Privacy Matters! Efficient Graph Representation Unlearning with Data Removal Guarantee

Weilin Cong
Penn State
weilin@psu.edu

Mehrdad Mahdavi
Penn State
mzm616@psu.edu

Abstract

As privacy protection receives much attention, unlearning the effect of a specific node from a pre-trained model has become equally important as graph representation learning. However, due to the *node dependency* in the graph, graph representation unlearning is challenging and is still less well explored, even on the linear-GNN structure[1]. In this paper, we fill in this gap by studying the unlearning problem on the linear-GNN. To unlearn the knowledge of a specific node, we propose a projection-based unlearning approach PROJECTOR that projects the weight parameters of the pre-trained model onto a subspace that is irrelevant to the deleted node features. PROJECTOR could overcome the challenges caused by node dependency and is guaranteed to unlearn the deleted node features from the pre-trained model. Empirical results on real-world datasets illustrate the effectiveness, efficiency, and robustness of PROJECTOR. For example, when comparing to baseline unlearning methods, PROJECTOR enjoys a perfect data removal guarantee, and can achieve compatible performance compared to re-training the ordinary GNNs from scratch. [code] [Video]

1 Introduction

As graph representation learning has achieved great success in real-world applications (e.g., social networks [23, 18], knowledge graphs [37, 38], and recommendation system [2, 44]), privacy protection in graph representation learning has become equally important and has received much attention. Recently, as "*Right to be forgotten*" [40] gradually implemented in multiple jurisdictions, users are empowered with the right to request any organizations or companies to remove the effect of their private data from a machine learning model that previously trained on their private data. For example, when a Twitter user deletes his/her 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, which is known as machine unlearning [3], so as to prevent the private information in the deleted post be inferred by any malicious third party.

Existing machine unlearning approaches can be roughly classified into exact and approximate unlearning methods. The goal of exact unlearning [3, 10] is to produce the performance of the model 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 size is large, or even infeasible if not all the data are available to retrain. To reduce the computation overhead, approximate unlearning methods [15, 17] propose to approximate the model trained without the deleted data. However, these methods have limitations: ① these methods lack guarantee on whether all information associated with the deleted data is perfectly removed, ② the extension of these methods to graph-structured data due to inter-dependency of training data is

[1]Linear-GNN is referring the type of GNN structure that removes the non-linearities in ordinary GNNs and only has a single weight matrix in the neural network architecture, e.g., SGC [41] and APPNP [24].

Preprint. Under review.

non-trivial, and ③ these methods require using large regularization on weight parameters to stabilize the unlearning process, which could deteriorate the overall model performance.

Employing graph representation unlearning is even more challenging due to *node dependency*. For example, GRAPHERASER [10] proposes to partition the original graph into multiple subgraphs, then train an individual model on each subgraph and aggregate their prediction during inference. Upon receiving the data deletion requests, the model provider only needs to retrain the corresponding subgraph model. However, splitting too many subgraphs could hurt the model performance due to data heterogeneity and lack of training data for each individual model; on the other hand, splitting too few subgraphs could result in retraining on massive data, which could be potentially time-consuming. Moreover, graph partition could also result in the missing of node dependencies across the partitioned subgraphs, which further deteriorate the model performance [29]. A plausible direction to overcome the performance degradation caused by graph partitioning is to generalize the existing approximate unlearning approaches to graphs. For example, influence-based [17] or Fisher-based [15] second-order gradient unlearning approaches could be generalized to graph data by taking node dependencies into account. However, application of these methods to graph data suffers from a cubic computation complexity in terms of the neighborhood size of all deleted nodes within multi-hops (which grows exponentially as a function of GNN depth), require recording the computation graph of each node when using sampling strategies for large-scale training (which results in storage overhead and massive engineering efforts), and lack *perfect* data removal guarantees from an algorithmic standpoint.

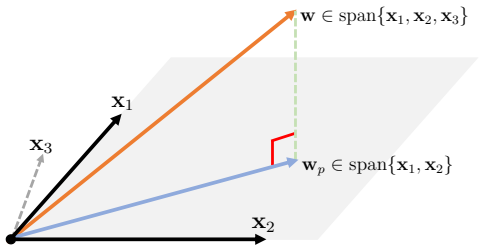

Figure 1: An illustration on the main idea of PROJECTOR. The original weight $\mathbf{w}$ exists inside the subspace defined by node feature vectors $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$. We can unlearn $\mathbf{x}_3$ and obtain the new weight $\mathbf{w}_p$ by projecting $\mathbf{w}$ on to the subspace defined without $\mathbf{x}_3$.

Motivated by the importance and the challenges of graph representation unlearning, we propose an efficient graph representation unlearning algorithm PROJECTOR, which enjoys the following appealing advantages: ① **Lower complexity**: PROJECTOR is an one-shot unlearning approach that exhibits a computation complexity that grows linearly in terms the number of deleted nodes, which is relatively low in practice comparing to either retraining from scratch or applying approximate unlearning approaches [17, 15] to graph; ② **Data removal guarantee**: PROJECTOR enjoys a *perfect* data removal guarantee that can be shown from the algorithmic-level under mild assumptions on deleted data. We emphasize the importance of such an algorithmic-level data removal guarantee by empirically showing that existing approximate unlearning methods might fail to unlearn all information related to the deleted data in practice; ③ **Better flexibility**: PROJECTOR can be directly implemented with existing sampling strategies for large-scale training, which does not require graph partitioning or caching all the sampled computation graph of each node during training (which is necessary if applying [17, 15] to graph-structured data).

To be able to perfectly unlearn, in this paper we focus on linear GNNs by removing non-linearities and collapsing weight matrices between consecutive layers[2]. Then, according to our observation that *"the weight parameters of linear-GNN (trained by logistic regression) belong to the linear span of all input node features"*, we propose to unlearn any specific node features by projecting linear-GNN's weight parameters to a subspace that is irrelevant to the unlearned node features (Section 4.1 and Section 4.2). Please refer to Figure 1 for an illustration. Notice that since the graph convolutions in linear-GNN can be considered as a linear combination of node features, the node dependency issue could be bypassed by the projection step in the PROJECTOR. To understand the effectiveness of PROJECTOR, we theoretically upper bound the difference between "the weight parameters obtained by PROJECTOR" and "the weight parameters obtained by retraining from scratch" (Section 4.3). To further boost the performance, we propose an unlearning-friendly adaptive-diffusion graph convolution operator that could empower linear-GNN to aggregate features based on feature

[2]Most approximate unlearning methods [14, 17, 15, 42] require such linearity requirement. Their extension to non-linear models (e.g., as in [14]) requires first pre-training a deep neural network as a feature extractor for the linear model, and then only unlearning the linear model without updating the feature extractor.

similarity (Section 5). We empirically validate the effectiveness of PROJECTOR on real-world datasets (Section 6 and Appendix F). For example, on the OGB-Arxiv dataset, applying PROJECTOR on linear-GNN only requires less than 0.1 second to unlearn and achieves compatible empirical performance compared to re-training an ordinary non-linear GNN from scratch.

2 Related work

Exact unlearning. The most straightforward way of exact unlearning is to retrain the model from scratch, which is in general computationally demanding, except for some model-specific or deterministic problems such as SVM [6], K-means [13], and decision tree [5]. To reduce the computation cost for general gradient-based training problems, [3] 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 [1, 19]. GRAPHERASER [10] extends [3] 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. [9] further generalize [10] to the recommendation 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. [35] 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. [26, 35, 30] study the unlearning from the generalization theory perspective, 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, [17] proposes to unlearn by removing the influence of the deleted data on the model parameters by second-order gradient update, where the Hessian is computed on the remaining data and gradient is computed on the deleted data. [15] performs Fisher forgetting by taking a single step of Newton's method on the remaining training data. [14] generalize 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. [20] speeds up [17] by using the leave-one-out residuals for the linear model update, which reduces time complexity to linear in the dimension of the deleted data and is independent of the size of the dataset. If objective function is finite sum, [42] 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 [13] for K-means and [43] for logistic regression. Notice that due to the nature of approximate unlearning, these methods only approximately unlearn the sensitive information and lack of perfect data removal guarantee in practice [33].

3 Graph representation unlearning

Problem setup. We consider solving the semi-supervised binary node classification problem using the linear GNN structure as proposed in [41], which could be easily extended to multi-class classification by using either cross-entropy loss or under one-over-rest setting. More specifically, given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges, let suppose each node $v_i \in \mathcal{V}$ is associated with 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, respectively. Then, a L-layer linear GNN computes the node representation $\mathbf{H} = \mathbf{P}^L\mathbf{X}$ by applying L propagation matrices $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ 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 $y_i \in \{-1, +1\}$ and 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

$$\min_{\mathbf{w}} F(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{\mathcal{V}_{\text{train}}} \sum\nolimits_{v_i \in \mathcal{V}_{\text{train}}} f_i(\mathbf{w}),\ f_i(\mathbf{w}) = \log\left(1 + \exp(-y_i \mathbf{w}^\top \mathbf{h}_i)\right), \quad (1)$$

where $\mathbf{h}_i = [\mathbf{P}^L\mathbf{X}]_i$. 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 $\mathcal{V}_{\text{delete}}$ from the weight parameters.

Challenges in graph unlearning. The key challenge of graph representation unlearning is due to the dependency between nodes in a graph. For example in Figure 2, our goal is to unlearn the feature

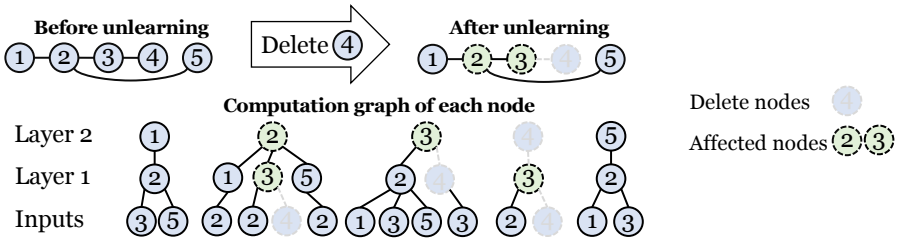

Figure 2: An illustration of how the computation graph of a 2-layer GNN (with neighbor average aggregation) is affected after deleting the node v_4 from graph.

of node v_4 from a pre-trained 2-layer GNN. 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. To generalize existing approximate unlearning approaches [17, 15] to graph, we not only need to remove the effect the deleted node representations $\mathbf{h}_4$ but also remove the effect of their multi-hop neighboring node representations $\mathbf{h}_2, \mathbf{h}_3$. Furthermore, when using the node sampling approach for a larger graph, we have to keep a record on the randomly sampled computation graph of each node representation at each iterations to know which node to unlearn, which could be storage prohibitive and requires many engineering efforts. GRAPHERASER [10] alleviates the exploding node dependency issue by partitioning the original graph into multiple subgraphs and ignoring the node dependency across multiple subgraphs. As a result, the node dependency is restricted to the subgraph it belongs to. However, splitting too many subgraphs could hurt the model performance due to data heterogeneity and lack of training data for each individual model; on the other hand, splitting too few subgraphs could result in retraining on massive data. Moreover, node dependencies across the partitioned subgraphs are missing after graph partition, which could further degenerate the model performance.

On the importance of data removal guarantee. Most approximate unlearning algorithms aim to generate the approximate unlearned model that is close to an exactly retrained model [42, 1, 20]. However, as pointed out by [33, 17], one cannot infer *"whether the data have been deleted"* solely from *"the closeness of the approximately unlearned and exactly retrained model in the parameter space"*. In fact, [33] 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 1 below, we provide theoretical justification for the empirical observation in [33] under the binary classification setting and the proof is deferred to Appendix A.

Theorem 1. *Consider a general binary classification problem using logistic regression* $\min_{\mathbf{w}\in\mathbb{R}^d} f_{LR}(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^N \log\left(1+\exp(-y_i\mathbf{w}^\top\mathbf{x}_i)\right)$. *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 [17, 15] 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 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.

4 PROJECTOR

In this section, we introduce PROJECTOR by first considering a general empirical risk minimization problem using logistic regression in Section 4.1, then introducing its application to graph representation unlearning in Section 4.2, and theoretically analyzing its effectiveness in Section 4.3.

4.1 Warm up: A projection-based unlearning approach

Before introducing PROJECTOR for graph representation unlearning, let us first consider logistic regression $F(\mathbf{w}) = \lambda\|\mathbf{w}\|_2^2/2 + \sum_{i\in\mathcal{V}}\log(1+\exp(-y_i\mathbf{w}^\top\mathbf{x}_i))$ to illustrate the key ideas. From

the optimization theory, the optimal $\mathbf{w}$ is a linear combination of training instances with coefficient $\boldsymbol{\alpha}$, i.e., $\mathbf{w} = \sum_{v_i \in \mathcal{V}} \alpha_i \mathbf{x}_i \in \mathbb{R}^d$. Suppose all features are independent[3] and we want to unlearn 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 $\mathbf{w}$ does not carry any information about $\mathbf{X}_{\text{delete}}$. Our main idea is to find an another linear combination by forcing $\alpha_i = 0$ for any $v_i \in \mathcal{V}_{\text{delete}}$. This can be achieved by finding an alternative solution $\mathbf{w}_p$ from a subspace irrelevant to $\mathbf{X}_{\text{delete}}$ that is close to $\mathbf{w}$, i.e., $\mathbf{w}_p = \arg\min_{\mathbf{v} \in \text{span}\{\mathbf{x}_i \mid v_i \in \mathcal{V}_{\text{remain}}\}} \|\mathbf{v} - \mathbf{w}\|_2^2$. More specifically, let $\mathcal{U}$ denote the linear subspace spanned by all remaining samples. To obtain the optimal solution, we can orthogonally project $\mathbf{w}$ onto the subspace $\mathcal{U}$. Knowing that any projection $\Pi_{\mathcal{U}}(\mathbf{w})$ onto $\mathcal{U}$ is necessarily an element of $\mathcal{U}$, which can be represented as linear combination of $\{\mathbf{x}_i \mid v_i \in \mathcal{V}_{\text{remain}}\}$ such that $\Pi_{\mathcal{U}}(\mathbf{w}) = \sum_{v_i \in \mathcal{V}_{\text{remain}}} \alpha_i \mathbf{x}_i$, where the coefficients of the orthogonal projection can be obtained by Proposition 1. The proof and detailed algorithm (Algorithm 1) are provided in Appendix B, and an illustration of the projection-based unlearning is shown in Figure 1.

Proposition 1. *Let $\mathbf{X}_{remain} = \{\mathbf{x}_j \mid v_j \in \mathcal{V}_{remain}\}$ denote the stack of all remaining features of size $r = |\mathcal{V}_{remain}|$ after deletion and $\boldsymbol{\alpha} = \{\alpha_j \mid v_j \in \mathcal{V}_{remain}\}$ as the vectorized coefficients, which could be computed by $\boldsymbol{\alpha} = \mathbf{X}_{remain}(\mathbf{X}_{remain}^\top \mathbf{X}_{remain})^\dagger \mathbf{w}$, where $\dagger$ denotes pseudo-inverse.*

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, respectively. However, if we suppose $\mathbf{X}^\top \mathbf{X}$ can be pre-computed before the unlearning request is made, under the circumstance that not all remaining node features are available during unlearning, one can still compute $\mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}}$ by $\mathbf{X}^\top \mathbf{X} - \mathbf{X}_{\text{delete}}^\top \mathbf{X}_{\text{delete}}$ and compute $(\mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^{-1}$ by applying the Woodbury identity [16] on $\mathbf{X}_{\text{delete}}^\top \mathbf{X}_{\text{delete}}$ and $\mathbf{X}^\top \mathbf{X}$, 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 $\mathbf{w}_p = \mathbf{X}_{\text{remain}}^\top \boldsymbol{\alpha}$. Besides, due to the similarity between logistic regression to SVM, PROJECTOR is also applicable to primal-based SVM unlearning to alleviate the high computation cost in the dual-based SVM unlearning approach [6]. Details are deferred to Appendix C.

4.2 PROJECTOR's application to graph data

In the following, we show in Proposition 2 that PROJECTOR is applicable to graph representation unlearning, because the graph convolution in the linear GNN is a linear operator on node features.

Proposition 2. *Suppose optimizing linear GNN (in Eq. 1) using mini-batch SGD with $\mathbf{w}(0) \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.[4] Then, after T-steps of gradient updates we have $\mathbf{w}(T) \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.*

To see this, let first recall the gradient of Eq. 1 with respect to $\mathbf{w}$, which is computed as

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

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

Based on this observation, we propose to remove the sensitive information of the deleted node features by projecting the weight onto a subspace that is spanned by the remaining node features $\mathbf{X}_{\text{remain}}$, where the subspace is irrelevant to the deleted node features $\mathbf{X}_{\text{delete}}$ as described in Section 4.1. One of the key advantages of applying PROJECTOR to graph representation unlearning is that it can bypass the node dependency issue and enjoys a relatively lower computation cost that is independent to the number of the GNN layers.

4.3 On the effectiveness of PROJECTOR

In this section, we study the effectiveness of PROJECTOR by measuring the ℓ_2-norm on the difference between $\mathbf{w}_p$ (PROJECTOR's output) to $\mathbf{w}_u$ (the solution obtained by retraining from scratch) on

[3] Features independent refers to any feature cannot be represented by linear combination of other node features. This can be guaranteed by adding small random noise to features that does not hurt performance.

[4] We denote $\mathbf{w}(t)$ as the weight after t steps of mini-batch SGD update. We say $\mathbf{w}(t)$ is in linear span of all node features $\mathbf{w}(t) \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ if there exists coefficients $\alpha_1, \ldots, \alpha_n$ such that $\mathbf{w}(t) = \sum_{i=1}^n \alpha_i \mathbf{x}_i$.

the dataset without the deleted nodes. Intuitively, we are expecting $\|\mathbf{w}_p - \mathbf{w}_u\|_2$ to be small. 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 $\mathcal{V}_{\text{remain}} = \mathcal{V}_{\text{train}} \setminus \mathcal{V}_{\text{delete}}$. Notice that 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 $\mathbf{P}_u$. Therefore, we can write down the objective after data deletion $F^u(\mathbf{w}_u)$ as

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

where $\mathbf{h}_i^u = [\mathbf{P}_u^L \mathbf{X}]_i$. Let $\mathbf{w}(t),\ \mathbf{w}_u(t)$ denote 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 $\mathbf{w}(t)$.

Before proceeding to our result, we make the following customery assumptions on graph propagation matrices, node features, and weight matrices in Assumption 1, on the correlation between node feature in Assumption 2, and on the variance of stochastic gradients in Assumption 3.

Assumption 1. *We assume each row of each propagation matrices before and after node deletion are 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 the 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 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 $\max_{v_j \in \mathcal{V}_{delete}} \min_{\boldsymbol{\gamma}} \left\|\sum_{i \in \mathcal{V}_{remain}} \gamma_i \mathbf{x}_i - \mathbf{x}_j\right\|_2 \leq \delta$.*

Assumption 3. *For any randomly selected deleted nodes set $\mathcal{V}_{delete}$, the variance of the stochastic gradient computed on the remaining nodes $\mathcal{V}_{remain} = \mathcal{V} \setminus \mathcal{V}_{delete}$ can be upper bounded by $G \geq 0$, i.e., $\mathbb{E}_{\mathcal{V}_{delete}}\left[\left\|\frac{1}{|\mathcal{V}_{train}|}\sum_{v_i \in \mathcal{V}_{train}} \nabla f_i^u(\mathbf{w}) - \frac{1}{|\mathcal{V}_{remain}|}\sum_{v_i \in \mathcal{V}_{remain}} \nabla f_i^u(\mathbf{w}_u)\right\|_2\right] \leq G$.*

In the following, we show that the difference between $\mathbf{w}_p$ (the output of PROJECTOR) and $\mathbf{w}_u(T)$ (retaining from scratch for T iterations) 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 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 2; ③ the variance of stochastic gradient computed on the remaining nodes, which is captured by G 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 where $\mathcal{V}_{\text{delete}} = \emptyset$, we have $P_d = \delta = G = 0$ and $\mathbf{w}_p = \mathbf{w}(T) = \mathbf{w}_u(T)$.

Theorem 2. *Let suppose Assumptions 1,2, and 3 hold and let us define $Q = \eta\big((1 + B_x B_w P_s) B_x P_d + G\big)$ where η is the learning rate used to pre-train the weight $\mathbf{w}(T)$ for T steps of gradient descent updates. Then, the closeness of $\mathbf{w}_p$ to the weight parameters $\mathbf{w}_u(T)$ can be bounded by*

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

After projection, we can finetune $\mathbf{w}_p$ for K iterations with learning rate $(\lambda + B_x^2 P_s^2)^{-1}$ to obtain $\tilde{\mathbf{w}}_p$ that has an error $F^u(\tilde{\mathbf{w}}_p) - \min_{\mathbf{w}} F^u(\mathbf{w}) \leq \mathcal{O}((\lambda + B_x^2 P_s^2)\Delta / K)$ if $\mathbf{w}_u(T)$ closes to optimal.

The proof of Theorem 2 is deferred to Appendix D. Besides, if δ satisfies the condition in Proposition 3, we know the solution of PROJECTOR is provably closer to the model retrained from scratch compared to [17, 15].

Proposition 3. *If the approximation error in Assumption 2 satisfying $\delta < \left((\lambda \eta T)^{-1} + 1\right) B_x \times \frac{|\mathcal{V}|}{|\mathcal{V}_{delete}|}$, then the solution of* PROJECTOR *is provably close to the solution obtained by retraining from scratch using influence-based [17] and Fisher-based [15] approximate unlearning methods.*

The proof of Proposition 3 is deferred to Appendix E. Moreover, we empirically validate the difference between the weight before and after unlearning in Section 6.2 to validate our theoretical results.

Table 1: Statistics of the datasets used in our experiments.

	OGB-Arxiv	OGB-Products	Cora	Pubmed
Nodes (Edges)	169,343 (1,166,243)	2,449,029 (61,859,140)	2,708 (10,556)	19,717 (88,648)

5 Toward a more powerful structure for linear-GNN

Diffusion equation: from image to graph. In image processing, diffusion has been interpreted as linear low-pass filtering for image demonising. However, diffusion can undesirably blurs neighborhood regions of different color and brightness. [27] propose an edge-preserving diffusion method by taking the difference of the pixels to be diffused into consideration, i.e., diffusion is strong if the difference is small (e.g., flat region or along edge) but weaker if the difference is large (e.g., across edges). Recently, the edge-preserving diffusion idea has been extended to graph domain aiming at alleviating the over-smoothing issue and build more powerful aggregation schema [7, 8, 32]. For instance, let suppose explicit Euler approximation is used for graph diffusion

$$\frac{\mathbf{H}^{(\ell+1)}-\mathbf{H}^{(\ell)}}{\gamma}=(\mathbf{D}_{\mathcal{G}}^{(\ell)}-\mathbf{I})\mathbf{H}^{(\ell)}\Rightarrow\mathbf{H}^{(\ell+1)}=\big((1-\gamma)\mathbf{I}+\gamma\mathbf{D}_{\mathcal{G}}^{(\ell)}\big)\mathbf{H}^{(\ell)}, \tag{5}$$

where $\mathbf{H}^{(\ell)}\in\mathbb{R}^{n\times d}$ denotes the node features after ℓ-steps of diffusion, γ is the discretization interval, and $\mathbf{D}_{\mathcal{G}}^{(\ell)}\in\mathbb{R}^{n\times n}$ is an algorithm dependent diffusion operator. For example when $\gamma=1$ and $\mathbf{D}_{\mathcal{G}}^{(\ell)}=\mathbf{D}^{-1}\mathbf{A}$, the solution is reduced to a linear GNN. GRAND [7, 32] initialize feature as $\mathbf{H}^{(0)}=\mathbf{X}$ and set diffusion operator $\mathbf{D}_{\mathcal{G}}^{(\ell)}$ as the multi-head self-attention [36] computed on $\mathbf{H}^{(\ell)}$. BLEND [8] utilize the same diffusion operator as GRAND but a different feature initialization, i.e., $\mathbf{H}^{(0)}=[\mathbf{X},\mathbf{U}]$ with $\mathbf{U}$ as a trainable positional encoding of each node. Although [7, 32, 8] are powerful, these methods are not ideal for graph unlearning because *perfectly remove the sensitive information from the weight parameters of multi-head self-attention module is non-trivial.*

An unlearning favorable adaptive diffusion operator. In contrast, we propose an unlearning favorable adaptive diffusion operation that take the similarity of both node feature and node label category information into consideration. Let us initialize features as $\mathbf{h}_i^{(0)}=\mathbf{x}_i$, initialize labels as $\mathbf{z}_i^{(0)}=\mathbf{y}_i$ if $i\notin\mathcal{V}_{\text{test}}$, and initialize $\mathbf{z}_i^{(0)}=\mathbf{0}$ if $i\in\mathcal{V}_{\text{test}}$. Then, the forward propagate is computed as $[\mathbf{H}^{(\ell+1)},\mathbf{Z}^{(\ell+1)}]=\big((1-\gamma)\mathbf{I}+\gamma\mathbf{D}_{\mathcal{G}}^{(\ell)}\big)[\mathbf{H}^{(\ell)},\mathbf{Z}^{(\ell)}]$, where $[\cdot,\cdot]$ is feature dimension concatenation operation and the i-th row j-th column of the ℓ-th diffusion operator is defined by

$$\Big[\mathbf{D}^{(\ell)}(\mathcal{G})\Big]_{i,j}=\frac{\omega_{ij}}{\sum_{k=1}^{N}\omega_{ik}},\ \omega_{ij}=\exp\left(-\frac{\|\mathbf{h}_i^{(\ell)}-\mathbf{h}_j^{(\ell)}\|_2^2}{\sigma_h^2}-\frac{\|\mathbf{z}_i^{(\ell)}-\mathbf{z}_j^{(\ell)}\|_2^2}{\sigma_z^2}\right), \tag{6}$$

and $\sigma_h,\sigma_z\in\mathbb{R}$ are parameters 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 $\mathbf{H}=[\mathbf{H}^{(L)},\mathbf{Z}^{(L)}]$ 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.

6 Experiments

In the following, we introduce experiment setup in Section 6.1 and report the experiment results in Section 6.2. The experiment configuration details and more results are provide in Appendix F.

6.1 Experiment setup

We choose OGB-Arxiv, OGB-Products, Cora, and Pubmed datasets to evaluate the performance of our model. The detailed dataset statistic is summarized in Table 1. We compare with graph representation unlearning approach GRAPHERASER [10], the application of influence-based unlearning approach INFLUENCE+ [17] and Fisher-based unlearning approach FISHER+ [15] to graph on the linear GNN. Besides, we compare with retraining GCN [23] and GraphSAGE [18] from scratch using various sampling-strategies.[5] For unlearning, we randomly select all deleted nodes from the training set and

[5]The code for INFLUENCE+ and FISHER+ are modified from certified-removal. Ordinary GNNs' implementations on OGB-Arxiv are modified from ogb-arxiv and on OGB-Productsis are modified from ogb-products.

Table 2: Comparison on the *F1-score accuracy* (Accuracy), and the norm of extra-feature weight channel (Weight norm) before unlearning and after unlearning (denoted as *before* $\rightarrow$ *after*), and wall-clock time (Time) using linear GNN. "-" stands for cannot generate meaningful results.

	Method	Metrics	Delete 2% nodes	Delete 5% nodes	Delete 10% nodes
OGB-Arxiv	PROJECTOR	Accuracy (%) Weight norm (Time)	73.39 $\rightarrow$ 73.32 19.4 $\rightarrow$ 0 (0.07 s)	73.33 $\rightarrow$ 73.39 21.7 $\rightarrow$ 0 (0.07 s)	73.25 $\rightarrow$ 73.39 56.8 $\rightarrow$ 0 (0.07 s)
	PROJECTOR (+ adapt diff)	Accuracy (%) Weight norm (Time)	73.44 $\rightarrow$ 73.52 21.0 $\rightarrow$ 0 (0.07 s)	73.42 $\rightarrow$ 73.48 24.3 $\rightarrow$ 0 (0.07 s)	73.34 $\rightarrow$ 73.44 25.6 $\rightarrow$ 0 (0.07 s)
	GRAPHERASER ($\times$8 subgraphs)	Accuracy (%) Weight norm (Time)	70.67 $\rightarrow$ 70.66 21.4 $\rightarrow$ 0 (1,866 $\times$ 8 s)	70.59 $\rightarrow$ 70.56 22.3 $\rightarrow$ 0 (1,866 $\times$ 8 s)	70.55 $\rightarrow$ 70.23 30.6 $\rightarrow$ 0 (1,866 $\times$ 8 s)
	INFLUENCE+	Accuracy (%) Weight norm (Time)	71.68 $\rightarrow$ 72.49 21.1 $\rightarrow$ 12.4 (1.1 s)	71.90 $\rightarrow$ 72.73 29.2 $\rightarrow$ 14.1 (1.1 s)	70.40 $\rightarrow$ 72.65 21.1 $\rightarrow$ 12.1 (1.1 s)
	FISHER+	Accuracy (%) Weight norm (Time)	72.44 $\rightarrow$ 72.49 21.1 $\rightarrow$ 12.4 (0.6 s)	72.29 $\rightarrow$ 72.73 29.2 $\rightarrow$ 14.1 (0.4 s)	71.71 $\rightarrow$ 72.65 35.4 $\rightarrow$ 15.6 (0.3 s)
OGB-Products	PROJECTOR	Accuracy (%) Weight norm (Time)	79.08 $\rightarrow$ 79.10 21.1 $\rightarrow$ 0 (0.06 s)	79.21 $\rightarrow$ 79.22 27.6 $\rightarrow$ 0 (0.06 s)	79.18 $\rightarrow$ 79.11 30.8 $\rightarrow$ 0 (0.06 s)
	PROJECTOR (+ adapt diff)	Accuracy (%) Weight norm (Time)	79.81 $\rightarrow$ 79.79 13.3 $\rightarrow$ 0 (0.06 s)	79.95 $\rightarrow$ 79.93 16.4 $\rightarrow$ 0 (0.06 s)	79.96 $\rightarrow$ 79.91 18.6 $\rightarrow$ 0 (0.06 s)
	GRAPHERASER ($\times$8 subgraphs)	Accuracy (%) Weight norm (Time)	70.79 $\rightarrow$ 70.78 21.4 $\rightarrow$ 0 (598 $\times$ 8 s)	70.80 $\rightarrow$ 70.78 25.4 $\rightarrow$ 0 (598 $\times$ 8 s)	70.80 $\rightarrow$ 70.78 28.9 $\rightarrow$ 0 (598 $\times$ 8 s)
	INFLUENCE+	Accuracy (%) Weight norm (Time)	72.28 $\rightarrow$ 72.58 4.4 $\rightarrow$ 1.8 (1.6 s)	72.23 $\rightarrow$ 72.78 8.9 $\rightarrow$ 3.1 (1.7 s)	72.08 $\rightarrow$ 72.51 14.3 $\rightarrow$ 4.2 (1.9 s)
	FISHER+	Accuracy (%) Weight norm (Time)	72.28 $\rightarrow$ 72.54 4.4 $\rightarrow$ 1.8 (1.6 s)	72.23 $\rightarrow$ 72.78 8.9 $\rightarrow$ 3.1 (1.3 s)	72.08 $\rightarrow$ 72.51 14.3 $\rightarrow$ 4.2 (1.1 s)
Cora	PROJECTOR	Accuracy (%) Weight norm (Time)	77.2 $\rightarrow$ 77.1 2.3 $\rightarrow$ 0 (0.13 s)	77.3 $\rightarrow$ 76.8 4.1 $\rightarrow$ 0 (0.13 s)	75.5 $\rightarrow$ 76.1 5.5 $\rightarrow$ 0 (0.13 s)
	INFLUENCE+	Accuracy (%) Weight norm (Time)	73.1 $\rightarrow$ 59.0 0.81 $\rightarrow$ 0.59 (0.2 s)	– –	– –
	FISHER+	Accuracy (%) Weight norm (Time)	73.1 $\rightarrow$ 73.6 0.81 $\rightarrow$ 0.08 (0.19 s)	72.0 $\rightarrow$ 72.7 0.72 $\rightarrow$ 0.08 (0.19 s)	72.0 $\rightarrow$ 72.4 0.78 $\rightarrow$ 0.18 (0.19 s)
Pubmed	PROJECTOR	Accuracy (%) Weight norm (Time)	86.1 $\rightarrow$ 86.1 9.1 $\rightarrow$ 0 (0.03 s)	86.3 $\rightarrow$ 86.3 10.1 $\rightarrow$ 0 (0.03 s)	86.7 $\rightarrow$ 86.7 11.1 $\rightarrow$ 0 (0.03 s)
	INFLUENCE+	Accuracy (%) Weight norm (Time)	80.8 $\rightarrow$ 82.4 4.6 $\rightarrow$ 4.7 (1.18 s)	80.9 $\rightarrow$ 82.8 5.9 $\rightarrow$ 10.0 (1.18 s)	80.6 $\rightarrow$ 82.2 7.0 $\rightarrow$ 29.1 (1.18 s)
	FISHER+	Accuracy (%) Weight norm (Time)	83.3 $\rightarrow$ 83.2 9.1 $\rightarrow$ 0.6 (1.17 s)	83.7 $\rightarrow$ 84.8 10.7 $\rightarrow$ 1.6 (1.16 s)	83.7 $\rightarrow$ 84.7 10.9 $\rightarrow$ 2.6 (1.15 s)

ask the model to unlearn. To evaluate the effectiveness of PROJECTOR, we first perform *"feature-label injection test"* to check whether PROJECTOR could perfectly unlearn the strong correlation between feature and label from the weight parameters. Then, we conduct *"comparison to retraining from scratch"* by comparing the closeness of PROJECTOR's output to retrained model. Finally, to understand the *"robustness of* PROJECTOR*"*, we compare the change of testing accuracy with respect to the number of deleted nodes against ordinary GNNs.

6.2 Experiment results

Feature-label injection test. To explicitly evaluate the effectiveness of unlearning, we first add an extra-label category to the dataset and all deleted nodes are changed to this extra-label category. Then, we append an extra binary feature to all nodes, set the extra binary feature as *one* for the deleted nodes and as *zero* for other nodes, and pre-train on the modified dataset. The effectiveness of unlearning method is evaluated by comparing the norm of weight parameters of the extra-feature channel before and after the unlearning process. For linear GNN, due to the strong correlation between the extra feature-label, we expect the weight norm of this extra-feature channel is large before unlearning. However, the value is expected to be zero after unlearning, because such correlation no longer exists in the dataset and will never be used if retraining from scratch. In this experiment, all deleted nodes are randomly selected as $2\%, 5\%, 10\%$ of the nodes from the training set after node deletion. As shown in Table 2, we have the following observations: ① By comparing the weight norm before and after unlearning, we know that PROJECTOR and GRAPHERASER can perfectly unlearn the correlation between the extra-feature and label by setting the extra-feature channel as zero, however, INFLUENCE+ and FISHER+ still have part of the information left in the weight; ② By comparing the wall-clock time, PROJECTOR has a lower computation cost and requires less time to unlearn, especially when comparing to GRAPHERASER; ③ By observing the accuracy of INFLUENCE+ and

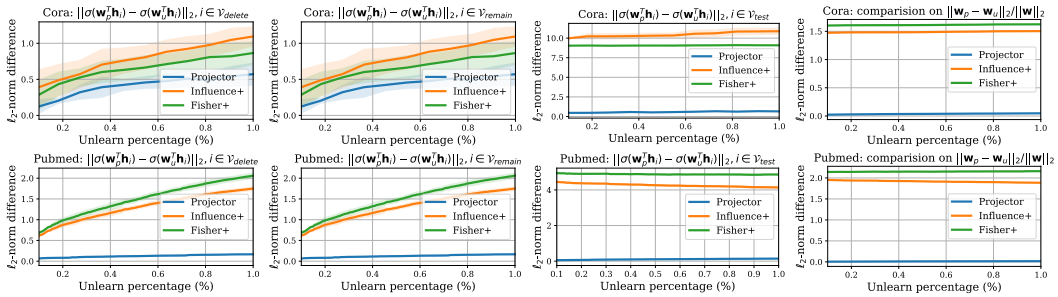

Figure 3: Comparison on the weight difference and model prediction (after the final layer activation function) before and after the unlearning process.

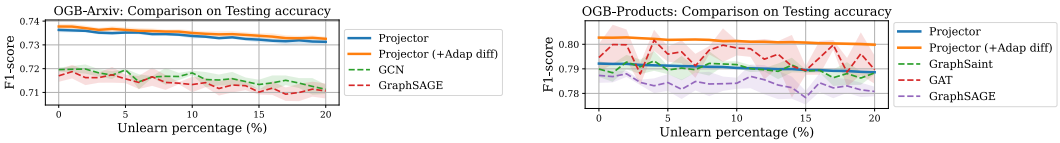

Figure 4: Comparison on the test performance with different number of node to unlearn.

FISHER+, these approaches suffers performance degradation because a stronger regularization is required to stabilize the unlearning process, which will dominates the loss; 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.

Comparison to retraining from scratch. Let $\mathbf{w}_p$ denote the solution obtained by unlearning and $\mathbf{w}_u$ denote the solution obtained by re-training from scratch. 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 $\mathcal{V}_{\text{remain}} = \mathcal{V}_{\text{train}} \setminus \mathcal{V}_{\text{delete}}$. 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 $\mathcal{B} \in \{\mathcal{V}_{\text{delete}}, \mathcal{V}_{\text{remain}}, \mathcal{V}_{\text{test}}\}$. Ideally, a powerful unlearning algorithm is expected to generate similar final weight parameters and activations to the retrained model. 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, when comparing to baseline methods. 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 result in Theorem 2.

Robustness of PROJECTOR. To evaluate the robustness of PROJECTOR, we study the change of testing accuracy as we progressively increase the unlearning ratio from 1% to 20%. We chose the test accuracy of re-training ordinary GNNs from scratch as the baselines. As shown in Figure 4, the change of testing accuracy in PROJECTOR is smaller (e.g., on OGB-Arvix dataset the test accuracy of PROJECTOR changes around 0.5% while the GNNs changes around $0.8\% \sim 1\%$) and more stable (i.e., the test accuracy fluctuate less when the fraction of unlearning nodes increases) than re-training ordinary GNNs. Besides, linear GNNs achieve compatible or even better performance than non-linear counterparts, which motivates us rethink the expressive power of linear GNN against ordinary GNNs.

7 Conclusion

In this paper, we study the problem of graph representation unlearning by proposing a projection-based unlearning approach PROJECTOR on the linear-GNN structure. PROJECTOR unlearn 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. Meanwhile, we note that PROJECTOR is only applicable to linear-GNN structure, which is the same case for most existing machine unlearning algorithms, e.g., [17, 15, 42]. One way to improve this is to first pre-train a MLP as a feature extractor for linear-GNN, then only unlearn the linear-GNN but with MLP feature extractor freezed. Please refer to Appendix G for more details. Since pre-training problem (e.g., pre-training dataset and strategy) for GNN is still less well explored, we leave this as an interesting future direction.

References

[1] Nasser Aldaghri, Hessam Mahdavifar, and Ahmad Beirami. Coded machine unlearning. *IEEE Access*, 2021.

[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. In *International Conference on Knowledge Discovery & Data Mining*, 2017.

[3] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[4] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *International Conference on Machine Learning*, 2021.

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

[7] Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pages 1407–1418. PMLR, 2021.

[8] Benjamin Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael Bronstein. Beltrami flow and neural diffusion on graphs. *Advances in Neural Information Processing Systems*, 34, 2021.

[9] Chong Chen, Fei Sun, Min Zhang, and Bolin Ding. Recommendation unlearning. *arXiv preprint arXiv:2201.06820*, 2022.

[10] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph unlearning. *arXiv preprint arXiv:2103.14991*, 2021.

[11] Christopher P Diehl and Gert Cauwenberghs. Svm incremental learning, adaptation and optimization. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 4, pages 2685–2690. IEEE, 2003.

[12] Honorius Gâlmeanu and Răzvan Andonie. Implementation issues of an incremental and decremental svm. In *International Conference on Artificial Neural Networks*, pages 325–335. Springer, 2008.

[13] Antonio Ginart, Melody Y Guan, Gregory Valiant, and James Zou. Making ai forget you: Data deletion in machine learning. *arXiv:1907.05012*, 2019.

[14] Aditya Golatkar, Alessandro Achille, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[15] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[16] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[17] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. In *International Conference on Machine Learning*, 2020.

[18] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

[19] Yingzhe He, Guozhu Meng, Kai Chen, Jinwen He, and Xingbo Hu. Deepobliviate: A powerful charm for erasing data residual memory in deep neural networks. *arXiv preprint arXiv:2105.06209*, 2021.

[20] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*, 2021.

[21] Masayuki Karasuyama and Ichiro Takeuchi. Multiple incremental decremental learning of support vector machines. *Advances in neural information processing systems*, 22, 2009.

[22] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[23] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[24] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[25] Pavel Laskov, Christian Gehl, Stefan Krüger, Klaus-Robert Müller, Kristin P Bennett, and Emilio Parrado-Hernández. Incremental support vector learning: Analysis, implementation and applications. *Journal of machine learning research*, 7(9), 2006.

[26] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. *arXiv preprint arXiv:2007.02923*, 2020.

[27] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

[28] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[29] Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Mahmut T Kandemir, and Anand Sivasubramaniam. Learn locally, correct globally: A distributed algorithm for training graph neural networks. *ICLR*, 2022.

[30] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember what you want to forget: Algorithms for machine unlearning. 2021.

[31] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[32] Matthew Thorpe, Tan Minh Nguyen, Hedi Xia, Thomas Strohmer, Andrea Bertozzi, Stanley Osher, and Bao Wang. Grand++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2021.

[33] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. *arXiv preprint arXiv:2110.11891*, 2021.

[34] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. 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, 2014.

[35] Enayat Ullah, Tung Mai, Anup Rao, Ryan Rossi, and Raman Arora. Machine unlearning via algorithmic stability. 2021.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[37] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *International Conference on Knowledge Discovery & Data Mining*, 2019.

[38] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. KGAT: knowledge graph attention network for recommendation. In *International Conference on Knowledge Discovery & Data Mining*, 2019.

[39] Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. Bag of tricks for node classification with graph neural networks. *arXiv preprint arXiv:2103.13355*, 2021.

[40] Wikipedia contributors. Right to be forgotten — Wikipedia, the free encyclopedia, 2021.

[41] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, 2019.

[42] Yinjun Wu, Edgar Dobriban, and Susan Davidson. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*, 2020.

[43] Yinjun Wu, Val Tannen, and Susan B Davidson. Priu: A provenance-based approach for incrementally updating regression models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.

[44] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *International Conference on Knowledge Discovery & Data Mining*, 2018.

[45] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Rajgopal Kannan, Viktor Prasanna, Long Jin, Andrey Malevich, and Ren Chen. Deep graph neural networks with shallow subgraph samplers. 2020.

Table of contents

Supplementary Material

A Proof of Theorem 1

The proof is pretty straightforward and 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:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \; f(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \log\Big(1+\exp(-y_i\mathbf{w}^\top\mathbf{x}_i)\Big). \tag{7}$$

By utilizing the following inequality

$$\log(1+\exp(-z)) \geq \big(-\alpha\log\alpha - (1-\alpha)\log(1-\alpha)\big) - \alpha z, \;\; \forall \alpha\in[0,1], \tag{8}$$

we can lower bound the objective in Eq. 7 by

$$\begin{aligned} f(\mathbf{w}) &= \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \log\Big(1+\exp(-y_i\mathbf{w}^\top\mathbf{x}_i)\Big) \\ &\geq \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \big(-\alpha_i\log\alpha_i - (1-\alpha_i)\log(1-\alpha_i) - \alpha_i y_i\mathbf{w}^\top\mathbf{x}_i\big) \\ &= L(\mathbf{w},\boldsymbol{\alpha}), \end{aligned} \tag{9}$$

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 [4], 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.,

$$\frac{\partial L(\mathbf{w}_\star,\boldsymbol{\alpha})}{\partial \mathbf{w}} = \lambda\mathbf{w} - \sum_{i=1}^{N}\alpha_i y_i\mathbf{x}_i = 0 \;\Rightarrow \mathbf{w}_\star = \frac{1}{\lambda}\sum_{i=1}^{N}\alpha_i y_i\mathbf{x}_i, \tag{10}$$

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

$$\alpha_i = 0 \;\vee\; y_i\mathbf{w}_\star^\top\mathbf{x}_i = -\log\alpha_i - \left(-1+\frac{1}{\alpha_i}\right)\log(1-\alpha_i) > 0. \tag{11}$$

From Eq. 11, 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 [6], therefore further updating the weight parameters is not required.

B 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:

$$\langle \mathbf{x}_i, \mathbf{w} - \Pi_{\mathcal{U}}(\mathbf{w})\rangle = 0, \;\; \forall i\in\mathcal{V}_{\text{remain}}. \tag{12}$$

Algorithm 1 PROJECTOR

Input: The pre-trained weight $\mathbf{w}$ of linear-GNN (or single-layer neural network) using logistic regression, randomly selected deleted nodes $\mathcal{V}_{\text{delete}}$ and remaining nodes $\mathcal{V}_{\text{remain}}$, node feature matrix $\mathbf{X}$, the pre-computed $\mathbf{M} = \mathbf{X}^\top\mathbf{X}$ and $\mathbf{M}^\dagger = (\mathbf{X}^\top\mathbf{X})^\dagger$.
Output: Unlearned weight parameters $\mathbf{w}_p$.

if $\mathbf{X}_{\text{remain}}$ is available and has enough computation resources for unlearning **then**
  Compute $\mathbf{M}_{\text{remain}} = \mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}}$.
  Compute $\mathbf{M}_{\text{remain}}^\dagger = (\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger$.
else if Only $\mathbf{X}_{\text{delete}}$ is available but $\mathbf{M}$, $\mathbf{M}^\dagger$ are pre-computed **then**
  Compute $\mathbf{M}_{\text{remain}}$ by

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

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

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

end if
Compute the projected weight parameters $\mathbf{w}_p = \mathbf{M}_{\text{remain}}\mathbf{M}_{\text{remain}}^\dagger\mathbf{w}$.

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. 12 can be formulated as $\mathbf{X}_{\text{remain}}(\mathbf{w} - \mathbf{X}_{\text{remain}}^\top\boldsymbol{\alpha}) = \mathbf{0}$. As a result, we have

$$\mathbf{X}_{\text{remain}}(\mathbf{w} - \mathbf{X}_{\text{remain}}^\top\boldsymbol{\alpha}) = \mathbf{0} \Rightarrow \boldsymbol{\alpha} = (\mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger\mathbf{X}_{\text{remain}}\mathbf{w}. \tag{13}$$

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. 13 from a different perspective by viewing it as the limit of the Ridge estimator with the Ridge parameter going to zero, i.e.,

$$(\mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger\mathbf{X}_{\text{remain}}\mathbf{w} = \lim_{\epsilon\to 0}(\epsilon\mathbf{I}_N + \mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger\mathbf{X}_{\text{remain}}\mathbf{w}. \tag{14}$$

Then, by using the Woodbury identity [16] we have

$$\begin{aligned}
&(\epsilon\mathbf{I}_N + \mathbf{X}_{\text{remain}}\mathbf{X}_{\text{remain}}^\top)^\dagger\mathbf{X}_{\text{remain}}\mathbf{w} \\
&= \Big(\frac{1}{\epsilon}\mathbf{I}_N - \mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger\mathbf{X}_{\text{remain}}^\top\Big)\mathbf{X}_{\text{remain}}\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}}\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger(\epsilon\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger(\mathbf{I}_N + \epsilon\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}} - \mathbf{I}_N)\mathbf{w} \\
&= \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} - \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}\mathbf{w} + \frac{1}{\epsilon}\mathbf{X}_{\text{remain}}(\mathbf{I}_N + \epsilon\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger\mathbf{w} \\
&= \mathbf{X}_{\text{remain}}(\epsilon\mathbf{I}_d + \mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger\mathbf{w}.
\end{aligned} \tag{15}$$

By taking the limit on both side, we have

$$\boldsymbol{\alpha} = \mathbf{X}_{\text{remain}}(\mathbf{X}_{\text{remain}}^\top\mathbf{X}_{\text{remain}})^\dagger\mathbf{w}, \tag{16}$$

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.

C Connection and potential application to SVM unlearning

The KKT-based unlearning has been studies in the SVM unlearning [6, 21, 12, 11] 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 generalization 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*$\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ *(e.g., Pegasos [31]) or using dual coordinate ascent (e.g.,SMO [28]), the (corresponding) primal solution after* t *iterations always satisfy* $\mathbf{w}(t) \in$ *span*$\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.

Therefore, we limit our following discussion in this section to linear SVM unlearning, which has the primal objective function defined as

$$f_{\text{SVM}}(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{N} \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i), \tag{17}$$

and dual objective function defined as

$$\begin{aligned} g_{\text{SVM}}(\boldsymbol{\alpha}) &= -\frac{1}{2\lambda}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^{N} \alpha_i, \\ \text{subject to } & \sum_{i=1}^{N} \alpha_i y_i = 0,\ \alpha_i \in [0, 1]. \end{aligned} \tag{18}$$

SVM unlearning [6] investigates how to maintain the KKT optimality condition when data are slightly changed. [6] proposes 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, [6] requires maintaining a $N \times N$ kernel matrix for enlarging or shrinking, which is memory consuming and engineering effort prohibitive when N is large [25]. Moreover, [6] 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 $\mathcal{M} = \{i \mid y_i \mathbf{w}_\star^\top \mathbf{x}_i = 1,\ 0 \le \alpha_i \le 1\}$,
- 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

$$\mathbf{w}_\star = \frac{1}{\lambda}\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \text{ and } \hat{y}_i = \mathbf{w}_\star^\top \mathbf{x}_i = \frac{1}{\lambda}\sum_{j \in \mathcal{M} \cup \mathcal{I}} \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle. \tag{19}$$

To delete data point $(\mathbf{x}_c, y_c)$, [6] 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

$$\Delta\alpha_c y_c \langle \mathbf{x}_i, \mathbf{x}_c \rangle + \sum\nolimits_{j \in \mathcal{M}} \Delta\alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle = 0, \tag{20}$$

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 [34].

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 [31]). We note that our primal SVM unlearning method shares the same spirit with [34], 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 [34].

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

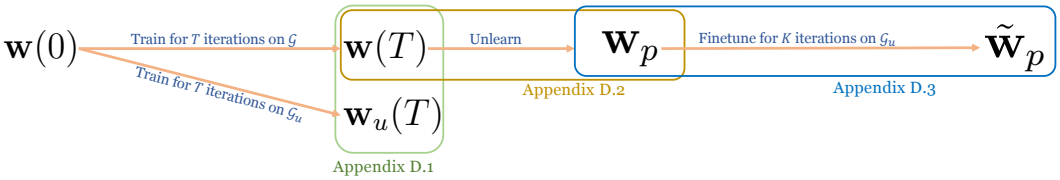

Figure 5: An illustration on the proof strategy of Theorem 2, where $\mathcal{G}$ stands for the graph used before node deletion and $\mathcal{G}_u$ stands for the graph after the node deletion.

D Proof of Theorem 2

To help reader better understand the proof strategy used in this section, we provide an overview on our proof strategy of Theorem 2. As shown in Figure 5, we derive the upper bound of $\|\mathbf{w}_u(T) - \mathbf{w}_p\|_2$ by first expanding the formula into two terms

$$\|\mathbf{w}_u(T) - \mathbf{w}_p\|_2 \leq \|\mathbf{w}_u(T) - \mathbf{w}(T)\|_2 + \|\mathbf{w}(T) - \mathbf{w}_p\|_2. \tag{21}$$

Then, we derive the upper bound of the first term in Appendix D.1 and the second term in Appendix D.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 D.3.

D.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

$$\begin{aligned}
\nabla F(\mathbf{w}) &= \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{i \in \mathcal{V}_{\text{train}}} \nabla f_i(\mathbf{w}),\ \nabla f_i(\mathbf{w}) = -y_i \sigma(-y_i \mathbf{w}^\top \mathbf{h}_i)\mathbf{h}_i + \lambda \mathbf{w}, \\
\nabla F^u(\mathbf{w}) &= \frac{1}{|\mathcal{V}_{\text{remain}}|} \sum_{i \in \mathcal{V}_{\text{remain}}} \nabla f_i^u(\mathbf{w}),\ \nabla f_i^u(\mathbf{w}) = -y_i \sigma(-y_i \mathbf{w}^\top \mathbf{h}_i^u)\mathbf{h}_i^u + \lambda \mathbf{w},
\end{aligned} \tag{22}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the Sigmoid function. From Eq. 22, 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

$$\begin{aligned}
\|\nabla f_i(\mathbf{w}_1) - \nabla f_i(\mathbf{w}_2)\|_2 &= \left\|-y_i \sigma(-y_i \mathbf{w}_1^\top \mathbf{h}_i)\mathbf{h}_i + y_i \sigma(-y_i \mathbf{w}_2^\top \mathbf{h}_i)\mathbf{h}_i + \lambda(\mathbf{w}_1 - \mathbf{w}_2)\right\|_2 \\
&\leq \|y_i \mathbf{h}_i\|_2 \cdot |\sigma(-y_i \mathbf{w}_1^\top \mathbf{h}_i) + \sigma(-y_i \mathbf{w}_2^\top \mathbf{h}_i)| + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq \|y_i \mathbf{h}_i\|_2 \cdot \|y_i \mathbf{w}_1^\top \mathbf{h}_i - y_i \mathbf{w}_2^\top \mathbf{h}_i\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq \|y_i \mathbf{h}_i\|_2^2 \cdot \|\mathbf{w}_1 - \mathbf{w}_2\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq \left(\max_i \left\|[\mathbf{P}^L \mathbf{X}]_i\right\|_2^2\right) \|\mathbf{w}_1 - \mathbf{w}_2\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq (\lambda + P_s^2 B_x^2) \|\mathbf{w}_1 - \mathbf{w}_2\|_2, \\
\|\nabla f_i^u(\mathbf{w}_1) - \nabla f_i^u(\mathbf{w}_2)\|_2 &\leq \left(\max_i \left\|[\mathbf{P}_u^L \mathbf{X}]_i\right\|_2^2\right) \|\mathbf{w}_1 - \mathbf{w}_2\|_2 + \lambda \|\mathbf{w}_1 - \mathbf{w}_2\|_2 \\
&\leq (\lambda + P_s^2 B_x^2) \|\mathbf{w}_1 - \mathbf{w}_2\|_2.
\end{aligned} \tag{23}$$

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

$$
\begin{aligned}
&\mathbb{E}\left[\|\nabla F(\mathbf{w}) - \nabla F^u(\mathbf{w})\|_2\right] \\
&= \mathbb{E}\left[\left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i(\mathbf{w}) - \frac{1}{|\mathcal{V}_{\text{remain}}|}\sum_{i\in\mathcal{V}_{\text{remain}}}\nabla f_i^u(\mathbf{w}_u)\right\|_2\right] \\
&\underset{(a)}{\leq} \left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i(\mathbf{w}) - \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i^u(\mathbf{w})\right\|_2 \\
&\quad + \mathbb{E}\left[\left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\nabla f_i^u(\mathbf{w}) - \frac{1}{|\mathcal{V}_{\text{remain}}|}\sum_{i\in\mathcal{V}_{\text{remain}}}\nabla f_i^u(\mathbf{w}_u)\right\|_2\right],
\end{aligned}
\tag{24}
$$

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

$$
\begin{aligned}
&\left\|\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i - \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i^u)\mathbf{h}_i^u\right\|_2 \\
&\underset{(a)}{\leq} \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\left\|y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i - y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i^u\right\|_2 + \\
&\quad + \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\left\|y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\mathbf{h}_i^u - y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i^u)\mathbf{h}_i^u\right\|_2 \\
&\leq \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\left|y_i\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i)\right|\|\mathbf{h}_i - \mathbf{h}_i^u\|_2 + \\
&\quad + \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}\|y_i\mathbf{h}_i^u\|_2\left|\sigma(-y_i\mathbf{w}^\top\mathbf{h}_i) - \sigma(-y_i\mathbf{w}^\top\mathbf{h}_i^u)\right| \\
&\underset{(b)}{\leq} \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}(\|\mathbf{h}_i - \mathbf{h}_i^u\|_2 + \|\mathbf{h}_i^u\|_2\|\mathbf{w}\|_2\|\mathbf{h}_i - \mathbf{h}_i^u\|_2) \\
&= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}}(1 + \|\mathbf{h}_i^u\|_2\|\mathbf{w}\|_2)\|\mathbf{h}_i - \mathbf{h}_i^u\|_2,
\end{aligned}
\tag{25}
$$

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

$$
\begin{aligned}
\|\mathbf{h}_i - \mathbf{h}_i^u\|_2 &= \max_j \|[\mathbf{P}^L\mathbf{X} - \mathbf{P}_u^L\mathbf{X}]_j\|_2 \\
&\leq \max_j \|\mathbf{x}_j\|_2 \cdot \max_j \left\|[\mathbf{P}^L - \mathbf{P}_u^L]_j\right\|_2 \\
&\leq B_x P_d, \\
\|\mathbf{h}_i^u\|_2 &= \max_j \|[\mathbf{P}_u^L\mathbf{X}]_j\|_2 \\
&\leq \max_j \|\mathbf{x}_j\|_2 \cdot \max_j \left\|[\mathbf{P}_u^L]_j\right\|_2 \\
&\leq B_x P_s.
\end{aligned}
\tag{26}
$$

Besides, the upper bound of the second term on the right hand side of inequality (a) is from Assumption 3. By plugging the results back to the right hand side of inequality (a), we have

$$
\|\nabla F(\mathbf{w}) - \nabla F^u(\mathbf{w})\|_2 \leq (1 + B_x B_w P_s) B_x P_d + G. \tag{27}
$$

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

$$\begin{aligned}
&\|\mathbf{w}_u(t+1) - \mathbf{w}(t+1)\|_2 \\
&\quad = \|\mathbf{w}_u(t) - \eta\nabla F^u(\mathbf{w}_u(t)) - \mathbf{w}(t) + \eta\nabla F(\mathbf{w}(t))\|_2 \\
&\quad \leq \|\mathbf{w}_u^t - \mathbf{w}^t\|_2 + \eta\|\nabla F^u(\mathbf{w}_u(t)) - \nabla F(\mathbf{w}_u(t)) + \nabla F(\mathbf{w}_u(t)) - \nabla F(\mathbf{w}(t))\|_2 \\
&\quad \leq (1 + \eta(\lambda + B_x^2 B_s^2))\|\mathbf{w}_u^t - \mathbf{w}^t\|_2 + \eta\|\nabla F^u(\mathbf{w}_u(t)) - \nabla F(\mathbf{w}_u(t))\|_2 \\
&\quad \underset{(a)}{\leq} (1 + \eta(\lambda + B_x^2 B_s^2))\|\mathbf{w}_u^t - \mathbf{w}^t\|_2 + \eta\Big((1 + B_x B_w P_s)B_x P_d + G\Big),
\end{aligned} \tag{28}$$

where (a) is due to Eq. 23 and Eq. 27. Then after T iterations, we can bound the different between two parameters as

$$\|\mathbf{w}_u(T) - \mathbf{w}(T)\|_2 \leq \eta\Big((1 + B_x B_w P_s)B_x P_d + G\Big)\sum_{t=1}^{T}(1 + \eta(\lambda + B_x^2 B_s^2))^{t-1}. \tag{29}$$

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

From Proposition 2 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. 22, we have

$$\begin{aligned}
\nabla F(\mathbf{w}(t)) &= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)\mathbf{h}_i + \lambda\mathbf{w}(t) \\
&= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)[\mathbf{P}^L\mathbf{X}]_i + \lambda\mathbf{w}(t) \\
&= \frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)\left(\sum_{j\in\mathcal{V}}[\mathbf{P}^L]_{ij}\mathbf{x}_j\right) + \lambda\mathbf{w}(t) \\
&= \sum_{j\in\mathcal{V}}\left(\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)[\mathbf{P}^L]_{ij}\right)\mathbf{x}_j + \lambda\mathbf{w}(t).
\end{aligned} \tag{30}$$

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

$$\begin{aligned}
\mathbf{w}(t+1) &= (1-\eta\lambda)\times\mathbf{w}(t) - \eta\sum_{j\in\mathcal{V}}\left(\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(t)\mathbf{h}_i)[\mathbf{P}^L]_{ij}\right)\mathbf{x}_j \\
&= \sum_{k=0}^{t}\eta(1-\eta\lambda)^{t-k}\left[\sum_{j\in\mathcal{V}}\left(\frac{1}{|\mathcal{V}_{\text{train}}|}\sum_{i\in\mathcal{V}_{\text{train}}} -y_i\sigma(-y_i\mathbf{w}^\top(k)\mathbf{h}_i)[\mathbf{P}^L]_{ij}\right)\mathbf{x}_j\right]
\end{aligned} \tag{31}$$

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

$$\mathbf{w}(T) = \sum_{j\in\mathcal{V}}\beta_j\mathbf{x}_j, \text{ where } \beta_j \leq \eta T \cdot \max_{i\in\mathcal{V}_{\text{train}}}[\mathbf{P}^L]_{ij} \leq \eta T, \tag{32}$$

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

$$\begin{aligned}
\|\mathbf{w}_p - \mathbf{w}(T)\|_2 &= \min_{\boldsymbol{\alpha}} \left\| \sum_{i\in\mathcal{V}_{\text{remain}}} \alpha_i \mathbf{x}_i - \sum_{i\in\mathcal{V}} \beta_i \mathbf{x}_i \right\|_2 \\
&= \min_{\boldsymbol{\alpha}} \left\| \sum_{i\in\mathcal{V}_{\text{remain}}} (\alpha_i - \beta_i)\mathbf{x}_i - \sum_{i\in\mathcal{V}_{\text{delete}}} \beta_i \mathbf{x}_i \right\|_2 \\
&\leq \sum_{i\in\mathcal{V}_{\text{delete}}} \beta_j \cdot \min_{\boldsymbol{\alpha}} \left\| \sum_{i\in\mathcal{V}_{\text{remain}}} \frac{\alpha_i - \beta_i}{\beta_j} \mathbf{x}_i - \mathbf{x}_j \right\|_2 \\
&\leq |\mathcal{V}_{\text{delete}}| \cdot \max_{j\in\mathcal{V}_{\text{delete}}} \beta_j \cdot \min_{\boldsymbol{\alpha}} \left\| \sum_{i\in\mathcal{V}_{\text{remain}}} \frac{\alpha_i - \beta_i}{\beta_j} \mathbf{x}_i - \mathbf{x}_j \right\|_2 \\
&\leq \eta T |\mathcal{V}_{\text{delete}}| \cdot \underbrace{\max_{j\in\mathcal{V}_{\text{delete}}} \min_{\boldsymbol{\alpha}} \left\| \sum_{i\in\mathcal{V}_{\text{remain}}} \frac{\alpha_i - \beta_i}{\beta_j} \mathbf{x}_i - \mathbf{x}_j \right\|_2}_{(a)}.
\end{aligned} \tag{33}$$

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

$$\max_{j\in\mathcal{V}_{\text{delete}}} \min_{\boldsymbol{\gamma}} \left\| \sum_{i\in\mathcal{V}_{\text{remain}}} \gamma_i \mathbf{x}_i - \mathbf{x}_j \right\|_2 \leq \delta, \tag{34}$$

where the upper bound is due to Assumption 2. Therefore, we have

$$\|\mathbf{w}_p - \mathbf{w}(T)\|_2 \leq \delta\eta T |\mathcal{V}_{\text{delete}}|. \tag{35}$$

D.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 2.. By knowing F^u is $(\lambda + B_x^2 P_s^2)$-smoothness, we have

$$\begin{aligned}
&F^u(\mathbf{w}_p(k+1)) \\
&\underset{(a)}{\leq} F^u(\mathbf{w}_p(k)) + \langle \nabla F^u(\mathbf{w}_p(k)), \mathbf{w}_p(k+1) - \mathbf{w}_p(k) \rangle + \frac{(\lambda + B_x^2 P_s^2)}{2} \|\mathbf{w}_p(k+1) - \mathbf{w}_p(k)\|_2^2 \\
&= F^u(\mathbf{w}_p(k)) - \eta \|\nabla F^u(\mathbf{w}_p(k))\|_2^2 + \frac{\eta^2(\lambda + B_x^2 P_s^2)}{2} \|\nabla F^u(\mathbf{w}_p(k))\|_2^2 \\
&= F^u(\mathbf{w}_p(k)) - \eta \left(1 - \frac{\eta_k(\lambda + B_x^2 P_s^2)}{2}\right) \|\nabla F^u(\mathbf{w}_p(k))\|_2^2,
\end{aligned} \tag{36}$$

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 $\eta = \frac{2}{(\lambda + B_x^2 P_s^2)}$, we have

$$\Big(F^u(\mathbf{w}_p(k+1)) - F^u(\mathbf{w}_\star)\Big) - \Big(F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star)\Big) \leq -\frac{1}{2(\lambda + B_x^2 P_s^2)} \|\nabla F^u(\mathbf{w}_p(k))\|_2^2. \tag{37}$$

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

$$\begin{aligned}
F^u(\mathbf{w}_p(k+1)) - F^u(\mathbf{w}_\star) &\leq \langle \nabla F^u(\mathbf{w}_p(k)), \mathbf{w}_p(k) - \mathbf{w}_\star \rangle \\
&\leq \|F^u(\mathbf{w}_p(k))\|_2 \|\mathbf{w}_p(k) - \mathbf{w}_\star\|_2.
\end{aligned} \tag{38}$$

By plugging it back to Eq. 37, we have

$$\begin{aligned} & F^u(\mathbf{w}_p(k+1)) - F^u(\mathbf{w}_\star) \\ & \leq (F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star)) \left(1 - \frac{F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star)}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p(k) - \mathbf{w}_\star\|_2^2}\right) \\ & \underset{(a)}{\leq} (F^u(\mathbf{w}_p) - F^u(\mathbf{w}_\star)) \left(1 - \frac{F^u(\mathbf{w}_p(T)) - F^u(\mathbf{w}_\star)}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}\right), \end{aligned} \tag{39}$$

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

$$\frac{1}{F^u(\mathbf{w}_p(k+1)) - F^u(\mathbf{w}_\star)} \geq \frac{1}{F^u(\mathbf{w}_p(k)) - F^u(\mathbf{w}_\star)} + \frac{1}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}. \tag{40}$$

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

$$\frac{1}{F^u(\tilde{\mathbf{w}}_p) - F^u(\mathbf{w}_\star)} \geq \frac{1}{F^u(\mathbf{w}_p) - F^u(\mathbf{w}_\star)} + \frac{T}{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}, \tag{41}$$

which implies

$$\frac{1}{F^u(\tilde{\mathbf{w}}_p) - F^u(\mathbf{w}_\star)} \leq \frac{2(\lambda + B_x^2 P_s^2)(F^u(\mathbf{w}_p - F^u(\mathbf{w}_\star))\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}{T(F^u(\mathbf{w}_p - F^u(\mathbf{w}_\star)) + 2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}. \tag{42}$$

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

$$F^u(\mathbf{w}_p) - F^u(\mathbf{w}_\star) \leq \frac{(\lambda + B_x^2 P_s^2)}{2}\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2. \tag{43}$$

Plugging back to Eq. 42, suppose T is large enough and $\mathbf{w}_u(T) \approx \mathbf{w}_\star$, we have

$$\begin{aligned} F^u(\tilde{\mathbf{w}}_p) - F^u(\mathbf{w}_\star) &\leq \frac{2(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_\star\|_2^2}{K+4} \\ &\approx \mathcal{O}\left(\frac{(\lambda + B_x^2 P_s^2)\|\mathbf{w}_p - \mathbf{w}_u(T)\|_2^2}{K}\right), \end{aligned} \tag{44}$$

E Proof on Proposition 3

The influence-based unlearning approach [17] unlearn by using second-order gradient update on the weight parameters. To apply [17] 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 [17] to graph requires updating the weight parameters by

$$\mathbf{w}_p = \mathbf{w}(T) - \left[\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{remain}})\right]^{-1} \nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}}), \tag{45}$$

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

To prove Proposition 3, we need to first analyze the upper bound on $\|\mathbf{w}_p - \mathbf{w}(T)\|_2$ for INFLUENCE [17] by

$$\begin{aligned} \|\mathbf{w}_p - \mathbf{w}(T)\|_2 &= \left\| \left[\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{remain}})\right]^{-1} \nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}}) \right\|_2 \\ &\leq \left\| \left[\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{remain}})\right]^{-1} \right\|_2 \left\| \nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}}) \right\|_2. \end{aligned} \tag{46}$$

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

$$\begin{aligned}
&\left\|\nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2 \\
&= \left\| \sum_{j\in\mathcal{V}} \left(\frac{1}{|\mathcal{V}_{\text{delete}}|} \sum_{i\in\mathcal{V}_{\text{delete}}} -y_i \sigma(-y_i \mathbf{w}^\top(T)\mathbf{h}_i)[\mathbf{P}^L]_{ij} \right) \mathbf{x}_j + \lambda \mathbf{w}(T) \right\|_2 \\
&\leq \left\| \sum_{j\in\mathcal{V}} \left(\frac{1}{|\mathcal{V}_{\text{delete}}|} \sum_{i\in\mathcal{V}_{\text{delete}}} -y_i \sigma(-y_i \mathbf{w}^\top(T)\mathbf{h}_i)[\mathbf{P}^L]_{ij} \right) \mathbf{x}_j \right\|_2 + \lambda \|\mathbf{w}(T)\|_2 \\
&\underset{(a)}{\leq} B_x|\mathcal{V}| + \lambda\|\mathbf{w}(T)\|_2,
\end{aligned} \tag{47}$$

where (a) is due to $|y_i| = 1,\ 0 < \sigma(x) < 1$, and $[\mathbf{P}^L]_{ij} \leq 1$. Meanwhile, from Eq. 31, we can upper bound $\|\mathbf{w}(T)\|_2$ by

$$\begin{aligned}
\|\mathbf{w}(T)\|_2 &= \left\| \sum_{k=0}^{T-1} \eta(1-\eta\lambda)^{T-1-k} \left[\sum_{j\in\mathcal{V}} \left(\frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{i\in\mathcal{V}_{\text{train}}} -y_i \sigma(-y_i \mathbf{w}^\top(k)\mathbf{h}_i)[\mathbf{P}^L]_{ij} \right) \mathbf{x}_j \right] \right\|_2 \\
&\leq \eta T \times B_x|\mathcal{V}|
\end{aligned} \tag{48}$$

By plugging the result back, we have

$$\left\|\nabla F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2 \leq (1+\lambda\eta T)B_x|\mathcal{V}|. \tag{49}$$

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

$$\left(\left\|\nabla^2 F(\mathbf{w}(T),\ \mathcal{V}^L_{\text{delete}})\right\|_2\right)^{-1} \leq \frac{1}{\lambda} \tag{50}$$

Therefore, we know that

$$\|\mathbf{w}_p - \mathbf{w}(T)\|_2 \leq \frac{(1+\lambda\eta T)B_x|\mathcal{V}|}{\lambda}. \tag{51}$$

By comparing Eq. 35 and Eq. 51, we know that if

$$\delta < \left(\frac{1}{\lambda\eta T} + 1\right) B_x \times \frac{|\mathcal{V}|}{|\mathcal{V}_{\text{delete}}|} \tag{52}$$

the solution of PROJECTOR if provable closer to the retraining from scratch than [17]. Moreover, the above discussion also holds for [15] by replacing the variable $\mathcal{V}_{\text{delete}}$ in Eq. 45, 46, and 47 as $\mathcal{V}_{\text{remain}}$.

F Additional details and results on experiments

F.1 Details on experiment setups

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 2 due to space limit).

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 [45] 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 [39] are used that leverage the training set node label information for inference.

For PROJECTOR, 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}.

For INFLUENCE- and FISHER-based unlearning, since these methods only support data unlearning and do not support data addition, we opt to unlearn the neighboring of all deleted nodes within multi-hops, which grows exponentially to GNN depth due to the node dependency issue. Besides, we

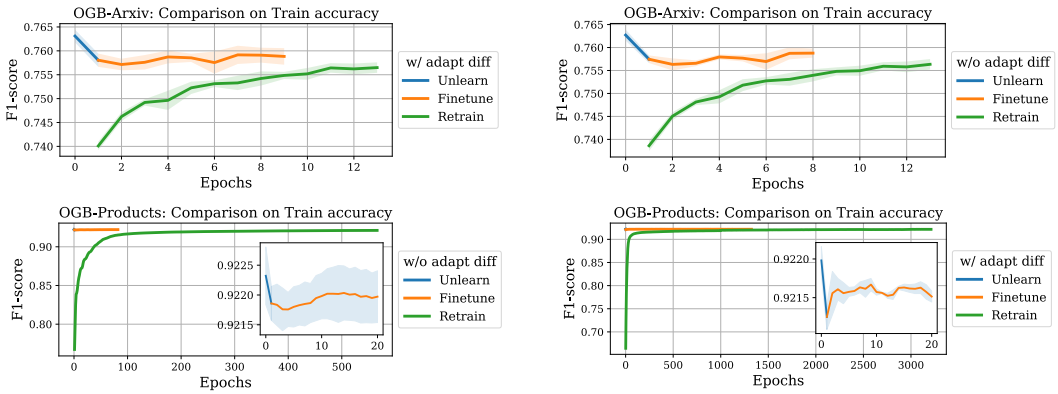

Figure 6: Evaluation on effect of finetuning on the training accuracy before and after 1% of the node from training set deleted.

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.

For GRAPHERASER, we split all nodes into 8 shards using graph partition algorithm Metis [22] 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. By assuming all shard models can re-train in parallel, we only report the wall-clock time for re-training a single shard model.

For ordinary GNNs, we take their code from the Open Graph Benchmark's online public implementation and use the same hyper-parameters as originally provided.

F.2 More experiment results

F.2.1 Performance before and after finetuning

The experiment results in Section 6.2 are reported without the fine-tuning process as mentioned in Theorem 2. 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 epochs on the OGB-Arvix dataset and $1,000$ epochs on the OGB-Products dataset. We repeat the experiment 5 times. Other setup remains the same as introduced in Section F.1.

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 3 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.

Table 3: 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

F.2.2 Evaluation on the δ term in Assumption 2

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 2. Therefore, we report the δ by computing

$$\delta = \max_{v_i \in \mathcal{V}} \left\| \mathbf{x}_i - \mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}} (\mathbf{X}_{\text{remain}}^\top \mathbf{X}_{\text{remain}})^\dagger \mathbf{x}_i \right\|_2 , \tag{53}$$

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 4, 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 4: 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

G An unlearning favorable extension from linear- to non-linear GNN

The geometric view of solving the logistic regression (as defined in Eq. 1) is finding a hyperplane to linearly separate the node representation $\mathbf{H}$ computed by linear GNN. However, node representations might not be linearly separable. To overcome this issue, one could first apply an *non-linear MLP* on each node features, then apply graph convolutions onto the output of the MLP before classification. This can be interpreted as finding a non-linear separation in the input space. We note that our extension to non-linear is different from most non-linear GNN structure where an activation function and weight parameter matrix is used at each layer. For example, if using a MLP with two weight matrices, the node representation is computed by the first row of the following equation, which is different from ordinary GNNs in the second row:

$$\begin{aligned} \mathbf{H} &= \mathbf{P}^L \sigma(\sigma(\mathbf{X}\mathbf{W}_{\text{mlp}}^{(1)})\mathbf{W}_{\text{mlp}}^{(2)})\mathbf{W}_{\text{gnn}}^{(3)} && \text{Our proposal} \\ \mathbf{H} &= \mathbf{P}\sigma(\mathbf{P}\sigma(\mathbf{P}\mathbf{X}\mathbf{W}_{\text{gnn}}^{(1)})\mathbf{W}_{\text{gnn}}^{(2)})\mathbf{W}_{\text{gnn}}^{(3)} && \text{Ordinary non-linear GNN} \end{aligned} \tag{54}$$

In our case, although the *input* of MLP is relatively non-linear to each other, the *output* of MLP is still relatively linear to each other, which allows us to utilize PROJECTOR if MLP model is known not carrying the deleted node feature information.

For example, we can first pre-train the non-linear GNN on a public dataset with training samples that do not need to be forgotten, then we only need to take care of the data removal from the linear model that is applied to 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

Table 5: Comparison on the performance of linear GNN and its non-linear extension with ordinary GNNs.

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

unlearning. For completeness, we conduct some preliminary experiments on the OGB-Arxiv and OGB-Product datasets by comparing the performance of non-linear GNNs to linear GNNs, where the MLP extractor is pre-trained on all node features except the deleted ones by supervised learning. As shown in Table 5, we have the following observations: ① Although using MLP as a feature extractor can slightly improve the average F1-score accuracy, it also increases the variance of the model performance. ② Linear GNN could achieve better performance than ordinary GNNs by carefully tuning the hyper-parameters and using the label reuse tricks [39]. ③ 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. Such an observation is aligned with the result originally reported in [10] for mean-average prediction aggregation.

H Some discussion on privacy and potential concerns

In this section, we give some discussion on the current stage of machine unlearning. This section can be think of is as an extension of the conclusion section (Section 7). We summarize the topic of each paragraph in boldface at the beginning of each paragraph.

Different privacy measurement. Most of the existing machine unlearning literature are heavily relying on the definition of differential privacy. For example, one of the most widely accepted notation is *certified removal* [17], which is defined as $\exp(-\epsilon) \leq P(\mathbf{w}_p)/P(\mathbf{w}_u) \leq \exp(\epsilon)$, where $P(\mathbf{w}_p)$ is the prediction on the unlearned solution $\mathbf{w}_p$, $P(\mathbf{w}_u)$ is the prediction on the re-training from scratch solution $\mathbf{w}_u$, and ϵ is any arbitrary small positive real number. Please notice that the certified notation shares the same spirit to our result in Theorem 2 because both the model considered in ours and [17] are linear and convex. However, since [17] is an approximate unlearning strategy by its nature, one cannot know whether the *influence* of the deleted data is perfectly removed, therefore adding random noises to the weight parameters is required during the unlearning process.

Different *influence* definition. In fact, the definition of *influence* is still not well defined and it is different from one literature to another. Although different notations on *influence* could be ambiguous, we believe each of them has its own value. For example, [17] measures the *influence* as the norm of a second-order gradient and they reduce such *influence* by using single-step of second-order gradient update. In this paper, we consider *influence* as whether the learned weight parameters actually contain the deleted data. Therefore, we propose an unlearning strategy to project the original solution onto a subspace that is irrelevant to the deleted node features.

An alternative view on our *influence* definition. Let's suppose the original linear GNN model is trained by uniform mini-batch neighbor sampling. Since our key idea is to find a new solution from a subspace that does not contain the deleted data, one can think of PROJECTOR as using importance

mini-batch neighbor sampling to train on the dataset after node deletion, where the nodes that have higher similarity to the deleted nodes are sampled more frequently.

Some other concern on subspace overlapping. A reader might have some concern on whether PROJECTOR still work well if there exist an node feature that could be represented by any other node features, i.e., $\mathbf{x}_i \in \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n\}$. Indeed, PROJECTOR cannot 100% remove the information of $\mathbf{x}_i$ from the learned weight parameters. In fact, we argue that such $\mathbf{x}_i$ is not necessarily removed as an adversary also cannot distinguish whether the information is from a single user's feature or a combination of multiple users' features. However, for the completeness and to avoid the potential confusion, we propose to add some tiny random noise to all node features so that $\mathbf{x}_i \notin \text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n\}$.