}^\top \mathbf{h}_i^u)\mathbf{h}_i^u + \lambda \mathbf{w},
\end{aligned} \tag{27}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the Sigmoid function. From Eq. 27, we know that $f_i(\mathbf{w}), f_i^u(\mathbf{w})$ is $(\lambda + P_s^2 B_x^2)$-smoothness, which is shown as follows

$$\begin{aligned}
\|\nabla f_i(\mathbf{w}_1) - \nabla f_i(\mathbf{w}_2)\|_2 &= \left\| -y_i \sigma(-y_i \mathbf{w}_1^\top \mathbf{h}_i)\mathbf{h}_i + y_i \sigma(-y_i \mathbf{w}_2^\top \mathbf{h}_i)\mathbf{h}_i + \lambda(\mathbf{w}_1 - \mathbf{w}_2) \right\|_2 \\
&\leq \|y_i \mathbf{h}_i\|_2 \cdot |\sigma(-y_i \mathbf{w}_1^\top \mathbf{h}_i) + \sigma(-y_i \mathbf{w}_2^\top \mathbf{h}_i)| + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq \|y_i \mathbf{h}_i\|_2 \cdot \|y_i \mathbf{w}_1^\top \mathbf{h}_i - y_i \mathbf{w}_2^\top \mathbf{h}_i\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq \|y_i \mathbf{h}_i\|_2^2 \cdot \|\mathbf{w}_1 - \mathbf{w}_2\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq \left(\max_i \left\| [\mathbf{P}^L \mathbf{X}]_i \right\|_2^2 \right) \|\mathbf{w}_1 - \mathbf{w}_2\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq (\lambda + P_s^2 B_x^2) \|\mathbf{w}_1 - \mathbf{w}_2\|_2, \\
\|\nabla f_i^u(\mathbf{w}_1) - \nabla f_i^u(\mathbf{w}_2)\|_2 &\leq \left(\max_i \left\| [\mathbf{P}_u^L \mathbf{X}]_i \right\|_2^2 \right) \|\mathbf{w}_1 - \mathbf{w}_2\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq (\lambda + P_s^2 B_x^2) \|\mathbf{w}_1 - \mathbf{w}_2\|_2.
\end{aligned} \tag{28}$$

We can upper bound the difference of the gradient computed before and after data deletion by

$$\begin{aligned}
&\mathbb{E}\left[\|\nabla F(\mathbf{w}) - \nabla F^u(\mathbf{w})\|_2\right] \\
&= \mathbb{E}\left[\left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i(\mathbf{w}) - \frac{1}{|\mathcal{V}_{\text{remain}}|}\sum_{i\in\mathcal{V}_{\text{remain}}}\nabla f_i^u(\mathbf{w}_u)\right\|_2\right] \\
&\underset{(a)}{\leq} \left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i(\mathbf{w}) - \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i^u(\mathbf{w})\right\|_2 \\
&\quad + \mathbb{E}\left[\left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i^u(\mathbf{w}) - \frac{1}{|\mathcal{V}_{\text{remain}}|}\sum_{i\in\mathcal{V}_{\text{remain}}}\nabla f_i^u(\mathbf{w}_u)\right\|_2\right],
\end{aligned} \tag{29}$$

where (a) is achieved by adding and subtracting the same term and $\|\mathbf{a}+\mathbf{b}\|_2 \leq \|\mathbf{a}\|_2 + \|\mathbf{b}\|_2$ and the expectation on the randomness the deleted node selection.

The first term on the right hand side of inequality (a) can be further upper bounded by

$$\begin{aligned}
&\left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i - \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i^u)\mathbf{h}_i^u\right\|_2 \\
&\underset{(a)}{\leq} \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} \left\|y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i - y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i^u\right\|_2 + \\
&\quad + \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} \left\|y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i^u - y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i^u)\mathbf{h}_i^u\right\|_2 \\
&\leq \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} |y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)|\,\|\mathbf{h}_i - \mathbf{h}_i^u\|_2 + \\
&\quad + \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} \|y_i\mathbf{h}_i^u\|_2\,|\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i) - \sigma(-y_i\mathbf{w}^\top\mathbf{h}_i^u)| \\
&\underset{(b)}{\leq} \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} (\|\mathbf{h}_i - \mathbf{h}_i^u\|_2 + \|\mathbf{h}_i^u\|_2\|\mathbf{w}\|_2\|\mathbf{h}_i - \mathbf{h}_i^u\|_2) \\
&= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} (1 + \|\mathbf{h}_i^u\|_2\|\mathbf{w}\|_2)\,\|\mathbf{h}_i - \mathbf{h}_i^u\|_2,
\end{aligned} \tag{30}$$

where (a) is due to $\|\mathbf{a}+\mathbf{b}\|_2 \leq \|\mathbf{a}\|_2 + \|\mathbf{b}\|_2$ and (b) is due to $|y_i| = 1$ and $|\sigma(x) - \sigma(y)| \leq |x-y|$. By using the definition of $\mathbf{h}_i^u$ and $\mathbf{h}_i$ and Assumption 1, we have for any i

$$\begin{aligned}
\|\mathbf{h}_i - \mathbf{h}_i^u\|_2 &= \max_j \|[\mathbf{P}^L\mathbf{X} - \mathbf{P}_u^L\mathbf{X}]_j\|_2 \\
&\leq \max_j \|\mathbf{x}_j\|_2 \cdot \max_j \left\|[\mathbf{P}^L - \mathbf{P}_u^L]_j\right\|_2 \\
&\leq B_x P_d, \\
\|\mathbf{h}_i^u\|_2 &= \max_j \|[\mathbf{P}_u^L\mathbf{X}]_j\|_2 \\
&\leq \max_j \|\mathbf{x}_j\|_2 \cdot \max_j \left\|[\mathbf{P}_u^L]_j\right\|_2 \\
&\leq B_x P_s.
\end{aligned} \tag{31}$$

Besides, the upper bound of the second term on the right hand side of inequality (a) is from Assumption 2. By plugging the results back to the right hand side of inequality (a), we have

$$\|\nabla F(\mathbf{w}) - \nabla F^u(\mathbf{w})\|_2 \leq (1 + B_x B_w P_s) B_x P_d + G. \tag{32}$$

Therefore, according to the gradinet update rule in gradient descent, we can bound the change of model parameters $\mathbf{w}_u(t)$ and $\mathbf{w}(t)$ by

$$\begin{aligned}
&\|\mathbf{w}_u(t+1) - \mathbf{w}(t+1)\|_2 \\
&= \|\mathbf{w}_u(t) - \eta\nabla F^u(\mathbf{w}_u(t)) - \mathbf{w}(t) + \eta\nabla F(\mathbf{w}(t))\|_2 \\
&\leq \|\mathbf{w}_u^t - \mathbf{w}^t\|_2 + \eta\|\nabla F^u(\mathbf{w}_u(t)) - \nabla F(\mathbf{w}_u(t)) + \nabla F(\mathbf{w}_u(t)) - \nabla F(\mathbf{w}(t))\|_2 \\
&\leq (1+\eta(\lambda + B_x^2 B_s^2))\|\mathbf{w}_u^t - \mathbf{w}^t\|_2 + \eta\|\nabla F^u(\mathbf{w}_u(t)) - \nabla F(\mathbf{w}_u(t))\|_2 \\
&\underset{(a)}{\leq} (1+\eta(\lambda + B_x^2 B_s^2))\|\mathbf{w}_u^t - \mathbf{w}^t\|_2 + \eta\Big((1 + B_x B_w P_s)B_x P_d + G\Big),
\end{aligned} \tag{33}$$

where (a) is due to Eq. 28 and Eq. 32. Then after T iterations, we can bound the different between two parameters as

$$\|\mathbf{w}_u(T) - \mathbf{w}(T)\|_2 \leq \eta\Big((1 + B_x B_w P_s)B_x P_d + G\Big)\sum_{t=1}^{T}(1+\eta(\lambda + B_x^2 B_s^2))^{t-1}. \tag{34}$$

G.2 Upper bound on $\|\mathbf{w}_p - \mathbf{w}(T)\|_2$ for PROJECTOR

From Proposition 1 we know that there exist a set of coordinates $\{\beta_i \mid i \in \mathcal{V}\}$ such that $\mathbf{w}(T) = \sum_{j\in\mathcal{V}} \beta_j \mathbf{x}_j$ holds. Besides, by using Eq. 27, we have

$$\begin{aligned}
\nabla F(\mathbf{w}(t)) &= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)\mathbf{h}_i + \lambda\mathbf{w}(t) \\
&= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)[\mathbf{P}^L\mathbf{X}]_i + \lambda\mathbf{w}(t) \\
&= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)\left(\sum_{j\in\mathcal{V}}[\mathbf{P}^L]_{ij}\mathbf{x}_j\right) + \lambda\mathbf{w}(t) \\
&= \sum_{j\in\mathcal{V}}\left(\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)[\mathbf{P}^L]_{ij}\right)\mathbf{x}_j + \lambda\mathbf{w}(t).
\end{aligned} \tag{35}$$

Then, according to the gradient descent update rule $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta\nabla F(\mathbf{w}(t))$, we have

$$\begin{aligned}
\mathbf{w}(t+1) &= (1-\eta\lambda)\times\mathbf{w}(t) - \eta\sum_{j\in\mathcal{V}}\left(\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)[\mathbf{P}^L]_{ij}\right)\mathbf{x}_j \\
&= \sum_{k=0}^{t}\eta(1-\eta\lambda)^{t-k}\left[\sum_{j\in\mathcal{V}}\left(\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(k)\mathbf{h}_i)[\mathbf{P}^L]_{ij}\right)\mathbf{x}_j\right]
\end{aligned} \tag{36}$$

Then, we know that after T iterations of gradient updates, for any $j \in \mathcal{V}$ we have

$$\mathbf{w}(T) = \sum_{j\in\mathcal{V}}\beta_j\mathbf{x}_j, \text{ where } \beta_j \leq \eta T\cdot\max_{i\in\mathcal{V}_{\text{train}}}[\mathbf{P}^L]_{ij} \leq \eta T, \tag{37}$$

the first inequality is due to $\lambda\eta < 1, |y_i| = 1, 0 < \sigma(x) < 1$ and the second inequality hold because any element $0 \leq [\mathbf{P}^L]_{ij} \leq 1$.

After projection, we find another set of $\{\alpha_i \mid i \in \mathcal{V}_{\text{remain}}\}$ and construct the unlearned weight parameter by $\mathbf{w}_p =$

$\sum_{i\in\mathcal{V}_{\text{remain}}}\alpha_i\mathbf{x}_i$. Then, the upper bound on $\|\mathbf{w}_p-\mathbf{w}(T)\|_2$ can be written as

$$\begin{aligned}
\|\mathbf{w}_p-\mathbf{w}(T)\|_2 &= \min_{\boldsymbol{\alpha}}\left\|\sum_{i\in\mathcal{V}_{\text{remain}}}\alpha_i\mathbf{x}_i-\sum_{i\in\mathcal{V}}\beta_i\mathbf{x}_i\right\|_2\\
&= \min_{\boldsymbol{\alpha}}\left\|\sum_{i\in\mathcal{V}_{\text{remain}}}(\alpha_i-\beta_i)\mathbf{x}_i-\sum_{i\in\mathcal{V}_{\text{delete}}}\beta_i\mathbf{x}_i\right\|_2\\
&\le \sum_{i\in\mathcal{V}_{\text{delete}}}\beta_j\cdot\min_{\boldsymbol{\alpha}}\left\|\sum_{i\in\mathcal{V}_{\text{remain}}}\frac{\alpha_i-\beta_i}{\beta_j}\mathbf{x}_i-\mathbf{x}_j\right\|_2\\
&\le |\mathcal{V}_{\text{delete}}|\cdot\max_{j\in\mathcal{V}_{\text{delete}}}\beta_j\cdot\min_{\boldsymbol{\alpha}}\left\|\sum_{i\in\mathcal{V}_{\text{remain}}}\frac{\alpha_i-\beta_i}{\beta_j}\mathbf{x}_i-\mathbf{x}_j\right\|_2\\
&\le \eta T|\mathcal{V}_{\text{delete}}|\cdot\underbrace{\max_{j\in\mathcal{V}_{\text{delete}}}\min_{\boldsymbol{\alpha}}\left\|\sum_{i\in\mathcal{V}_{\text{remain}}}\frac{\alpha_i-\beta_i}{\beta_j}\mathbf{x}_i-\mathbf{x}_j\right\|_2}_{(a)}.
\end{aligned}\tag{38}$$

Notice that (a) is equivalent to finding another set of coefficient $\boldsymbol{\gamma}$ that

$$\max_{j\in\mathcal{V}_{\text{delete}}}\min_{\boldsymbol{\gamma}}\left\|\sum_{i\in\mathcal{V}_{\text{remain}}}\gamma_i\mathbf{x}_i-\mathbf{x}_j\right\|_2\le\delta,\tag{39}$$

where the upper bound is due to Assumption 3. Therefore, we have

$$\|\mathbf{w}_p-\mathbf{w}(T)\|_2\le\delta\eta T|\mathcal{V}_{\text{delete}}|.\tag{40}$$

G.3 Convergence rate for fune-tuning

Let $\mathbf{w}_p(k)$ denote fune-tuning on $\mathbf{w}_p$ for k iterations, where $\mathbf{w}_p(0)=\mathbf{w}_p$ and $\mathbf{w}_p(K)=\tilde{\mathbf{w}}_p$ as we used in Theorem 1.. By knowing F^u is $(\lambda+B_x^2P_s^2)$-smoothness, we have

$$\begin{aligned}
&F^u(\mathbf{w}_p(k+1))\\
&\underset{(a)}{\le} F^u(\mathbf{w}_p(k))+\langle\nabla F^u(\mathbf{w}_p(k)),\mathbf{w}_p(k+1)-\mathbf{w}_p(k)\rangle+\frac{(\lambda+B_x^2P_s^2)}{2}\|\mathbf{w}_p(k+1)-\mathbf{w}_p(k)\|_2^2\\
&= F^u(\mathbf{w}_p(k))-\eta\|\nabla F^u(\mathbf{w}_p(k))\|_2^2+\frac{\eta^2(\lambda+B_x^2P_s^2)}{2}\|\nabla F^u(\mathbf{w}_p(k))\|_2^2\\
&= F^u(\mathbf{w}_p(k))-\eta\left(1-\frac{\eta_k(\lambda+B_x^2P_s^2)}{2}\right)\|\nabla F^u(\mathbf{w}_p(k))\|_2^2,
\end{aligned}\tag{41}$$

where inequality (a) is due to the update rule $\mathbf{w}_p(k+1)=\mathbf{w}_p(k)-\eta\nabla F^u(\mathbf{w}_p(k))$.

Let $\mathbf{w}_\star=\arg\min_{\mathbf{w}}F^u(\mathbf{w})$. By choosing $\eta=\frac{2}{(\lambda+B_x^2P_s^2)}$, we have

$$\Big(F^u(\mathbf{w}_p(k+1))-F^u(\mathbf{w}_\star)\Big)-\Big(F^u(\mathbf{w}_p(k))-F^u(\mathbf{w}_\star)\Big)\le-\frac{1}{2(\lambda+B_x^2P_s^2)}\|\nabla F^u(\mathbf{w}_p(k))\|_2^2.\tag{42}$$

Since function F^u is convex, we know the following inequality holds:

$$\begin{aligned}
F^u(\mathbf{w}_p(k+1))-F^u(\mathbf{w}_\star)&\le\langle\nabla F^u(\mathbf{w}_p(k)),\mathbf{w}_p(k)-\mathbf{w}_\star\rangle\\
&\le\|F^u(\mathbf{w}_p(k))\|_2\|\mathbf{w}_p(k)-\mathbf{w}_\star\|_2.
\end{aligned}\tag{43}$$

By plugging it back to Eq. 42, we have

$$\begin{aligned} &F^u(\mathbf{w}_p(k+1)) - F^u(\mathbf{w}_\star) \\ &\leq (F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star)) \left(1 - \frac{F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star)}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p(k) - \mathbf{w}_\star\|_2^2}\right) \\ &\underset{(a)}{\leq} (F^u(\mathbf{w}_p) - F^u(\mathbf{w}_\star)) \left(1 - \frac{F^u(\mathbf{w}_p(T)) - F^u(\mathbf{w}_\star)}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}\right), \end{aligned} \tag{44}$$

where inequality (a) is due to $1 \leq k \leq K$. Since $F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star) \leq 2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2$, we have

$$\frac{1}{F^u(\mathbf{w}_p(k+1)) - F^u(\mathbf{w}_\star)} \geq \frac{1}{F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star)} + \frac{1}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}. \tag{45}$$

Telescoping from $k = T, \ldots, T+K$, we get

$$\frac{1}{F^u(\tilde{\mathbf{w}}_p) - F^u(\mathbf{w}_\star)} \geq \frac{1}{F^u(\mathbf{w}_p) - F^u(\mathbf{w}_\star)} + \frac{T}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}, \tag{46}$$

which implies

$$\frac{1}{F^u(\tilde{\mathbf{w}}_p) - F^u(\mathbf{w}_\star)} \leq \frac{2(\lambda + B_x^2 P_s^2)(F^u(\mathbf{w}_p - F^u(\mathbf{w}_\star))\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}{T(F^u(\mathbf{w}_p - F^u(\mathbf{w}_\star)) + 2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}. \tag{47}$$

By using the smoothness at $\mathbf{w}_\star$, we have

$$F^u(\mathbf{w}_p) - F^u(\mathbf{w}_\star) \leq \frac{(\lambda + B_x^2 P_s^2)}{2}\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2. \tag{48}$$

Plugging back to Eq. 47, suppose T is large enough and $\mathbf{w}_u(T) \approx \mathbf{w}_\star$, we have

$$\begin{aligned} F^u(\tilde{\mathbf{w}}_p) - F^u(\mathbf{w}_\star) &\leq \frac{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}{K+4} \\ &\approx \mathcal{O}\left(\frac{(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_u(T)\|_2^2}{K}\right). \end{aligned} \tag{49}$$

H Proof on Proposition 2

The influence-based unlearning approach Guo et al. (2020) unlearn by using second-order gradient update on the weight parameters. To apply Guo et al. (2020) onto a L-layer linear GNN, due to the node dependency, we have to unlearn all the L-hop neighbors of the deleted nodes. Therefore, the generalization of Guo et al. (2020) to graph requires updating the weight parameters by

$$\mathbf{w}_p = \mathbf{w}(T) - \left[\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}_{\text{remain}}^L)\right]^{-1} \nabla F(\mathbf{w}(T),\ \mathcal{V}_{\text{delete}}^L), \tag{50}$$

where $\mathcal{V}_{\text{delete}}^L = \text{unique}\{v_j \mid \text{SPD}(v_i, v_j) < L,\ v_i \in \mathcal{V}_{\text{delete}}\}$ denotes the set of nodes that has shortest path distance (SPD) less than L to nodes in $\mathcal{V}_{\text{delete}}$, $\mathcal{V}_{\text{remain}}^L = \mathcal{V} \setminus \mathcal{V}_{\text{delete}}^L$, $\nabla^2 F(\mathbf{w},\ \mathcal{V}_{\text{remain}}^L)$ denote computing the Hessain on $\mathcal{V}_{\text{remain}}^L$, and $\nabla F(\mathbf{w},\ \mathcal{V}_{\text{delete}}^L)$ denote computing the gradient on $\mathcal{V}_{\text{remain}}^L$.

To prove Proposition 2, we need to first analyze the upper bound on $\|\mathbf{w}_p - \mathbf{w}(T)\|_2$ for INFLUENCE Guo et al. (2020) by

$$\begin{aligned} \|\mathbf{w}_p - \mathbf{w}(T)\|_2 &= \left\| \left[\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}_{\text{remain}}^L)\right]^{-1} \nabla F(\mathbf{w}(T),\ \mathcal{V}_{\text{delete}}^L) \right\|_2 \\ &\leq \left\| \left[\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}_{\text{remain}}^L)\right]^{-1} \right\|_2 \left\| \nabla F(\mathbf{w}(T),\ \mathcal{V}_{\text{delete}}^L) \right\|_2. \end{aligned} \tag{51}$$

Let us first upper bound $\left\|\nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2$ by

$$\begin{aligned}
&\left\|\nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2 \\
&= \left\| \sum_{j\in\mathcal{V}} \left(\frac{1}{|\mathcal{V}_{\text{delete}}|} \sum_{i\in\mathcal{V}_{\text{delete}}} -y_i \sigma(-y_i \mathbf{w}^\top(T)\mathbf{h}_i)[\mathbf{P}^L]_{ij} \right) \mathbf{x}_j + \lambda \mathbf{w}(T) \right\|_2 \\
&\leq \left\| \sum_{j\in\mathcal{V}} \left(\frac{1}{|\mathcal{V}_{\text{delete}}|} \sum_{i\in\mathcal{V}_{\text{delete}}} -y_i \sigma(-y_i \mathbf{w}^\top(T)\mathbf{h}_i)[\mathbf{P}^L]_{ij} \right) \mathbf{x}_j \right\|_2 + \lambda \|\mathbf{w}(T)\|_2 \\
&\underset{(a)}{\leq} B_x|\mathcal{V}| + \lambda\|\mathbf{w}(T)\|_2,
\end{aligned} \tag{52}$$

where (a) is due to $|y_i| = 1,\ 0 < \sigma(x) < 1$, and $[\mathbf{P}^L]_{ij} \leq 1$. Meanwhile, from Eq. 36, we can upper bound $\|\mathbf{w}(T)\|_2$ by

$$\begin{aligned}
\|\mathbf{w}(T)\|_2 &= \left\| \sum_{k=0}^{T-1} \eta(1-\eta\lambda)^{T-1-k} \left[\sum_{j\in\mathcal{V}} \left(\frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{i\in\mathcal{V}_{\text{train}}} -y_i \sigma(-y_i \mathbf{w}^\top(k)\mathbf{h}_i)[\mathbf{P}^L]_{ij} \right) \mathbf{x}_j \right] \right\|_2 \\
&\leq \eta T \times B_x|\mathcal{V}|
\end{aligned} \tag{53}$$

By plugging the result back, we have

$$\left\|\nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2 \leq (1+\lambda\eta T)B_x|\mathcal{V}|. \tag{54}$$

Knowing that $\left\|\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2 > \lambda$ due to the strongly convexity of objective function $F(\mathbf{w})$, we have

$$\left(\left\|\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2\right)^{-1} \leq \frac{1}{\lambda} \tag{55}$$

Therefore, we know that

$$\|\mathbf{w}_p - \mathbf{w}(T)\|_2 \leq \frac{(1+\lambda\eta T)B_x|\mathcal{V}|}{\lambda}. \tag{56}$$

By comparing Eq. 40 and Eq. 56, we know that if

$$\delta < \left(\frac{1}{\lambda\eta T} + 1\right) B_x \times \frac{|\mathcal{V}|}{|\mathcal{V}_{\text{delete}}|} \tag{57}$$

the solution of PROJECTOR if provable closer to the retraining from scratch than Guo et al. (2020). Moreover, the above discussion also holds for Golatkar et al. (2020) by replacing the variable $\mathcal{V}_{\text{delete}}$ in Eq. 50, 51, and 52 as $\mathcal{V}_{\text{remain}}$.

I Proof of Proposition 3

The proof is an application of the proof of Theorem 4.1 in Wang and Zhang (2022) to the linear-GNN structure $g_{\mathbf{w}}(\mathbf{L},\mathbf{X}) = \sum_{\ell=1}^{n}(\mathbf{P}^{\ell-1}\mathbf{X})\mathbf{w}_\ell$ that we used in the experiment. Please notice that $g_{\mathbf{w}}(\mathbf{L},\mathbf{X})$ is equivalent to first concatenating all polynomial graph convolutions then apply a single weight vector

$$g_{\mathbf{w}}(\mathbf{L},\mathbf{X}) = [\mathbf{X} \,||\, \mathbf{P}\mathbf{X} \,||\, \ldots \,||\, \mathbf{P}^{n-1}\mathbf{X}]\mathbf{w},$$

where $[\mathbf{A} \,||\, \mathbf{B}] \in \mathbb{R}^{n\times 2d}$ is concatenating matrices $\mathbf{A},\mathbf{B} \in \mathbb{R}^{n\times d}$ along their feature dimension. We assume $\mathbf{y} = f(\mathbf{P},\mathbf{X}) \in \mathbb{R}^{n\times 1}$ as the target function we want to approximate by linear-GNN.

Let us define $\mathbf{U},\boldsymbol{\lambda}$ as the eigenvectors and eigenvalues of graph propagation matrix $\mathbf{P}$. Then, the linear-GNN could be written as

$$g_{\mathbf{w}}(\mathbf{L},\mathbf{X}) = \sum_{\ell=1}^{n}(\mathbf{P}^{\ell-1}\mathbf{X})\mathbf{w}_\ell = \sum_{\ell=1}^{n}\mathbf{U}\Lambda^{\ell-1}\mathbf{U}\mathbf{X}\mathbf{w}_\ell,\ \text{where } [\Lambda^{\ell-1}]_{i,i} = \lambda_i^{\ell-1}$$

Since we assume all rows in $\mathbf{U}\mathbf{X}$ is non-zero vectors, we know that there always exists a set of $\mathbf{w}_\ell,\ \forall \ell \in \{1,\ldots,k\}$ such

that all elements in $\mathbf{U}\mathbf{X}\mathbf{w}_\ell,\ \forall \ell \in \{1,\dots,k\}$ is non-zero. For example, we can select $\mathbf{w}_\ell$ from $\mathbb{R}^d$ that is not orthogonal to all vectors in $\mathbf{U}\mathbf{X}$ and not equal to zero vector. Let us denote $\mathbf{z}_\ell = \mathbf{U}\mathbf{X}\mathbf{w}_\ell \in \mathbb{R}^{n\times 1}$, then our linear-GNN could be written as

$$g_{\mathbf{w}}(\mathbf{L},\mathbf{X}) = \sum_{\ell=1}^{n} \mathbf{U}\left(\boldsymbol{\lambda}^{\ell-1}\cdot \mathbf{z}_\ell\right) = \mathbf{U}\left(\sum_{\ell=1}^{n}\left(\boldsymbol{\lambda}^{\ell-1}\cdot \mathbf{z}_\ell\right)\right),$$

where $\mathbf{a}\cdot\mathbf{b}$ is element-wise dot product between $\mathbf{a},\mathbf{b}$.

In order to use $g_{\mathbf{w}}(\mathbf{L},\mathbf{X})$ approximate any function $\mathbf{y} = f(\mathbf{P},\mathbf{X}) \in \mathbb{R}^{n\times 1}$, we have to make sure

$$\mathbf{U}\left(\sum_{\ell=1}^{n}\left(\boldsymbol{\lambda}^{\ell-1}\cdot \mathbf{z}_\ell\right)\right) = \mathbf{y} \rightarrow \left(\sum_{\ell=1}^{n}\left(\boldsymbol{\lambda}^{\ell-1}\cdot \mathbf{z}_\ell\right)\right) = \mathbf{U}^\top \mathbf{y},$$

Let us write as $\mathbf{w}_\ell = \mathbf{w}\cdot\alpha_\ell$, then we have the following equality

$$\underbrace{\begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \dots & \lambda_1^{n-1} \\ 1 & \lambda_2 & \lambda_2^2 & \dots & \lambda_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \lambda_n & \lambda_n^2 & \dots & \lambda_n^{n-1} \end{bmatrix}}_{\mathbf{M}\in\mathbb{R}^{n\times n}} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} [\mathbf{U}^\top\mathbf{y}]_1/[\mathbf{U}\mathbf{X}\mathbf{w}]_1 \\ [\mathbf{U}^\top\mathbf{y}]_2/[\mathbf{U}\mathbf{X}\mathbf{w}]_2 \\ \vdots \\ [\mathbf{U}^\top\mathbf{y}]_n/[\mathbf{U}\mathbf{X}\mathbf{w}]_n \end{bmatrix}$$

If no elements in $\boldsymbol{\lambda}$ is identical, $\mathbf{M}$ is inversible and there is always exists a unique set of $\alpha_i,\ i\in\{1,\dots,n\}$ that satisfy the above equality.

J Connection and potential application to SVM unlearning

The KKT-based unlearning has been studied in the SVM unlearning Cauwenberghs and Poggio (2000); Karasuyama and Takeuchi (2009); Gâlmeanu and Andonie (2008); Diehl and Cauwenberghs (2003) dated back to the last two decades. Due to the great similarity between SVM and logistic regression, it is interesting to compare it with our projection-based unlearning. Although generalizing the KKT-based unlearning from SVM to logistic regression is non-trivial, the other way around is possible according to Proposition 4.

Proposition 4 *When training SVM using primal gradient descent with the initial solution $\mathbf{w}_0 \in$ span$\{\mathbf{x}_1,\dots,\mathbf{x}_N\}$ (e.g., Pegasos Shalev-Shwartz et al. (2011)) or using dual coordinate ascent (e.g.,SMO Platt (1998)), the corresponding primal solution after t iterations always satisfy $\mathbf{w}_t \in$ span$\{\mathbf{x}_1,\dots,\mathbf{x}_N\}$.*

Therefore, we limit our following discussion to linear SVM unlearning, where the primal objective function is defined as

$$f_{\text{SVM}}(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N}\max(0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i), \tag{58}$$

and its dual objective function is defined as

$$\begin{aligned} g_{\text{SVM}}(\boldsymbol{\alpha}) &= -\frac{1}{2\lambda}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle + \sum_{i=1}^{N}\alpha_i, \\ \text{subject to } &\sum_{i=1}^{N}\alpha_i y_i = 0,\ \alpha_i\in[0,1]. \end{aligned} \tag{59}$$

J.1 Existing SVM unlearning and its limitation

SVM unlearning Cauwenberghs and Poggio (2000) investigates how to maintain the KKT optimality condition when data are slightly changed. They propose to quickly identify the dual variable α_i for each data point $\mathbf{x}_i$ and solve the linear system that maintain the KKT optimality condition. In practice, they require maintaining a $N\times N$ kernel matrix for enlarging or shrinking, which is memory consuming and engineering effort prohibitive when N is large Laskov et al. (2006). Moreover,

their method requires multiple iterations to unlearn a single data point, which could potentially be inefficient. To see this, let us first classify all data points into three categories according to its geometric position to the marginal hyperplanes:

- Outside the marginal hyperplanes $\mathcal{O} = \{i \mid y_i \mathbf{w}_\star^\top \mathbf{x}_i > 1,\ \alpha_i = 0\}$,
- On the marginal hyperplanes $\mathcal{M} = \{i \mid y_i \mathbf{w}_\star^\top \mathbf{x}_i = 1,\ 0 \leq \alpha_i \leq 1\}$,
- Between the marginal hyperplanes $\mathcal{I} = \{i \mid y_i \mathbf{w}_\star^\top \mathbf{x}_i < 1,\ \alpha_i = 1\}$.

Besides, according to the KKT condition, the optimal primal solution $\mathbf{w}_\star$ and its prediction on any data point $\hat{y}_i$ can be expanded as

$$\mathbf{w}_\star = \frac{1}{\lambda} \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \text{ and } \hat{y}_i = \mathbf{w}_\star^\top \mathbf{x}_i = \frac{1}{\lambda} \sum_{j \in \mathcal{M} \cup \mathcal{I}} \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle. \tag{60}$$

To delete data point $(\mathbf{x}_c, y_c)$, Cauwenberghs and Poggio (2000) needs to decrease the corresponding Lagrangian parameters α_c to 0, meanwhile keep the optimal conditions of other parameters satisfied, i.e., for any $i \in \mathcal{M}$ we have

$$\Delta \alpha_c y_c \langle \mathbf{x}_i, \mathbf{x}_c \rangle + \sum\nolimits_{j \in \mathcal{M}} \Delta \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle = 0, \tag{61}$$

where $\Delta \alpha_j$ denote the amount of the change of variable α_j. The update direction of each dual variable can be obtained by solving the linear system with respect to each $\Delta \alpha_j$. The update step size is selected as the largest step length under the condition that no element moves across $\mathcal{M}, \mathcal{O}$, and $\mathcal{I}$. When any $\alpha_j,\ \forall j \in \mathcal{M} \cup \{c\}$ is increased to 1 or decreased to 0, we have to move one point from one set to another, and repeat the above process multiple iterations until stable. Since every iteration only one element is moving across $\mathcal{M}, \mathcal{O}$, and $\mathcal{I}$, we have to solve $|\mathcal{M}|$ linear equations multiple iterations[6], each of which requires to inverse a $|\mathcal{M}| \times |\mathcal{M}|$ matrix, which could result in computation overhead of $\mathcal{O}(|\mathcal{M}|^3)$ for unknown number of iterations, which is not guaranteed to be faster than re-training the data from scratch Tsai et al. (2014).

J.2 PROJECTOR for SVM unlearning

As an alternative, under the assumption that slightly dataset change only cause minor change on the optimal weight parameters, we propose to apply PROJECTOR directly to primal solution of SVM and then fine-tune for several iterations using gradient descent methods (e.g., Pegasos Shalev-Shwartz et al. (2011)). We note that our primal SVM unlearning method shares the same spirit with Tsai et al. (2014), in which they propose an approximate unlearning method that directly finetune the primal solution on the new dataset (without the deleted data points), therefore the sensitive information are not guaranteed to be perfectly removed. In contrast, our projection-based method could provide such guarantee for Tsai et al. (2014). We leave this an an interesting future direction which could explore the idea in the future.

[6]However, the number of iterations is unknown, which could be extremely large when comparing to $|\mathcal{M}|$.