Graph Unlearning

Min Chen[1*] Zhikun Zhang[1*] Tianhao Wang[2†] Michael Backes[1]
Mathias Humbert[3] Yang Zhang[1]
[1]CISPA Helmholtz Center for Information Security [2]University of Virginia [3]University of Lausanne

ABSTRACT

The right to be forgotten states that a data subject has the right to erase their data from an entity storing it. In the context of machine learning (ML), it requires the ML model provider to remove the data subject's data from the training set used to build the ML model, a process known as *machine unlearning*. While straightforward and legitimate, retraining the ML model from scratch upon receiving unlearning requests incurs a high computational overhead when the training set is large. To address this issue, a number of approximate algorithms have been proposed in the domain of image and text data, among which SISA (Sharded, Isolated, Sliced, and Aggregated) is the state-of-the-art solution. It randomly partitions the training set into multiple shards and trains a constituent model for each shard. However, directly applying SISA to the graph data can severely damage the graph structural information, and thereby the resulting ML model utility.

In this paper, we propose GraphEraser, a novel machine unlearning framework tailored to graph data. Its contributions include two novel graph partition algorithms and a learning-based aggregation method. We conduct extensive experiments on five real-world graph datasets to illustrate the unlearning efficiency and model utility of GraphEraser. We observe that GraphEraser achieves 2.06× (small dataset) to 35.94× (large dataset) unlearning time improvement compared to retraining from scratch. On the other hand, GraphEraser achieves up to 62.5% higher F1 score than that of random partitioning. In addition, our proposed learning-based aggregation method achieves up to 112% higher F1 score than that of the majority vote aggregation. Our code is available at https://github.com/MinChen00/Graph-Unlearning.

KEYWORDS

Machine Unlearning; Graph Neural Networks; Machine Learning Security and Privacy

ACM Reference Format:
Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, Yang Zhang. 2022. Graph Unlearning. In *2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/nn.nnnn/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data protection has attracted increasing attentions recently, and several regulations have been proposed to protect the privacy of individual users, such as the General Data Protection Regulation (GDPR) [1] in the European Union, the California Consumer Privacy Act (CCPA) [2] in California, the Personal Information Protection and Electronic Documents Act (PIPEDA) [3] in Canada, and the Brazilian General Data Protection Law (LGPD) in Brazil [4]. One of the most important and controversial articles in these regulations is *the right to be forgotten*, which entitles data subject the right to delete their data from an entity storing it. In the context of machine learning (ML), researchers have argued that, under the requirement of the right to be forgotten, the *model provider* has the obligation to eliminate any impact of the data whose owner requested to be forgotten, which is referred to as *machine unlearning* [11, 13].

The most straightforward machine unlearning approach consists in removing the revoked sample and retraining the ML model from scratch. However, this method can be computationally prohibitive when the underlying dataset is large. To address this issue, a number of approximate machine unlearning methods have been proposed [11, 13, 14, 21, 23, 28, 34, 42, 44], among which the SISA (Sharded, Isolated, Sliced, and Aggregated) is the most general one in terms of ML models [11]. The basic idea of SISA is to randomly split the training dataset into several disjoint shards and train each shard model separately. Upon receiving an unlearning request, the model provider only needs to retrain the corresponding shard model.

SISA has been designed to handle image and text data in the Euclidean space. However, numerous important real-world datasets are represented in the form of graphs, such as social networks [49], financial networks [40], biological networks [37], recommender systems [46, 71], or transportation networks [19]. In order to take advantage of the rich information contained in graphs, a new family of ML models, graph neural networks (GNNs), has been recently proposed and has already shown great promise [6, 12, 17, 19, 31, 38, 46, 59, 62, 67, 69]. The core idea of GNNs is to transform the graph data into low-dimensional vectors by aggregating the feature information from neighboring nodes. Similar to other ML models, GNNs can be trained on sensitive graphs such as social networks [46, 49], where the data subject may request to revoke their data. However, learning representative GNNs rely on graph structural information. Randomly partitioning the nodes into sub-graphs as in SISA could severely damage the resulting model utility. Therefore, there is a

[*]Min and Zhikun contributed equally to the paper.
[†]Tianhao did part of the work while at Purdue University and Carnegie Mellon University.

CCS '22, November 7–11, 2022, Los Angeles, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN xxx-x-xxxx-xxxx-x/xx/xx...$15.00
https://doi.org/nn.nnnn/nnnnnnn.nnnnnnn

pressing need for novel methods for unlearning previously seen – but revoked – data samples in the context of GNNs.

Our Contributions. In this paper, we propose GraphEraser, an efficient unlearning framework to achieve high unlearning efficiency and reserve high model utility in GNNs. Concretely, we first identify two common types of machine unlearning requests in the context of GNN models, namely *node unlearning* and *edge unlearning*. We then propose a general pipeline for machine unlearning in GNN models.

To permit efficient retraining while keeping the structural information of the graph, we propose two graph partition strategies. The first strategy focuses on the graph structural information and tries to preserve it to the greatest extent by relying on community detection. Our second strategy takes both graph structural and node feature information into consideration. In order to keep both pieces of information, we transform the node features and graph structure into embedding vectors that we then cluster into different shards. However, a graph partitioned by traditional community detection[50, 52, 63, 74] and clustering methods might lead to highly unbalanced shard sizes due to the structural properties of real-world graphs [22, 74]. In such case, many (if not most) of the revoked samples would belong to the largest shard whose retraining time would be substantial and the unlearning process would then become highly inefficient. We propose a general principle for balancing the shards resulting from the graph partition and instantiating it with two novel balanced graph partition algorithms to avoid this situation. In addition, considering that the different shard models do not uniformly contribute to the final prediction, we further propose a learning-based aggregation method that optimizes the importance score of the shard models to eventually improve the global model utility.

In order to illustrate the unlearning efficiency and model utility resulting from GraphEraser, we conduct extensive experiments on five real-world graph datasets. The experimental results show that GraphEraser can effectively improve the unlearning efficiency. For instance, the average unlearning time is up to 2.06× shorter on the smallest dataset and up to 35.94× shorter on the largest dataset compared to retraining from scratch, while GraphEraser achieves comparable model utility with retraining from scratch. In addition, GraphEraser provides an advanced model utility than random partitioning. Concretely, GraphEraser achieves up to 62.5% higher F1 score than that of random partitioning. Furthermore, our learning-based aggregation method can effectively improve the model utility compared to the mean and majority-vote aggregation methods. Our proposed learning-based aggregation achieves up to 93% higher F1 score than that of the mean aggregation and 112% higher F1 score than that of the majority vote aggregation .

In summary, we make the following contributions.

- To the best of our knowledge, GraphEraser is the first approach that addresses the machine unlearning problem for GNN models. Concretely, we formally define two types of machine unlearning requests in the context of GNN and propose a general pipeline for graph unlearning.
- We propose a unified principle to achieve balanced graph partitioning and instantiate it with two balanced graph partition algorithms.
- To improve the model utility resulting from GraphEraser, we propose a learning-based aggregation method.
- We conduct extensive experiments on five real-world graph datasets and four state-of-the-art GNN models to illustrate the unlearning efficiency and model utility resulting from GraphEraser.

2 PRELIMINARIES

2.1 Graph Neural Networks

We first denote an attributed graph by $\mathcal{G} = \langle \mathcal{V}, A, X \rangle$, where $\mathcal{V}$ is the set of all nodes in graph $\mathcal{G}$, $A \in \{0,1\}^{n\times n}$ is the corresponding adjacency matrix ($n = |\mathcal{V}|$), and $X \in \mathbb{R}^{n\times d_X}$ is the feature matrix with d_X being the dimension of node features. We further denote $u, v \in \mathcal{V}$ as two nodes in $\mathcal{G}$, denote $e_{u,v}$ as an edge between u and v in $\mathcal{G}$. The notations frequently used in this paper are summarized in Table 9 of Appendix A.

The basic intuition behind GNNs is that the neighboring nodes in a graph tend to have similar features; the GNN models are designed to aggregate the feature information from the neighborhood of each node to generate the node's embedding (e.g., a size-512 vector). The node embeddings can then be used to conduct downstream tasks, such as node classification [30, 46, 62], link prediction [65, 73], and graph classification [66, 68].

In this paper, we focus on *node classification tasks*, whose goal is to use a GNN to predict the label of a node $u \in \mathcal{V}$ given the node's features X_u and its neighborhood information. To train a GNN, we rely on the notion of *message passing*.

Message Passing. Typically, message passing is used to obtain the node embeddings. First, each node's features are assigned initial embeddings. In the following steps, every node receives a "message" from its neighbor nodes, then aggregates the messages as its intermediate embedding. After ℓ steps, the embedding of the node can aggregate information from its ℓ-hop neighbors. Formally, during the i-th step, the embedding E_u^i of node $u \in \mathcal{V}$ is updated using information aggregated from u's neighbors $\mathcal{N}_u$ using a pair of *aggregation operation* Φ and *updating operation* Ψ:

$$E_u^{i+1} = \Psi^i\left(E_u^i, \mathbf{m}_{\mathcal{N}_u}^i\right)$$
$$\mathbf{m}_{\mathcal{N}_u}^i = \Phi^i\left(E_v^i, \forall v \in \mathcal{N}_u)\right)$$

where $\mathbf{m}_{\mathcal{N}_u}^i$ is the "message" received from u's neighbors. There are several possible implementations of Φ and Ψ, we refer the readers to Appendix B for more details.

2.2 Machine Unlearning

Thanks to new legislation ensuring the "right to be forgotten", individuals can now formally request the deletion of their data from the service provider (or the data controller). In the ML context, this implies that the model provider should delete the revoked sample from its training set. Still, it should also eliminate any influence of the revoked sample on the resulting ML model.

Retraining from Scratch. The most direct way to implement unlearning is to delete the revoked sample and retrain the ML model from scratch by using as training set the original dataset without the deleted sample. Retraining from scratch is an effective and easy-to-enforce method for unlearning. However, when the model is complex and the original training dataset is large, the computational

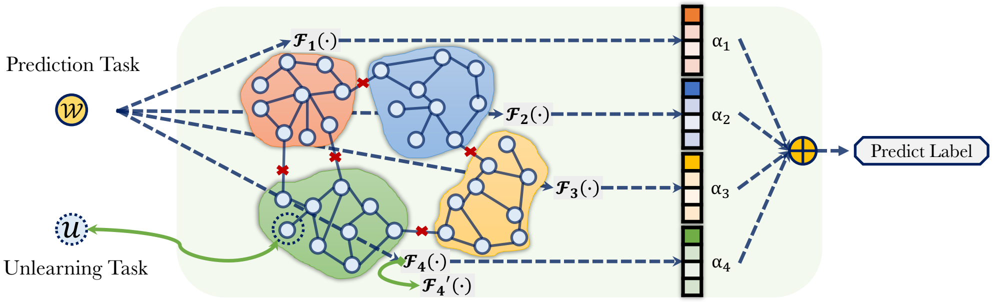


Figure 1: A schematic view of the framework of GraphEraser**. It partitions the original training graph into disjoint shards, parallelly trains a set of shard models $\mathcal{F}_i$, and learns an optimal importance score α_i for each shard model. When a node w needs prediction,** GraphEraser **sends w to all the shard models and obtains the corresponding posteriors, which are then aggregated using the optimal importance score α_i to make the prediction. When a node u mounts an unlearning request,** GraphEraser **removes u from the corresponding shard and retrains the shard model.**

overhead of retraining becomes prohibitive. In order to reduce the computational overhead, several approximation approaches have been proposed [9, 13, 20, 23, 24, 34, 54], among which SISA [11] is the most versatile one.

SISA. SISA refers to Sharded, Isolated, Sliced, and Aggregated, which is an ensemble learning-based method that can handle different ML model architectures. With this approach, the training set $\mathcal{D}_o$ is first partitioned into k disjoint shards $\mathcal{D}_o^1, \mathcal{D}_o^2, \cdots, \mathcal{D}_o^k$. These k shards are then used separately to train a set of ML models $\mathcal{F}_o^1, \mathcal{F}_o^2, \cdots, \mathcal{F}_o^k$. At inference time, the k individual predictions from the different shard models are simply aggregated (e.g., with majority voting) to provide a global prediction. When the model owner receives a request to delete a new data sample, it just needs to retrain the shard model whose shard contains this sample, leading to a significant time gain with respect to retraining the whole model from scratch.

3 GRAPH UNLEARNING

In this section, we introduce our framework, namely GraphEraser, to perform machine unlearning on GNNs. We first define the machine unlearning problem in the domain of GNNs and discuss its differences and challenges compared to the unlearning of other types of data and ML models. We then describe our general pipeline of GraphEraser.

3.1 Problem Definition

In the context of GNNs, the training set $\mathcal{D}_o$ is in the form of a graph $\mathcal{G}_o$, and a sample $x \in \mathcal{D}_o$ corresponds to a node $u \in \mathcal{G}_o$. For presentation purposes, we use *training graph* to represent *training set* in the rest of this paper. We identify two types of machine unlearning scenarios in the GNN setting, namely *node unlearning* and *edge unlearning*.

Node Unlearning. For a trained GNN model $\mathcal{F}_o$, the data of each data subject corresponds to a node in the GNN's training graph $\mathcal{G}_o$. In node unlearning, when a data subject u asks the model provider to revoke all their data, this means the model provider should unlearn u's node features and their links with other nodes from the GNN's training graph. Taking social network as an example, node unlearning means a user's profile information and social connections need to be deleted from the training graph of a target GNN. Formally, for node unlearning with respect to a node u, the model provider needs to obtain an unlearned model $\mathcal{F}_u$ trained on $\mathcal{G}_u = \mathcal{G}_o \setminus \{X_u, e_{u,v} | \forall v \in \mathcal{N}_u\}$, where X_u is the feature vector of u.

Edge Unlearning. In edge unlearning, a data subject wants to revoke one edge between their node u and another node v. Still using social network as an example, edge unlearning means a social network user wants to hide their relationship with another individual. Formally, to respond to the unlearning request for $e_{u,v}$, the model provider needs to obtain an unlearned model $\mathcal{F}_u$ trained on $\mathcal{G}_u = \mathcal{G}_o \setminus \{e_{u,v} | v \in \mathcal{N}_u\}$. The features of the two nodes remain in the training graph.

General Unlearning Objectives. In the design of machine unlearning algorithms, we consider two major factors: *unlearning efficiency* and *model utility*. The former is related to the retraining time when receiving an unlearning request. This time should be as short as possible. The latter is related to the unlearned model's prediction accuracy. Ideally, the prediction accuracy should be close to retraining from scratch. In summary, the unlearning algorithm should satisfy two general objectives: *High Unlearning Efficiency* and *Comparable Model Utility*.

Challenges of Unlearning in GNNs. As mentioned before, the state-of-the-art approach for machine unlearning is SISA [11], which randomly partitions the training set into multiple shards and trains a constituent model for each shard. SISA has shown to achieve high unlearning efficiency and comparable model utility for ML models whose inputs reside in the Euclidean space, such as images and texts. However, the input of a GNN is a graph, and data samples, i.e., nodes of the graph, are not independent identically distributed. Naively applying SISA on GNNs for unlearning, i.e., randomly partitioning the training graph into multiple shards, will destroy the training graph's structure which may result in large model utility loss. One solution is to rely on community detection methods to partition the training graph by the detected communities, which can preserve the graph structure to a large extent. However, directly adopting classical community detection methods may lead to highly unbalanced shards in terms of shard size due to the specific structural properties of real-world graphs [22, 50, 74] (see Section 4.1 for more details). In consequence, the efficiency of the unlearning process will be affected. Indeed, a revoked record would be more likely to belong to a large shard whose retraining time would be larger. Therefore, in the context of GNNs, the unlearning algorithm should satisfy the following objectives:

- **G1: Balanced Shards.** Different shards should share a similar size in terms of the number of nodes in each shard. In this way, each shard's retraining time is similar, which improves the efficiency of the whole graph unlearning process. Enforcing this objective can automatically satisfy the general unlearning pursuit of high unlearning efficiency.
- **G2: Comparable Model Utility.** Graph structural information is the major factor that determines the performance of GNN [30, 38, 66]. To achieve comparable model utility, i.e., high prediction accuracy in node classification tasks, each shard should preserve the structural properties of the training graph.

3.2 GraphEraser Framework

To address the above mentioned challenges of unlearning in GNNs, we propose GraphEraser, which consists of the following three phases: Balanced graph partition, shard model training, and shard model aggregation. The general framework of GraphEraser is illustrated in Figure 1.

Balanced Graph Partition. It is a crucial step of GraphEraser to fulfill the two requirements defined in Section 3.1. We propose to use balanced graph partition methods to partition the training graph into disjoint shards. We discuss the existing balanced graph partition methods in Section 8, and explain our proposed two balanced graph partition methods in Section 4.

Shard Model Training. After the training graph is partitioned, the model owner can train one model for each of the shard graph, referred to as the *shard model* ($\mathcal{F}_i$). All shard models share the same model architecture. To further speed up the training process, the model owner can train isolated shard models in parallel.

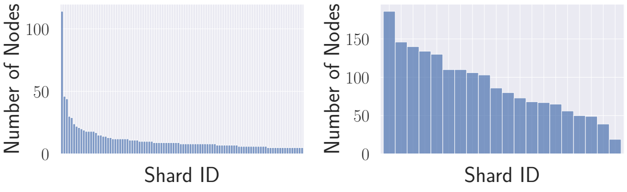
Shard Model Aggregation. At the inference phase, for predicting the label of node w, GraphEraser sends the corresponding data (the features of w, the features of its neighbors, and the graph structure among them) to all the shard models simultaneously, and the final prediction is obtained by aggregating the predictions from all the shard models. We discuss the existing aggregation strategies and introduce our learning-based aggregation LBAggr in Section 5.

4 BALANCED GRAPH PARTITION

In this section, we introduce the graph partition module. Considering the node features and graph structural information in graph data, we identify three graph partition strategies.

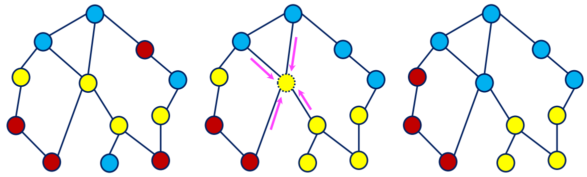
- **Strategy 0.** Consider the node feature information only and randomly partition the nodes. Concretely, we assume the node features are independently and identically distributed as in SISA. In this sense, we can randomly partition the graph based on its node IDs.

This strategy can perfectly satisfy **G1** (Balanced Shards) in Section 3.1, while it cannot satisfy **G2** (Comparable Model Utility) since it can destroy the structural information of the graph. Thus, we treat this strategy as a baseline strategy. To address **G2**, we also propose **Strategy 1** and **Strategy 2**.



(a) LPA **(b) Embedding clustering**

Figure 2: Distribution of shard sizes with classical graph partition methods on the Cora dataset. The classical LPA algorithm generates 341 shards, and we only show the top 100 shards in terms of their sizes to make the figure clearer.



(a) Initial state **(b) Propagation** **(c) Stable state**

Figure 3: Illustration of LPA's workflow. Different colors represent different shards.

- **Strategy 1.** Consider the structural information only and try to preserve it as much as possible. One promising approach to do this is relying on community detection [26, 63, 64].
- **Strategy 2.** Consider both the structural information and the node features. To implement this, we can first represent the node features and graph structure into low-dimensional vectors, namely node embeddings, and then cluster the node embeddings into different shards.

However, directly applying them can result in a highly unbalanced graph partition due to the underlying structural properties of real-world graphs (see the distribution of shard sizes with classical partition methods in Figure 2). To address this issue, we propose a general principle for achieving balanced graph partition (corresponding to **G1**), and apply this principle to design new approaches to achieve balanced graph partition for both **Strategy 1** and **Strategy 2**. In the following, we elaborate on our balanced graph partition algorithms for **Strategy 1** and **Strategy 2**.

4.1 Community Detection Method

For **Strategy 1**, we rely on community detection, which aims at dividing the graph into groups of nodes with dense connections internally and sparse connections between groups. A spectrum of community detection methods have been proposed, such as Louvain [74], Infomap [52], and Label Propagation Algorithm (LPA) [50, 63]. Among them, LPA has the advantage of low computational overhead and superior performance. Thus, in this paper, we rely on LPA to design our graph partition algorithm. For consistency purposes, we use *shard* to represent the *community*.

Label Propagation Algorithm. Figure 3 gives an illustration of the workflow of LPA. At the initial state, each node is assigned a random shard label (Figure 3a). During the label propagation phase

(Figure 3b → Figure 3c), each node sends out its own label, and updates its label to be the majority of the labels received from its neighbors. For instance, the yellow node with a dashed outline in Figure 3b will change its label to blue because the majority of its neighbors (two nodes above it) are labeled blue. The label propagation process iterates through all nodes multiple times until convergence (there are no nodes changing their labels).

Unbalanced Partition. LPA is intriguing and powerful; however, directly applying the classical LPA results in a highly unbalanced graph partition. For instance, Figure 2a shows the distribution of shard size on the Cora dataset [70] (2166 nodes in the training graph). We observe that the largest shard contains 113 nodes, while the smallest one contains only 2 nodes. We provide a visualization of the shards detected by classical LPA in Appendix C. Directly adopting the unbalanced shards detected by the classical LPA does not satisfy **G1**, which severely affects the unlearning efficiency. For instance, if the revoked node is in the largest shard, there is not much benefit in terms of unlearning time.

General Principle to Achieve Balanced Partition. To address the above issue, we propose a general recipe to achieve balanced graph partition. Given the desired shard number k and maximal shard size δ, we define a *preference* for every *node-shard pair* representing the node is assigned to the shard (which is referred to as *destination shard*). This results in $k \times n$ node-shard pairs with different preference values. Then, we sort the node-shard pairs by preference values. For each pair in descending preference order, we assign the node to the destination shard if the current number of nodes in that destination shard does not exceed δ.

Balanced LPA (BLPA**).** Following the general principle for achieving balanced partition, we define the preference as the *neighbor counts* (the number of neighbors belonging to a destination shard) of the node-shard pairs, and the node-shard pairs with larger neighbor counts have higher priority to be assigned.

Algorithm 1 gives the workflow of BLPA. The algorithm takes as input the set of nodes $\mathcal{V}$, the adjacency matrix A, the number of desired shards k, the maximum number of nodes in each shard δ, maximum iteration T, and works in four steps as follows:

- **Step 1: Initialization.** We first randomly assign each node to one of the k shards (Line 2).
- **Step 2: Reassignment Profiles Calculation.** For each node u, we denote its *reassignment profile* using a tuple $\langle u, \mathbb{C}_{src}, \mathbb{C}_{dst}, \xi \rangle$, where $\mathbb{C}_{src}$ and $\mathbb{C}_{dst}$ are the current and destination shards of node u, ξ is the neighbor counts of the destination shard $\mathbb{C}_{dst}$ (Line 5 - Line 7). We store all the reassignment profiles in $\mathbb{F}$.
- **Step 3: Reassignment Profiles Sorting.** We rely on the intuition that the reassignment profile with larger neighbor counts should have a higher priority to be fulfilled; thus we sort $\mathbb{F}$ in descending order by ξ and obtain $\mathbb{F}_s$ (Line 8).
- **Step 4: Label Propagation.** Finally, we enumerate every instance of $\mathbb{F}_s$. If the size of the destination shard $\mathbb{C}_{dst}$ does not exceed the given threshold δ, we add the node u to the destination shard and remove it from the current shard (Line 9 - Line 12). After that, we remove all the remaining tuples containing node u from $\mathbb{F}_s$.

Algorithm 1: BLPA Algorithm

```
    Input: The set of all nodes $\mathcal{V}$, adjacency matrix $A$, number
           of shards $k$, maximum number of nodes in each
           shard $\delta$; maximum iteration $T$;
    Output: Shards $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \cdots, \mathbb{C}_k\}$;
 1  Initialization:
 2  Randomly allocate all nodes into $k$ shards and obtain
      $\mathbb{C}^0 = \{\mathbb{C}^0_1, \mathbb{C}^0_2, \cdots, \mathbb{C}^0_k\}$, step $t = 0$;
 3  Label Propagation:
 4  while True do
 5      foreach node $u$ in $\mathcal{V}$ do
 6          foreach shard $\mathbb{C}_{dst}$ in $\{\mathbb{C}_i | v \in \mathcal{N}_u, v \in \mathbb{C}_i\}$ do
 7              Store tuple $\langle u, \mathbb{C}_{src}, \mathbb{C}_{dst}, \xi \rangle$ in $\mathbb{F}$;
 8      Sort $\mathbb{F}$ by $\xi$ in descending order and obtain $\mathbb{F}_s$;
 9      foreach tuple in $\mathbb{F}_s$ do
10          if $|\mathbb{C}^t_{dst}| < \delta$ then
11              $\mathbb{C}^t_{dst} \leftarrow \mathbb{C}^{t-1}_{dst} \cup u$;
12              $\mathbb{C}^t_{src} \leftarrow \mathbb{C}^{t-1}_{src} \setminus u$;
13              Remove all the remaining tuples containing
                  node $u$ from $\mathbb{F}_s$;
14      if $t > T$ or the shard does not change then
15          break;
16      $t \leftarrow t + 1$;
17  return $\mathbb{C}^t$.
```

The BLPA algorithm repeats steps 2-4 until the algorithm reaches the maximum iteration T, or the shard does not change (Line 14 - Line 15).

Algorithm Analysis. The computational complexity of BLPA depends on the size of the reassignment profile $\mathbb{F}$. Based on its definition, the number of tuples of each node u in $\mathbb{F}$ equals to the number of neighbors of u. Thus, the computational complexity of BLPA is $O(n \cdot d_{ave})$, where n is the number of nodes, and d_{ave} is the average node degree of the training graph.

Regarding the convergence of BLPA, it is difficult to theoretically prove the convergence. Instead, we conduct empirical experiments to validate the convergence performance by checking the number of changed nodes in each iteration. We refer the readers to Appendix D for the detailed experimental results.

4.2 Embedding Clustering Method

For **Strategy 2**, we rely on embedding clustering which takes into consideration both the node features and the graph structural information for the graph partitioning. In order to partition the graph, we first use a pretrained GNN model to obtain all the node embeddings, and then we perform clustering on the resulting node embeddings.

Embedding Clustering. We can adopt any state-of-the-art GNN models introduced in Section 2.1 to project each node into an embedding space. With respect to clustering, we rely on the widely used k-means algorithm[35], which consists of three phases: Initialization, nodes reassignment, and centroids updating. In the initialization phase, we randomly sample k *centroids* which represent the "center" of each shard. In the node reassignment phase, each node is assigned to its "nearest" shard in terms of the Euclidean distance from the centroids. In the centroids updating phase, the new centroids are recalculated as the average of all the nodes in their corresponding shard.

Similar to the case of the LPA method, directly using k-means can also produce highly unbalanced shards. In Figure 2b, we observe that on the Cora dataset, the largest shard contains 10.24% of the nodes, while the smallest one only contains 1.05% of the nodes.

Balanced Embedding k-means (BEKM). Following the same principle for achieving a balanced partition, we propose BEKM as shown in Algorithm 2. We define the preference as the Euclidean distance between the node embedding and the centroid of the shard for all the node-shard pairs. A shorter distance implies a higher priority. BEKM takes as input the set of all node embeddings $\mathbb{E} = \{E_1, E_2, \cdots, E_n\}$, the number of desired shards k, the maximum number of node embeddings in each shard δ, the maximum number of iterations T, and works in four steps as follows:

- **Step 1: Initialization.** We first randomly select k centroids $C^0 = \{C_1^0, C_2^0, \cdots, C_k^0\}$ (Line 2).
- **Step 2: Embedding-Centroid Distance Calculation.** Then, we calculate all the pairwise distance between the node embeddings and the centroids, which results in $n \times k$ embedding-centroid pairs. These pairs are stored in $\mathbb{F}$ (Line 5 - Line 7).
- **Step 3: Embedding-Centroid Distance Sorting.** We rely on the intuition that the embedding-centroid pairs with closer distance have higher priorities; thus we sort $\mathbb{F}$ in ascending order and obtain $\mathbb{F}_s$ (Line 8).
- **Step 4: Node Reassignment and Centroid Updating.** For each embedding-centroid pair in $\mathbb{F}_s$, if the size of $\mathbb{C}_j$ is smaller than δ, we assign node u to shard $\mathbb{C}_j$ (Line 9 - Line 15). After that, we remove all the remaining tuples containing node i from $\mathbb{F}_s$. Finally, the new centroids are calculated as the average of all the nodes in their corresponding shards.

The BEKM algorithm repeats steps 2-4 until the algorithm reaches the maximum iteration T, or the centroid does not change (Line 16 - Line 17).

Algorithm Analysis. Similar to BLPA, the computational complexity of BEKM depends on the size of $\mathbb{F}$. Since there are n nodes and k shards, the computational complexity of BEKM is $O(k \cdot n)$. We empirically validate the convergence performance of BEKM in Appendix D.

4.3 Discussion

Choice of Graph Partition Algorithms. The choice between BLPA and BEKM depends on the GNN structure. In Section 6.3, we provide a guideline on which one to choose. In addition, we emphasize that GraphEraser is a general framework for graph unlearning, and any other balanced graph partition methods can be plugged into it. In Section 6.5, we empirically compare our proposed BLPA and BEKM with several existing representative balanced graph partition methods, and show that our proposed methods are either more computational efficient or better performing.

Guarantee of Unlearning. Note that the shard models are deterministically unlearned but the clustering (graph partition) is not; thus, GraphEraser is doing approximate unlearning. As such, we empirically quantify the possible information leakage using the state-of-the-art information leakage quantification method for machine unlearning system [15] in Section 6.6, and show that GraphEraser does not leak much extra information.

Algorithm 2: BEKM Algorithm

```
   Input: Node embeddings $\mathbb{E} = \{E_1, E_2, \cdots, E_n\}$, the number
          of clusters $k$, maximum number of nodes embedding
          in each cluster $\delta$; maximum number of iteration $T$;
   Output: Clusters $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \cdots, \mathbb{C}_k\}$;
 1 Initialization:
 2 Randomly select $k$ centroids $C^0 = \{C_1^0, C_2^0, \cdots, C_k^0\}$, step
     $t = 0$;
 3 while True do
 4     Nodes Reassignment:
 5     foreach node embedding $i \in \mathbb{E}$ do
 6         foreach centroid $j \in \mathbb{C}$ do
 7             Store $||E_i - C_j||_2$ in $\mathbb{F}$;
 8     Sort $\mathbb{F}$ in ascending order and obtain $\mathbb{F}_s$.
 9     foreach node $i$ and centroid $j$ in $\mathbb{F}_s$ do
10         if $|\mathbb{C}_j^t| < \delta$ then
11             $\mathbb{C}_j^t \leftarrow \mathbb{C}_j^t \cup i$;
12             Remove all the remaining tuples containing
                 node $i$ from $\mathbb{F}_s$;
13     Centroids Updating:
14     foreach cluster $j \in \mathbb{C}^t$ do
15         $C_j^t = \frac{\sum_{i \in \mathbb{C}_j^t} E_i}{|\mathbb{C}_j^t|}$;
16     if $t > T$ or the centroid do not change then
17         break;
18     $t \leftarrow t + 1$;
19 return $\mathbb{C}^t$.
```

Furthermore, from a legal-scholarship perspective, does not repartition the graph also satisfies the right to be forgotten. Note that legal-scholarship is vague and open to different explanations; below we just explain our understanding. The third item of Art. 7 in the GDPR states that: "*The data subject shall have the right to withdraw his or her consent at any time. The withdrawal of consent* ***shall not affect the lawfulness of processing based on consent before its withdrawal.***" In our case, graph partition is a preprocessing step of the graph dataset, and the partitioned graph can be regarded as another form of the raw training graph. It suffices to delete the data owners' revoked data from the processed data, e.g., remove the revoked node from the partitioned graph in our case, instead of removing the revoked data from the raw dataset and redo the preprocess operations. This is supported by the application of the right to be forgotten in search engines [10]: When the data owners ask to delete their data from the search results of a search engine, the service providers such as Google only need to directly delete the data from the current search results, instead of rerunning the ranking and recommendation algorithms on the raw data.

5 LEARNING-BASED AGGREGATION (LBAggr)

Existing Aggregation Strategies. The most straightforward aggregation strategy, also mainly used in [11], is majority voting, where each shard model predicts a label and w takes the label predicted most often. We refer to this aggregation strategy as MajAggr. An alternative solution is to gather the posterior vectors of all shard models and average them to obtain aggregated posteriors. The target nodes are predicted as the highest posterior in this aggregation. We refer to this aggregation strategy as MeanAggr.

Note that different shard models can have different contributions to the final prediction; thus allocating the same *importance score* for all shard models during the aggregation phase might not achieve the best prediction accuracy.

Our Proposal. In this section, we propose a learning-based aggregation method LBAggr. We assign an importance score to each shard model, which can be learned based on the following loss function.

$$\min_{\alpha} \mathop{\mathbb{E}}_{w \in \mathcal{G}_o} \left[\mathcal{L} \left(\sum_{i=0}^{m} \alpha_i \cdot \mathcal{F}_i(X_w, \mathcal{N}_w), y \right) \right] + \lambda \sum_{i=0}^{m} ||\alpha_i|| \qquad (1)$$

where X_w and $\mathcal{N}_w$ are the feature vector and neighborhood of a node w from the training graph, y is the true label of w, $\mathcal{F}_i(\cdot)$ represents shard model i, α_i is the importance score for $\mathcal{F}_i(\cdot)$, and m is the total number of shards. We regulate the summation of all importance scores to 1. Further, $\mathcal{L}$ represents the loss function and we adopt the standard cross-entropy loss in this paper. The regularization term $|| \cdot ||$ is used to reduce overfitting.

Solving the Optimization Problem. To solve the optimization problem, we can run gradient descent to find the optimal α. However, directly running gradient descent can result in negative values in α. To address this problem, after each gradient descent iteration, we map the negative importance score back to 0. The procedure of mapping the negative importance scores to 0 follows the general idea of projected gradient descent (PGD) [7]. In addition, the summation of the importance scores could deviate from 1. We have tried to normalize the importance score using the summation of current scores in each iteration; however, we empirically found that the loss could be extremely unstable across different epochs. Thus, instead of using summation, we use the softmax function for normalization in each iteration.

Importance Scores Unlearning. Note that we use the nodes in the training graph to learn the optimal importance scores. However, these nodes can also be requested to be revoked by their data subjects. Therefore, we need to relearn the shard importance scores if a request-to-unlearn node is used to train the LBAggr, and this learning time is counted as part of the unlearning time. To reduce this relearning time, we propose to use a small random subset of nodes from the training graph to learn the shard importance scores. We empirically show in Section 6.4 that using only 10% of the nodes in the training graph can achieve comparable utility as that of using all nodes. In this sense, we do not need to relearn the optimal shard importance scores when the unlearned nodes are not used to train the LBAggr.

5.1 Putting Things Together: GraphEraser

Algorithm 3 illustrates the overall workflow of GraphEraser. It takes as input the training graph $\mathcal{G}_0$, the GNN model type f, and all necessary parameters for Algorithm 1 and Algorithm 2 (k, δ, and T). If f is a GCN, we invoke Algorithm 1 to partition $\mathcal{G}_0$; otherwise, we use Algorithm 2 (Line 1 - Line 5). We then use the partitioned graph $\mathcal{G}_s$ to train a set of shard models $\mathcal{F}$ (Line 6). Finally, we randomly sample a set of nodes $\mathcal{V}_0$ from $\mathcal{G}_0$ to train the importance scores α for each shard models. The shard models and importance scores produced by GraphEraser can be used to predict the label of new samples. When some nodes or edges are revoked by the data owner, we only need to retrain the corresponding shard model.

```
Algorithm 3: GraphEraser
   Input: Training graph $\mathcal{G}_0$, GNN model type $f$, number of
          shards $k$, maximum number of nodes in each shard
          $\delta$, maximum iteration $T$;
   Output: Shard models $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_k\}$, importance
           scores $\alpha = \{\alpha_1, \alpha_2, \cdots \alpha_k\}$;
 1 Graph Partition:
 2 if the GNN model type $f$ is GCN: then
 3     Partitioning $\mathcal{G}_0$ into $k$ shards with Algorithm 1 and
         obtain $\mathcal{G}_s = \{\mathcal{G}_s^1, \mathcal{G}_s^2, \cdots, \mathcal{G}_s^k\}$;
 4 else
 5     Partitioning $\mathcal{G}_0$ into $k$ shards with Algorithm 2 and
         obtain $\mathcal{G}_s = \{\mathcal{G}_s^1, \mathcal{G}_s^2, \cdots, \mathcal{G}_s^k\}$;
 6 Shard Model Training:
 7 Using $\mathcal{G}_s$ to train shard models $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_k\}$;
 8 Importance Scores Learning:
 9 Randomly sampling a set of nodes $\mathcal{V}_0$ from $\mathcal{G}_0$;
10 Replacing $\mathcal{G}_0$ in Equation 1 with $\mathcal{V}_0$ and train $\alpha$;
11 return $\mathcal{F}$, $\alpha$.
```

6 EVALUATION

In this section, we first evaluate the unlearning efficiency and model utility of GraphEraser, respectively. Second, we conduct experiments to show the superiority of our proposed learning-based aggregation method LBAggr. Third, we compare our proposed balanced graph partition methods with existing methods. Fourth, we illustrate the unlearning power of GraphEraser.

In addition, we investigate the following issues, and due to space limitation, the corresponding results are deferred to the appendix: (1) We investigate the correlation between the properties of the shard models and the importance scores resulting from LBAggr (Appendix F). (2) We conduct ablation studies to show the impact of k and δ on the unlearning efficiency and model utility (Appendix G). (3) We show the robustness of GraphEraser to the number of unlearned nodes/edges (Appendix H). (4) We investigate the impact of graph structure in a more controllable manner (Appendix I).

6.1 Experimental Setup

Datasets. We conduct our experiments on five public graph datasets, including Cora, Citeseer, Pubmed [70], CS [55], and Physics [55]. These datasets are widely used as benchmark datasets for evaluating the performance of GNN models [38, 57, 75]. We refer the readers to Appendix E for the detailed description of these datasets. Table 1 summarizes the statistics of all the datasets.

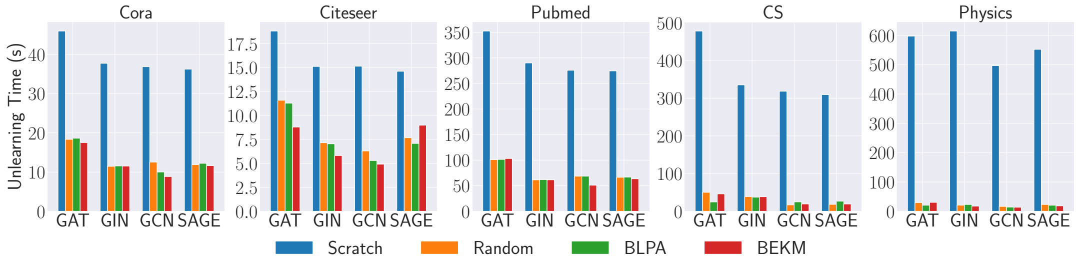


Figure 4: Comparison of node unlearning efficiency for all graph unlearning methods. BLPA **and** BEKM **stand for** GraphEraser-BLPA **and** GraphEraser-BEKM **unlearning methods, respectively. The results of edge unlearning are in Figure 14.**

Table 1: Dataset statistics.

Dataset	Category	#. Nodes	#. Edges	#. Classes	#. Features
Cora	Citation	2,708	5,429	7	1,433
Citeseer	Citation	3,327	4,732	6	3,703
Pubmed	Citation	19,717	44,338	3	500
CS	Coauthor	18,333	163,788	15	6805
Physics	Coauthor	34,493	495,924	5	8415

GNN Models. We evaluate the efficiency and utility of GraphEraser on four state-of-the-art GNN models, including SAGE, GCN, GAT, and GIN (discussed in Appendix B). For each GNN model, we stack two layers of GNN modules. All the models are implemented with the PyTorch Geometric[1] library. All the GNN models (including the shard models) considered in this paper are trained for 100 epochs. We use Adam optimizer and set default learning rate to 0.01 with 0.001 weight decay.

Metrics. In the design of GraphEraser, we mainly consider two aspects of performance, unlearning efficiency and model utility.

- **Unlearning Efficiency.** Directly measuring the unlearning time for one unlearning request is inaccurate due to the diversity of shards. Thus, we calculate the *average unlearning time* for 100 independent unlearning requests. Concretely, we randomly sample 100 nodes/edges from the training graph, record the retraining time of their corresponding shard models, and calculate the average retraining time.
- **Model Utility.** We use the *Micro F1 score* to measure the model utility, which is widely used to evaluate the prediction ability of GNN models on multi-class classification [27]. The F1 score is a harmonic mean of *precision* and *recall*, and can provide a good measure of the incorrectly classified cases.

Competitors. We have two natural baselines: The training from scratch method (which is referred to as Scratch) and the random method (which is based on partitioning the training graph randomly rely on Strategy 0, and we refer it to as Random). The Scratch method can achieve good model utility but its unlearning efficiency is low. On the other hand, the Random method can achieve high unlearning efficiency but suffers from poor model utility.

We implement both community detection and embedding clustering based graph partition methods in Section 4 for GraphEraser. For presentation purpose, we refer them to as GraphEraser-BLPA and GraphEraser-BEKM, respectively.

Experimental Settings. For each dataset, we randomly split the whole graph into two disjoint parts, where 80% of nodes are used in the training graph of GNN models, and 20% of nodes are used to evaluate the model utility.

Note that the graph partition algorithms are only applied to the training graph. By default, we set the number of shards k for Cora, Citeseer, Pubmed, CS, and Physics to 20, 20, 50, 30, and 100, respectively, which ensures each shard is trained on a reasonable number of nodes and edges. The maximum number of nodes in each shard δ is set as $\lceil \frac{n}{k} \rceil$. We validate the effectiveness of this setting in Appendix G. The maximum number of iterations T of both BLPA and BEKM are set to 30. We show in Appendix D that $T = 30$ can guarantee the convergence of both algorithms. Besides, we set the embedding dimension of BEKM as 32. **Implementation.** We implement GraphEraser with Python 3.7 and PyTorch 1.7. All experiments are run on an NVIDIA DGX-A100 server with 2 TB memory and Ubuntu 18.04 LTS system. All the experiments regarding to model utility are run 10 times and we report the mean and standard deviation.

6.2 Evaluation of Unlearning Efficiency

In this section, we evaluate the unlearning efficiency of different graph unlearning methods on five datasets and four GNN models. **Setup.** Figure 4 illustrates the node unlearning efficiency for different graph unlearning methods. For the shard-based unlearning methods, i.e., Random, GraphEraser-BLPA, and GraphEraser-BEKM, each unlearning request time cost consists of two parts: Retraining the shard models and relearning the importance scores of LBAggr. As discussed in Section 5, we only use a small portion of nodes in the training graph to learn the importance scores. The *average relearning time* of LBAggr on all datasets is shown in the last column of Table 2. The results show that the relearning time is less than 30s for most of the datasets, which is negligible compared to retraining the shard models.

Results. We observe that the shard-based unlearning methods can significantly improve the unlearning efficiency compared to the Scratch method. For all the four GNN models, we observe a similar time efficiency improvement level. In addition, the relative efficiency improvement of larger datasets (Pubmed, CS, and Physics) is more significant than that of smaller datasets (Cora and Citeseer). For instance, the unlearning time improvement is of 4.16× for the Cora dataset, 3.08× for the Citeseer dataset, 5.40× for the Pubmed dataset, 19.25× for the CS dataset, and 35.9× for the Physics datasets. This is expected. From the Scratch method perspective,

[1] https://github.com/rusty1s/pytorch_geometric

Table 2: Computational costs of the GraphEraser pipeline on five datasets. We report the prediction cost and the relearning cost of LBAggr for BEKM.

Dataset	Graph Partition Cost Random	 BLPA	 BEKM	Prediction Cost Scratch	 Shard	Learn Cost of LBAggr
Cora	0.8s	3s	26s	0.002s	0.003s	1.3s
Citeseer	0.5s	2s	20s	0.003s	0.004s	1.5s
Pubmed	1s	20s	240s	0.004s	0.008s	19s
CS	1s	13s	220s	0.004s	0.009s	25s
Physics	1s	40s	480s	0.005s	0.021s	33s

training a large graph can cost a large amount of time. From the shard-based methods perspective, we can tolerate more shards for larger graphs while preserving the model utility. Comparing different shard-based methods, we observe that GraphEraser-BLPA and GraphEraser-BEKM have similar unlearning time as Random. This is made possible by our approach for achieving balanced partition with BLPA and BEKM (see Section 4).

Additional Time Cost Analysis. Besides the unlearning cost, there are two additional costs in the GraphEraser pipeline: Graph partition cost and prediction cost. Table 2 illustrates these two costs on five datasets. We observe that the graph partition costs of BLPA and BEKM are higher than Random. This is expected since both BLPA and BEKM need to iterate multiple times to preserve the structural information. Once the graph partition is done, we keep it fixed without unlearning it. In this sense, we can tolerate this cost since it is only executed once. We further show in Appendix H that using a fixed partition does not result in noticeable model utility degradation for GraphEraser.

For the prediction cost, the shard-based methods are slightly more time-consuming compared to the Scratch method, since we need to obtain the prediction from all shard models and aggregate them. Fortunately, the prediction cost is negligible since most of their values are smaller than 0.01 second.

We reach similar conclusions for edge unlearning whose results are shown in Appendix K.

6.3 Evaluation of Model Utility

Next, we evaluate the model utility of different graph unlearning methods. Table 3 (the red ground columns) shows the experimental results for node unlearning. For a fair comparison, we also apply LBAggr for Random.

Influence of Datasets. We first observe that on the Cora and Citeseer datasets, our proposed method, GraphEraser-BEKM and GraphEraser-BLPA, can achieve a much better F1 score compared to the Random method. For instance, on the GCN model trained on the Cora dataset, the F1 score for GraphEraser-BLPA is 0.676, while the corresponding result is 0.509 for Random. For the Pubmed, CS, and Physics datasets, the F1 score of the Random method is comparable to GraphEraser-BEKM and GraphEraser-BLPA, and can even achieve a similar F1 score as the Scratch method in some settings. We conjecture this is due to the different contributions of the graph structural information to the utility of GNN models. Intuitively, if the graph structural information does not contribute much to the GNN models, it is not surprising that the Random method can achieve comparable model utility as GraphEraser-BLPA and GraphEraser-BEKM.

To validate whether the graph structural information indeed diversely affects the GNN models' performance among different datasets, we introduce a baseline that uses a 3-layer MLP (multilayer perceptron) to train the prediction models for all datasets. Note that we only use the node features to train the MLP model, without considering any graph structural information. Table 4 depicts the comparison of the F1 scores between the MLP model and four GNN models on five datasets. We observe that for the Cora and Citeseer datasets, the F1 score of the MLP model is significantly lower than that of the GNN models, which means the graph structural information plays a major role in the GNN models. On the other hand, the MLP model can achieve adequate F1 score compared to the GNN models on Pubmed, CS, and Physics datasets, which means the graph structural information does not contribute much in the GNN models. To better illustrate the correlation between the importance of the graph structure and the utility improvement over Random, we conduct an ablation study in Appendix G.

In conclusion, the contribution of the graph structural information to the GNN model can significantly affect the behaviors of different shard-based graph unlearning methods.

Guideline for Choosing an Unlearning Method. In practice, we would suggest the model provider evaluate the role of graph structure before choosing a proper graph unlearning method. To this end, they can first compare the F1 score of MLP and GNN, if the gap in the F1 score between MLP and GNN is small, the Random method can be a good choice since it is much easier to implement, and it can achieve comparable model utility as GraphEraser-BLPA and GraphEraser-BEKM. Otherwise, GraphEraser-BLPA and GraphEraser-BEKM are better choices due to better model utility.

Regarding the choice between the two shard partition methods, i.e., GraphEraser-BLPA and GraphEraser-BEKM, we empirically observe that if the GNN follows the GCN structure, one can choose GraphEraser-BLPA, otherwise, one can adopt GraphEraser-BEKM. We posit this is because the GCN model requires the node degree information for normalization (see Section 2.1), and the GraphEraser-BLPA can preserve more local structural information thus better preserve the node degree [64].

Comparison with Scratch. Interestingly, we could observe that GraphEraser-BEKM performs slightly better than Scratch in some cases. For instance, the F1 score of GraphEraser-BEKM is 0.801 on the Cora dataset and the GIN model, while the corresponding F1 score of Scratch is 0.787. There are two possible reasons for this phenomenon. First, sampling often can eliminate some "noise" in the dataset, which is consistent with the observation of prior studies [16, 72]. Second, GraphEraser makes the final prediction by aggregating all submodels' results, in this sense, GraphEraser performs an ensemble which is another way to improve model performance.

We reach similar conclusions for edge unlearning whose results are shown in Appendix K. Considering the conclusions for node unlearning and edge unlearning are similar in terms of both unlearning efficiency and model utility, we only provide the results for node unlearning in the following parts.

Table 3: Comparison of F1 scores for unlearning methods and different aggregation methods. Note that the Scratch method does not need aggregation. We highlight the Scratch method in the green ground and our proposed methods in the red . For each graph partition strategy, we highlight the best value in bold. and for each GNN model, we highlight the best value in blue bold. The results of edge unlearning are in Table 10.

Dataset/		Scratch	Random			GraphEraser-BLPA			GraphEraser-BEKM		
	Model		MeanAggr	MajAggr	LBAggr	MeanAggr	MajAggr	LBAggr	MeanAggr	MajAggr	LBAggr
Cora	GAT	0.823 ± 0.006	0.649 ± 0.006	0.638 ± 0.010	**0.706 ± 0.004**	0.356 ± 0.005	0.492 ± 0.009	**0.727 ± 0.009**	0.672 ± 0.004	0.669 ± 0.012	**0.754 ± 0.009**
	GCN	0.739 ± 0.003	0.337 ± 0.006	0.188 ± 0.004	**0.509 ± 0.009**	0.590 ± 0.008	0.319 ± 0.007	**0.676 ± 0.004**	0.390 ± 0.011	0.247 ± 0.012	**0.493 ± 0.006**
	GIN	0.787 ± 0.013	0.760 ± 0.030	0.702 ± 0.033	**0.736 ± 0.021**	0.681 ± 0.039	0.594 ± 0.028	**0.753 ± 0.015**	0.758 ± 0.016	0.742 ± 0.031	**0.801 ± 0.018**
	SAGE	0.824 ± 0.004	0.583 ± 0.009	0.572 ± 0.012	**0.682 ± 0.013**	0.354 ± 0.008	0.486 ± 0.012	**0.684 ± 0.014**	0.673 ± 0.008	0.646 ± 0.010	**0.740 ± 0.013**
Citeseer	GAT	0.691 ± 0.015	0.502 ± 0.012	0.507 ± 0.016	**0.631 ± 0.015**	0.504 ± 0.010	0.486 ± 0.009	**0.676 ± 0.004**	0.744 ± 0.007	0.712 ± 0.010	**0.746 ± 0.006**
	GCN	0.493 ± 0.006	0.263 ± 0.014	0.157 ± 0.011	**0.277 ± 0.009**	0.372 ± 0.006	0.192 ± 0.006	**0.450 ± 0.006**	0.298 ± 0.005	0.129 ± 0.007	**0.332 ± 0.006**
	GIN	0.587 ± 0.031	0.611 ± 0.028	0.540 ± 0.056	**0.626 ± 0.022**	0.451 ± 0.062	0.447 ± 0.032	**0.612 ± 0.026**	0.725 ± 0.016	0.696 ± 0.014	**0.739 ± 0.020**
	SAGE	0.668 ± 0.013	0.519 ± 0.024	0.536 ± 0.026	**0.623 ± 0.014**	0.447 ± 0.007	0.472 ± 0.024	**0.657 ± 0.012**	0.708 ± 0.003	0.710 ± 0.007	**0.716 ± 0.007**
Pubmed	GAT	0.851 ± 0.004	0.852 ± 0.001	0.851 ± 0.002	**0.857 ± 0.002**	0.843 ± 0.002	0.840 ± 0.002	**0.858 ± 0.003**	0.853 ± 0.001	0.852 ± 0.001	**0.860 ± 0.003**
	GCN	0.748 ± 0.017	0.484 ± 0.004	0.207 ± 0.000	**0.551 ± 0.005**	0.644 ± 0.004	0.423 ± 0.011	**0.718 ± 0.010**	0.353 ± 0.003	0.207 ± 0.000	**0.482 ± 0.003**
	GIN	0.837 ± 0.015	0.854 ± 0.003	0.852 ± 0.003	**0.856 ± 0.003**	0.849 ± 0.002	0.843 ± 0.002	**0.855 ± 0.004**	0.859 ± 0.002	0.851 ± 0.010	**0.859 ± 0.003**
	SAGE	0.874 ± 0.003	0.854 ± 0.002	0.852 ± 0.003	**0.857 ± 0.002**	0.841 ± 0.003	0.836 ± 0.003	**0.863 ± 0.002**	0.854 ± 0.002	0.852 ± 0.002	**0.862 ± 0.002**
CS	GAT	0.919 ± 0.004	0.880 ± 0.001	0.877 ± 0.001	**0.882 ± 0.000**	0.664 ± 0.015	0.662 ± 0.009	**0.858 ± 0.004**	0.885 ± 0.001	0.882 ± 0.003	**0.906 ± 0.002**
	GCN	0.903 ± 0.006	0.644 ± 0.002	0.528 ± 0.001	**0.706 ± 0.008**	0.658 ± 0.004	0.440 ± 0.003	**0.750 ± 0.023**	0.620 ± 0.003	0.502 ± 0.003	**0.812 ± 0.012**
	GIN	0.867 ± 0.005	0.856 ± 0.006	0.839 ± 0.004	**0.858 ± 0.005**	0.655 ± 0.024	0.691 ± 0.011	**0.789 ± 0.013**	0.857 ± 0.005	0.844 ± 0.005	**0.891 ± 0.002**
	SAGE	0.932 ± 0.004	0.896 ± 0.005	0.896 ± 0.003	**0.905 ± 0.004**	0.745 ± 0.009	0.679 ± 0.003	**0.886 ± 0.010**	0.904 ± 0.007	0.903 ± 0.001	**0.927 ± 0.002**
Physics	GAT	0.955 ± 0.005	0.917 ± 0.001	0.915 ± 0.001	**0.920 ± 0.002**	0.871 ± 0.032	0.858 ± 0.044	**0.921 ± 0.004**	0.920 ± 0.001	0.917 ± 0.000	**0.925 ± 0.001**
	GCN	0.947 ± 0.002	0.597 ± 0.001	0.533 ± 0.001	**0.747 ± 0.010**	0.817 ± 0.003	0.770 ± 0.001	**0.858 ± 0.008**	0.575 ± 0.003	0.506 ± 0.001	**0.815 ± 0.001**
	GIN	0.934 ± 0.003	0.903 ± 0.002	0.916 ± 0.001	**0.921 ± 0.002**	0.842 ± 0.009	0.840 ± 0.006	**0.907 ± 0.003**	0.924 ± 0.002	0.919 ± 0.001	**0.926 ± 0.001**
	SAGE	0.956 ± 0.005	0.712 ± 0.003	0.717 ± 0.002	**0.823 ± 0.011**	0.905 ± 0.003	0.894 ± 0.003	**0.922 ± 0.001**	0.926 ± 0.003	0.924 ± 0.002	**0.933 ± 0.001**

Table 4: Comparison of F1 scores for MLP model and four GNN models. Larger gap in F1 scores for MLP model and GNN models means that the graph structural information is more important for the GNN models.

Model	**Cora**	**Citeseer**	**Pubmed**	**CS**	**Physics**
MLP	0.657 ± 0.019	0.587 ± 0.029	0.868 ± 0.002	0.927 ± 0.007	0.950 ± 0.003
GAT	0.823 ± 0.006	0.691 ± 0.015	0.851 ± 0.004	0.919 ± 0.004	0.955 ± 0.005
GCN	0.739 ± 0.003	0.493 ± 0.006	0.748 ± 0.017	0.903 ± 0.006	0.947 ± 0.002
GIN	0.787 ± 0.013	0.587 ± 0.031	0.837 ± 0.015	0.867 ± 0.005	0.934 ± 0.003
SAGE	0.824 ± 0.004	0.668 ± 0.013	0.874 ± 0.003	0.932 ± 0.004	0.956 ± 0.005

Table 5: Impact of the number of training nodes for learning LBAggr. "10%" and "1000" stand for randomly selecting 10% and 1000 nodes from the training graph, respectively. "All" stands for using all nodes in the training graph. We highlight our recommended choices in the red ground.

$\mathcal{F}$	**#. Nodes**	**Cora**	**Citeseer**	**Pubmed**	**CS**	**Physics**
GAT	**10%**	0.70 ± 0.02	0.71 ± 0.01	0.86 ± 0.00	0.91 ± 0.00	0.93 ± 0.00
	1000	0.73 ± 0.01	0.72 ± 0.02	0.86 ± 0.00	0.91 ± 0.01	0.93 ± 0.00
	All	0.74 ± 0.00	0.72 ± 0.00	0.86 ± 0.00	0.91 ± 0.00	0.93 ± 0.00
GCN	**10%**	0.44 ± 0.00	0.31 ± 0.01	0.48 ± 0.00	0.81 ± 0.00	0.82 ± 0.00
	1000	0.49 ± 0.01	0.31 ± 0.02	0.47 ± 0.01	0.81 ± 0.00	0.80 ± 0.00
	All	0.50 ± 0.00	0.32 ± 0.03	0.48 ± 0.00	0.82 ± 0.01	0.81 ± 0.01
GIN	**10%**	0.70 ± 0.00	0.72 ± 0.00	0.86 ± 0.00	0.88 ± 0.00	0.93 ± 0.00
	1000	0.72 ± 0.02	0.73 ± 0.02	0.86 ± 0.00	0.89 ± 0.00	0.91 ± 0.03
	All	0.76 ± 0.00	0.71 ± 0.00	0.86 ± 0.00	0.89 ± 0.00	0.93 ± 0.00
SAGE	**10%**	0.71 ± 0.01	0.70 ± 0.00	0.87 ± 0.00	0.93 ± 0.00	0.94 ± 0.00
	1000	0.73 ± 0.03	0.71 ± 0.00	0.87 ± 0.00	0.92 ± 0.00	0.93 ± 0.01
	All	0.74 ± 0.00	0.72 ± 0.00	0.87 ± 0.00	0.92 ± 0.00	0.94 ± 0.00

6.4 Effectiveness of LBAggr

To validate the effectiveness of the LBAggr method proposed in Section 5, we compare with MeanAggr and MajAggr by conducting experiments on five datasets and four GNN models. Table 3 illustrates the F1 scores of different aggregation methods for Scratch, GraphEraser-BLPA, and GraphEraser-BEKM.

Observations. In general, the LBAggr method can effectively improve the F1 score in most cases compared to MeanAggr and MajAggr. For instance, on the Cora dataset with GraphEraser-BLPA unlearning method, LBAggr achieves 0.357 higher F1 score than that of MajAggr for the GCN model. We also observe that the MajAggr method performs the worst in most cases. We posit it is because MajAggr neglects information of the posteriors obtained from each shard model. Concretely, if the posteriors of the shard models have high confidence to multiple classes rather than a single class, the MajAggr method will lose information about the runner-up classes, leading to bad model utility.

Comparing different GNN models, GCN benefits the most while GIN benefits the least from LBAggr. In terms of model utility, the GraphEraser-BLPA method benefits the most from LBAggr. We conjecture this is because the BLPA partition method can capture the local structural information while losing some of the global structural information of the training graph [57, 64]. Using LBAggr helps better capture the global structural information by assigning different importance scores to shard models.

Impact of the Number of Training Nodes. As discussed in Section 5, to further improve the unlearning efficiency, one can use a small portion of nodes in the training graph to learn the importance score. Doing this can effectively reduce the relearning time of LBAggr, as shown in Section 6.2. Here we evaluate its impact on the model utility. We experiment on three different cases: randomly sample 10% of nodes, randomly sample a fixed number of 1,000 nodes, and use all nodes, in the training graph.

Table 5 illustrates the results on five datasets and four GNN models for GraphEraser-BEKM. We observe that both using 10% of nodes and using a fixed number of 1,000 nodes can achieve comparable model utility as that of using all nodes. In practice, we suggest the model provider to adopt the minimum of 10% and 1,000 to learn the importance scores. In another word, the model provider can use 10% for small graphs, and use 1,000 for large graphs. The conclusions are the same for GraphEraser-BLPA.

6.5 Comparison with Existing Balanced Graph Partition Solutions

Note that there are existing solutions for balanced graph partitioning [39, 43, 60]. In this section, we empirically compare with them in terms of running time and model utility.

Competitors. These algorithms can be broadly classified into three categories: The first category considers only the graph structural information and relies on community detection as GraphEraser-BLPA. The second category also considers only the graph structural information but without relying on community detection. The third category considers both structural information and node features as GraphEraser-BEKM. For each category, we choose one most representative method as competitor, and we list the details of them as follows.

- **BLPA-LP [60].** Similar to our proposed GraphEraser-BLPA, this method achieves a balanced graph partition by constraining the label propagation process. The general idea of BLPA-LP is to formulate the label propagation process as a linear programming problem with $2k(k-1)$ variables and $2k^2 + nk(k-1)$ constraints, where n and k are the number of nodes and the number of shards, respectively. When the size of the graph and the number of shards are large, solving the linear programming problem is time-consuming.
- **METIS [39].** The objective of METIS is to obtain the balanced graph partition while cutting the minimum number of edges. The computational complexity of METIS is $O((n+m) \cdot \log k)$, where m is the number of edges. We implement this method with official METIS 5.1.0[2] and a Python wrapper[3] for METIS library.
- **BEKM-Hungarian [43].** BEKM-Hungarian shares the general idea of our GraphEraser-BEKM. The main difference is that it has a different mechanism in the node reassignment step for achieving balanced k-means. Concretely, BEKM-Hungarian formulates the node reassignment step as a matching problem and is approximately solved by the Hungarian algorithm. The computational complexity of the Hungarian algorithm is $O(n^3)$.

The reason why we choose these three algorithms is that they achieve the-state-of-the-art performance for each category. We discuss other existing balanced graph partitioning algorithms and how these three algorithms fit into the whole balanced graph partitioning field in Section 8.

Results. Table 6 and Table 7 illustrate the model utility and graph partitioning efficiency for different methods. For a fair comparison, we apply LBAggr for all the graph partitioning methods.

In general, we observe that the graph partitioning methods rely on both graph structural information and node features. i.e., GraphEraser-BEKM and BEKM-Hungarian, achieve the best model utility when the target model is GAT, GIN, and SAGE, which is consistent with the conclusion of Section 6.3. Comparing GraphEraser-BEKM and BEKM-Hungarian, we observe that they achieve similar model utility; however, the computational complexity of BEKM-Hungarian ($O(n^3)$) is much higher than that of GraphEraser-BEKM ($O(k \cdot n)$). From Table 7, we can see that BEKM-Hungarian is not scalable to large graphs.

When the target model is GCN, the community detection based methods, i.e., GraphEraser-BLPA and BLPA-LP, achieve better model utility than the minimum-cut based method (METIS). We suspect this is because the GCN model requires the node degree information for normalization, and the community detection based methods can preserve more local structural information thus better preserve the node degree. Comparing GraphEraser-BLPA and BLPA-LP, GraphEraser-BLPA is more computationally efficient than BLPA-LP (see Table 7) while achieving comparable model utility.

Remarks. GraphEraser is a general framework for GNN unlearning; any balanced graph partitioning method which meets the requirements in Section 3.1 can be considered. Therefore, we encourage the research community to develop more efficient and better performing balanced graph partitioning algorithms for the graph unlearning application.

6.6 Unlearning Power of GraphEraser

Since our method is highly empirical, we adopt the state-of-the-art attack against machine unlearning [15] to quantify the extra information leakage of GraphEraser when the graph is not repartitioned. In particular, Chen et al. [15] showed that the attackers, using an enhanced membership inference attack [58], can determine whether a target sample exists in the original model and is revoked from the unlearned model when they have access to both the original model and unlearned model. Here we quantify the extra information leakage of GraphEraser as *the attack's performance difference between deterministic unlearning and* GraphEraser *unlearning*. Concretely, we introduce two scenarios of membership inference attacks. We start from the same set of original shard models. In scenario 1, the unlearned models are obtained by directly deleting the revoked nodes from the corresponding shard graph, and retrain the corresponding shard models. This is how GraphEraser generates the unlearned models. In scenario 2, we retrain from scratch (re-partition the graph, and train a set of new shard models). This type of unlearning deterministically unlearn every component while it is extremely time-consuming. Denoting the two scenarios as $\mathcal{A}_I$ and $\mathcal{A}_{II}$, the extra information leakage is the difference of the attack AUC between $\mathcal{A}_I$ and $\mathcal{A}_{II}$. We use the implementation[4] of [15] to conduct our experiments. The experimental results in Table 8 show that the attack AUC of both $\mathcal{A}_I$ and $\mathcal{A}_{II}$ are close to 0.5 (random guessing), meaning that GraphEraser does not leak much extra information. This is also consistent with the observation of [15] that the membership inference performs bad on SISA based method due to the fact that the aggregation reduces the influence of a specific sample on its global model.

[2] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[3] https://github.com/inducer/pymetis

[4] https://github.com/MinChen00/UnlearningLeaks

Table 6: Comparison of F1 scores for different graph partition methods. We highlight our proposed method in the red ground and the best results in bold.

Dataset $\mathcal{D}$	Model $\mathcal{F}$	BLPA-based GraphEraser-BLPA	 BLPA-LP	BEKM-based GraphEraser-BEKM	 BEKM-Hungarian	Minimum Edge Cut METIS
Cora	**GAT**	0.727 ± 0.009	0.712 ± 0.006	**0.754 ± 0.009**	0.740 ± 0.006	0.683 ± 0.007
	GCN	**0.676 ± 0.004**	0.668 ± 0.020	0.531 ± 0.009	0.552 ± 0.005	0.458 ± 0.010
	GIN	0.753 ± 0.015	0.722 ± 0.029	**0.801 ± 0.018**	0.795 ± 0.016	0.703 ± 0.020
	SAGE	0.684 ± 0.014	0.708 ± 0.002	**0.740 ± 0.013**	0.739 ± 0.005	0.694 ± 0.008
Citeseer	**GAT**	0.688 ± 0.005	0.590 ± 0.009	**0.738 ± 0.006**	0.737 ± 0.003	0.615 ± 0.002
	GCN	**0.516 ± 0.004**	0.504 ± 0.022	0.417 ± 0.018	0.397 ± 0.023	0.457 ± 0.006
	GIN	0.597 ± 0.021	0.589 ± 0.041	**0.678 ± 0.072**	0.655 ± 0.059	0.574 ± 0.064
	SAGE	0.642 ± 0.005	0.682 ± 0.007	**0.743 ± 0.002**	0.734 ± 0.002	0.677 ± 0.004
Pubmed	**GAT**	0.858 ± 0.003	0.857 ± 0.001	**0.860 ± 0.003**	0.857 ± 0.003	0.841 ± 0.001
	GCN	**0.718 ± 0.010**	0.709 ± 0.004	0.659 ± 0.020	0.628 ± 0.034	0.650 ± 0.018
	GIN	0.855 ± 0.004	0.854 ± 0.001	**0.859 ± 0.003**	0.853 ± 0.001	0.836 ± 0.001
	SAGE	0.863 ± 0.002	0.857 ± 0.003	**0.862 ± 0.002**	0.858 ± 0.00	0.849 ± 0.003
CS	**GAT**	0.858 ± 0.004	0.862 ± 0.003	**0.906 ± 0.002**	0.901 ± 0.003	0.891 ± 0.013
	GCN	0.750 ± 0.023	0.745 ± 0.004	**0.812 ± 0.012**	0.806 ± 0.007	0.782 ± 0.021
	GIN	0.789 ± 0.013	0.786 ± 0.003	**0.891 ± 0.002**	0.883 ± 0.007	0.862 ± 0.002
	SAGE	0.886 ± 0.010	0.889 ± 0.023	**0.927 ± 0.002**	0.922 ± 0.002	0.906 ± 0.004
Physics	**GAT**	0.921 ± 0.004	0.918 ± 0.004	**0.925 ± 0.001**	0.923 ± 0.001	0.918 ± 0.002
	GCN	**0.858 ± 0.008**	0.856 ± 0.005	0.815 ± 0.001	0.808 ± 0.001	0.810 ± 0.001
	GIN	0.907 ± 0.003	0.897 ± 0.011	**0.926 ± 0.001**	0.923 ± 0.002	0.895 ± 0.003
	SAGE	0.922 ± 0.001	0.913 ± 0.002	**0.933 ± 0.001**	0.931 ± 0.001	0.911 ± 0.005

Table 7: Comparison of graph partition efficiency for different balanced graph partition methods. We highlight our proposed partition methods in the red ground.

Dataset $\mathcal{D}$	BLPA-based GraphEraser	 LP	BEKM-based GraphEraser	 Hungarian	Minimum Edge Cut METIS
Cora	**3s**	179s	**26s**	817s	4s
Citeseer	**2s**	30s	**20s**	1,309s	3s
Pubmed	**20s**	301s	**240s**	174,684s	21s
CS	**13s**	705s	**220s**	174,498s	15s
Physics	**40s**	2,351s	**480s**	948,790s	58s

Table 8: Attack AUC of membership inference on GraphEraser**.**

Model Dataset	GAT $\mathcal{A}_I$	 $\mathcal{A}_{II}$	GCN $\mathcal{A}_I$	 $\mathcal{A}_{II}$	GIN $\mathcal{A}_I$	 $\mathcal{A}_{II}$	SAGE $\mathcal{A}_I$	 $\mathcal{A}_{II}$
Cora	0.512	0.508	0.511	0.510	0.513	0.510	0.511	0.510
Citeseer	0.515	0.510	0.510	0.510	0.513	0.513	0.512	0.510
Pubmed	0.509	0.510	0.511	0.509	0.512	0.511	0.510	0.511
CS	0.510	0.509	0.520	0.511	0.515	0.514	0.515	0.513
Physics	0.519	0.515	0.518	0.512	0.512	0.510	0.517	0.517

7 DISCUSSION

7.1 Analysis of GraphEraser

Guarantee to the right to be forgotten. Note that when the adversaries have access to both the original and the unlearned models, the presence of the deleted node might be inferred using the friendship information of the graph. A previous study [15] has shown that machine unlearning is vulnerable to membership inference attack. However, these attacks are orthogonal to our work since the primary goal of machine unlearning is to comply with "legitimate regulations" such as the GDPR. In this sense, as long as the model is trained without the revoked sample, the requirement of the right to be forgotten is satisfied. To mitigate the potential attacks, one can deploy some defense mechanisms as discussed in [15], which can be add-ons of GraphEraser.

Compatibility with Commercial Graph Services. Compared with the existing graph-learning-based services, the additional cost of GraphEraser is graph partition; however, once the partition is defined, we can keep it fixed without extra effort. The process of training shard models is the same as the existing services. Once this pipeline is built, the maintenance effort of dealing with unlearning requests is much lower than existing services, since GraphEraser only needs to retrain the sub-model containing the deleted samples.

Additional Cost of Maintaining Large-scale Shards. One might argue that maintaining the shard models is more expensive than maintaining one global model. However, in machine unlearning, the cost of retraining the global model is much higher than maintaining the shard models. Comparing to maintaining one global model, the additional cost of maintaining the shard models comes from two sources: (1) Additional prediction time cost due to the aggregation process; (2) Additional storage cost of storing multiple shard models instead of storing one global model. From the time cost perspective, we have empirically shown in Section 6.2 that the additional prediction time introduced by the shard-based methods is much less than the retraining time of the global model. From the economic cost perspective, it is well-known that the computation cost (of retraining the global model) is much higher than the storage cost (of renting disk for storing the shard models) [11]. For instance, the storage costs are of \$0.026/*GB* per month on Google Cloud, \$0.023/*GB* per month on Amazon Web Services, and \$0.018/*GB* per month on Azure at the time of writing. Instead, renting the cheapest GPUs starts at \$0.35/*hour* on Google Cloud, \$0.526/*hour* on Amazon Web Services, and \$0.90/*hour* on Azure.

Adaptive Machine Unlearning. The authors in [29] define the notion of (α, β, γ)-unlearning, which enforces that the output of any unlearning algorithm should be similar to that of retraining from scratch. The authors prove that the general family of distributed learning and unlearning algorithms such as SISA method satisfies (α, β, γ)-unlearning in the *non-adaptive* setting (unlearning requests arrive in a non-adaptive way), but it does not satisfy (α, β, γ)-unlearning in the *adaptive* setting. As GraphEraser belongs to this general family, GraphEraser also satisfies (α, β, γ)-unlearning in the non-adaptive setting, but does not satisfy the adaptive setting.

7.2 Handling Different Scenarios

Community-Dependent Removal Requests. GraphEraser can deal with the scenario where unlearning requests come from specific types of communities, since it can be seen as the shard-size-constraint version of community detection algorithms, and nodes in specific types of communities tend to be allocated into the same shards. When the community-dependent requests come, only a few shard models need to be retrained.

To further validate the robustness of GraphEraser, we use the vanilla LPA method (without community size constraints) to partition the graph. We then randomly choose a community detected by vanilla LPA, and gradually delete the corresponding nodes (with the same IDs in the selected community) from the shard models of GraphEraser. We observe that deleting nodes from a single community does not significantly affect the model utility of GraphEraser. Due to space limitation, we defer the results to Appendix J.

Node Insertion Scenarios. In real-world applications, there are likely nodes to be inserted into the training graph of the GNN model. In this case, we can insert the node to the shard containing the highest number of its neighbors and retrain the corresponding shard model.

8 RELATED WORK

Machine Unlearning. The notion of machine unlearning was first proposed by Cao et al. [13]. Subsequently, the research in machine unlearning has proceeded into two directions: Deterministic unlearning and approximate unlearning. The objective of *deterministic unlearning* (some papers call it *exact unlearning*) is to guarantee that the influence of the revoked samples are completely removed from the target model. The straightforward approach of retraining the global model from scratch perfectly satisfies deterministic unlearning; however, it is computationally infeasible when the dataset is large. To reduce the computational cost of retraining from scratch, Cao et al. [13] consider statistical query learning and dissect the model into a summation form, so that removing a sample can be done efficiently by subtracting the summand corresponds to that sample. However, the algorithm in [13] only applies to learning algorithms that can be transformed into summation form, limiting itself not for neural networks.

Recently, Ginart et al. [21] have proposed the notion of (ϵ, δ)-approximate unlearning in a way reminiscent of DP. It guarantees that the output distribution of the unlearned model is *close* to the model trained without the revoked samples. Formally, an unlearning algorithm U_A satisfies (ϵ, δ)-approximate unlearning if $\Pr[A(D_{-i}) \in S | D_{-i}] \le \epsilon \cdot \Pr[U_A(D, A(D), i) \in S | D_{-i}] + \delta$, where A is the learning algorithm, D is the training dataset, S is the possible output of the model, and i is the revoked sample. In this sense, DP is a natural choice to support (ϵ, δ)-approximate unlearning. However, when there are a group of samples to be deleted, we would need to use group DP, which greatly increases the amount of noise needed and decreases the model utility; thus, DP is not directly adopted to implement approximate unlearning [21]. Ginart et al. [21] proposed a (ϵ, δ)-approximate unlearning algorithm for the k-means problem. Guo et al. [28] gave approximate unlearning algorithms for linear and logistic regression. It first performs a convex optimization step and is followed by a Gaussian perturbation. The algorithm yields error that grows linearly with the number of updates. Izzo et al. [34] focus on linear regression and show how to improve the run-time per deletion of [28] from quadratic to linear in the dimension. Neel et al. [44] leverages a distributed optimization that partitions the data, separately optimizes on each partition, and then averages the parameters. It guarantees that, for a fixed accuracy target, the run-time of the update operation is constant in the length of the update sequence, and it can deal with all convex models.

Due to the strong theoretical requirement of (ϵ, δ)-approximate unlearning, most of previous studies can only deal with linear or convex models. Note that GNN is a highly non-convex model, which makes it difficult to theoretically prove that GraphEraser satisfies (ϵ, δ)-approximate unlearning; thus, we empirically quantify the possible information leakage relying on membership inference.

Balanced Graph Partitioning. As discussed in Section 6.5, the existing balanced graph partitioning algorithms can be broadly classified into three categories. The first two categories adopt **Strategy 1** in Section 4 that only consider graph structural information. The third category adopt **Strategy 2** in Section 4 that consider both graph structural and node feature information.

As discussed in Section 4.1, community detection can inherently preserve graph structural information with the cost of unbalanced partitioning. Thus, the first category of previous studies aim to modify the existing community detection methods to satisfy balanced community size constraint. In BLPA-LP [60], the authors propose to modify the LPA algorithm to satisfy community size constraints by formulating the label propagation process as a linear programming problem.

On the other hand, the second category of previous studies do not rely on community detection. Instead, they directly partition the graph by optimizing some predefined criterion, such as minimizing the graph cut [18, 56] or maximizing the graph modularity [25, 45]. However, these optimization problems are always NP-hard and cannot be solved exactly; thus, the researchers proposed lots of approximate or intuitive algorithms. Spectral graph partitioning [48, 51] is a widely adopted approach. The general idea is to first calculate the Laplacian matrix of the graph, then calculate the eigenvectors of the Laplacian matrix. Each node is mapped to one of the eigenvalues in the second smallest eigenvector, and the graph partition is defined by the sign of the corresponding eigenvalues. To partition the graph into multiple shards, one can conduct the spectral graph partitioning method in a hierarchical manner. The main drawback of the spectral methods is they cannot deal with large-scale graphs. A promising solution for partitioning large-scale graphs is utilizing the multilevel graph partitioning methods. The general idea is first to contract edges and obtain smaller graphs, then cut the resulting graph, and finally unfold back to the original graph with some local improvement criterion [5, 7, 32, 33]. Among the multilevel graph partitioning methods, METIS [36, 39] is a family of the most widely known methods and achieves the state-of-the-art performance [8].

The general idea of the third category is first to transform the attributed graph into node embeddings and use balanced clustering methods to cluster the node embeddings. In BEKM-Hungarian [43], the authors modify the reassignment step of k-means algorithm to achieve balanced clusters. The core idea is to formulate the node reassignment problem as a matching problem which is approximately solved by the Hungarian algorithm. In [41], the authors propose to use linear regression to estimate the class-specific hyperplanes that partition each class of the data point from others. A soft balanced constraint is enforced to achieve balanced clustering. The drawback of this method is that we cannot precisely control the cluster size.

9 CONCLUSION

In this paper, we propose the first machine unlearning framework GraphEraser in the context of GNNs. Concretely, we first identify two types of machine unlearning requests, namely node unlearning and edge unlearning. We then propose a general pipeline for machine unlearning in GNN models. To achieve efficient retraining while keeping the structural information of the graph, we propose a general principle for balancing the shards resulting from the graph partitioning and instantiate it with two novel balanced graph partition algorithms. We further propose a learning-based aggregation method to improve the model utility. Extensive experiments on five real-world graph datasets and four state-of-the-art GNN models illustrate the high unlearning efficiency and high model utility resulting from GraphEraser.

ACKNOWLEDGMENTS

We thank all anonymous reviewers for their constructive comments. This work is partially funded by the Helmholtz Association within the project "Trustworthy Federated Data Analytics" (TFDA) (funding number ZT-I-OO1 4). Tianhao Wang is funded by National Science Foundation (NSF) under Grand No.1931443, a Bilsland Dissertation Fellowship, and a Packard Fellowship.

REFERENCES

[1] https://gdpr-info.eu/.
[2] https://oag.ca.gov/privacy/ccpa.
[3] https://laws-lois.justice.gc.ca/ENG/ACTS/P-8.6/index.html.
[4] https://iapp.org/media/pdf/resource_center/Brazilian_General_Data_Protection_Law.pdf.
[5] R. Andersen, F. R. K. Chung, and K. J. Lang. Local Graph Partitioning using PageRank Vectors. In *Annual Symposium on Foundations of Computer Science (FOCS)*, pages 475–486. IEEE, 2006.
[6] J. Atwood and D. Towsley. Diffusion-Convolutional Neural Networks. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1993–2001. NIPS, 2016.
[7] D. Avdiukhin, S. Pupyrev, and G. Yaroslavtsev. Multi-Dimensional Balanced Graph Partitioning via Projected Gradient Descent. *Proceedings of the VLDB Endowment*, 2019.
[8] A. Awadelkarim and J. Ugander. Prioritized Restreaming Algorithms for Balanced Graph Partitioning. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1877–1887. ACM, 2020.
[9] T. Baumhauer, P. Schöttle, and M. Zeppelzauer. Machine Unlearning: Linear Filtration for Logit-based Classifier. *CoRR abs/2002.02730*, 2020.
[10] T. Bertram, E. Bursztein, S. Caro, H. Chao, R. Chin, F. Fleischer, A. Gustafsson, J. Hemerly, C. Hibbert, L. InvernizziLanah, K. Donnelly, J. Ketover, J. Laefer, P. Nicholas, Y. Niu, H. Obhi, D. Price, A. Strait, K. Thomas, and A. Verney. Five Years of the Right to be Forgotten. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 959–972. ACM, 2019.
[11] L. Bourtoule, V. Chandrasekaran, C. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot. Machine Unlearning. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2021.
[12] J. Brophy and D. Lowd. Machine Unlearning for Random Forests. In *International Conference on Machine Learning (ICML)*, pages 1092–1104. PMLR, 2021.
[13] Y. Cao and J. Yang. Towards Making Systems Forget with Machine Unlearning. In *IEEE Symposium on Security and Privacy (S&P)*, pages 463–480. IEEE, 2015.
[14] Y. Cao, A. F. Yu, A. Aday, E. Stahl, J. Merwine, and J. Yang. Efficient Repair of Polluted Machine Learning Systems via Causal Unlearning. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 735–747. ACM, 2018.
[15] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang. When Machine Unlearning Jeopardizes Privacy. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 896–911. ACM, 2021.
[16] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 257–266. ACM, 2019.
[17] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3837–3845. NIPS, 2016.
[18] D. Delling, A. V. Goldberg, I. P. Razenshteyn, and R. F. F. Werneck. Graph Partitioning with Natural Cuts. In *International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1135–1146. IEEE, 2011.
[19] Z. Diao, X. Wang, D. Zhang, Y. Liu, K. Xie, and S. He. Dynamic Spatial-Temporal Graph Convolutional Neural Networks for Traffic Forecasting. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 890–897. AAAI, 2019.
[20] D. L. Felps, A. D. Schwickerath, J. D. Williams, T. N. Vuong, A. Briggs, M. Hunt, E. Sakmar, D. D. Saranchak, and T. Shumaker. Class Clown: Data Redaction in Machine Unlearning at Enterprise Scale. *CoRR abs/2012.04699*, 2020.
[21] A. A. Ginart, M. Y. Guan, G. Valiant, and J. Zou. Making AI Forget You: Data Deletion in Machine Learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 3513–3526. NeurIPS, 2019.
[22] M. Girvan and M. E. J. Newman. Community Structure in Social and Biological Networks. *Proceedings of the National Academy of Sciences*, 2002.
[23] A. Golatkar, A. Achille, and S. Soatto. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9301–9309. IEEE, 2020.
[24] A. Golatkar, A. Achille, and S. Soatto. Forgetting Outside the Box: Scrubbing Deep Networks of Information Accessible from Input-Output Observations. In *European Conference on Computer Vision (ECCV)*, pages 383–398. Springer, 2020.
[25] B. H. Good, Y.-A. D. Montjoye, and A. Clauset. Performance of Modularity Maximization in Practical Contexts. *Physical Review E*, 2010.
[26] S. Gregory. Finding Overlapping Communities in Networks by Label Propagation. *New Journal of Physics*, 2010.
[27] A. Grover and J. Leskovec. node2vec: Scalable Feature Learning for Networks. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 855–864. ACM, 2016.
[28] C. Guo, T. Goldstein, A. Y. Hannun, and L. van der Maaten. Certified Data Removal from Machine Learning Models. In *International Conference on Machine Learning (ICML)*, pages 3832–3842. PMLR, 2020.
[29] V. Gupta, C. Jung, S. Neel, A. Roth, S. Sharifi-Malvajerdi, and C. Waites. Adaptive Machine Unlearning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2021.
[30] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1025–1035. NIPS, 2017.
[31] S. He, F. Bastani, S. Jagwani, E. Park, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, S. Madden, and M. A. Sadeghi. RoadTagger: Robust Road Attribute Inference with Graph Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 10965–10972. AAAI, 2020.
[32] T. Heuer, P. Sanders, and S. Schlag. Network Flow-Based Refinement for Multilevel Hypergraph Partitioning. In *International Symposium on Experimental Algorithms (SEA)*, pages 1:1–1:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
[33] S. Huang, Y. Li, Z. Bao, and Z. Li. Towards Efficient Motif-based Graph Partitioning: An Adaptive Sampling approach. In *IEEE International Conference on Data Engineering (ICDE)*, pages 528–539. IEEE, 2021.
[34] Z. Izzo, M. A. Smart, K. Chaudhuri, and J. Zou. Approximate Data Deletion from Machine Learning Models: Algorithms and Evaluations. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2008–2016. PMLR, 2021.
[35] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.
[36] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.
[37] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley. Molecular Graph Convolutions: Moving Beyond Fingerprints. *Journal of Computer-Aided Molecular*

Design, 2016.

[38] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[39] D. LaSalle and G. Karypis. A Parallel Hill-Climbing Refinement Algorithm for Graph Partitioning. In *International Conference on Parallel Processing (ICCP)*, pages 236–241. IEEE Computer Society, 2016. http://glaros.dtc.umn.edu/gkhome/views/metis.

[40] X. Li, J. Saúde, P. Reddy, and M. Veloso. Classifying and Understanding Financial Data Using Graph Neural Network. In *The AAAI Workshop on Knowledge Discovery from Unstructured Data in Financial Services (KDF)*. AAAI, 2020.

[41] H. Liu, J. Han, F. Nie, and X. Li. Balanced Clustering with Least Square Regression. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2231–2237. AAAI, 2017.

[42] Y. Liu, Z. Ma, X. Liu, J. Liu, Z. Jiang, J. Ma, P. Yu, and K. Ren. Learn to Forget: Memorization Elimination for Neural Networks. *CoRR abs/2003.10933*, 2020.

[43] M. I. Malinen and P. Fränti. Balanced K-Means for Clustering. In P. Fränti, G. Brown, M. Loog, F. Escolano, and M. Pelillo, editors, *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2014, Joensuu, Finland, August 20-22, 2014. Proceedings*, volume 8621 of *Lecture Notes in Computer Science*, pages 32–41. Springer, 2014.

[44] S. Neel, A. Roth, and S. Sharifi-Malvajerdi. Descent-to-Delete: Gradient-Based Methods for Machine Unlearning. In *International Conference on Algorithmic Learning Theory (ICALT)*, pages 931–962. PMLR, 2021.

[45] M. E. Newman. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Sciences (PNAS)*, 103(23):8577–8582, 2006.

[46] A. Pal, C. Eksombatchai, Y. Zhou, B. Zhao, C. Rosenberg, and J. Leskovec. PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2311–2320. ACM, 2020.

[47] T. Pham, T. Tran, D. Q. Phung, and S. Venkatesh. Column Networks for Collective Classification. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2485–2491. AAAI, 2017.

[48] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM Journal on Matrix Analysis and Applications (SIMAX)*, 11(3):430–452, 1990.

[49] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang. DeepInf: Social Influence Prediction with Deep Learning. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2110–2119. ACM, 2018.

[50] U. N. Raghavan, R. Albert, and S. Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks. *Physical Review E*, 2007.

[51] M. A. Riolo and M. E. J. Newman. First-principles Multiway Spectral Partitioning of Graphs. *J. Complex Networks*, 2(2):121–140, 2014.

[52] M. Rosvall and C. T. Bergstrom. Maps of Random Walks on Complex Networks Reveal Community Structure. *Proceedings of the National Academy of Sciences*, 2008.

[53] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 2009.

[54] S. Schelter. "Amnesia" - Towards Machine Learning Models That Can Forget User Data Very Fast. In *Annual Conference on Innovative Data Systems Research (CIDR)*, 2020.

[55] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of Graph Neural Network Evaluation. *CoRR abs/1811.05868*, 2018.

[56] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[57] Y. Shi, Z. Huang, W. Wang, H. Zhong, S. Feng, and Y. Sun. Masked Label Prediction: Unified Massage Passing Model for Semi-Supervised Classification. *CoRR abs/2009.03509*, 2020.

[58] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy (S&P)*, pages 3–18. IEEE, 2017.

[59] W. Torng and R. B. Altman. Graph Convolutional Neural Networks for Predicting Drug-Target Interactions. *Journal of Chemical Information and Modeling*, 2019.

[60] J. Ugander and L. Backstrom. Balanced Label Propagation for Partitioning Massive Graphs. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 507–516. ACM, 2013.

[61] L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 2008.

[62] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[63] F. Wang and C. Zhang. Label Propagation through Linear Neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 2008.

[64] H. Wang and J. Leskovec. Unifying Graph Convolutional Neural Networks and Label Propagation. *CoRR abs/2002.06755*, 2020.

[65] X. Wei, L. Xu, B. Cao, and P. Yu. Cross View Link Prediction by Learning Noise-resilient Representation Consensus. In *International Conference on World Wide Web (WWW)*, pages 1611–1619. ACM, 2017.

[66] J. Wu, J. He, and J. Xu. DEMO-Net: Degree-specific Graph Neural Networks for Node and Graph Classification. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 406–415. ACM, 2019.

[67] Y. Wu, E. Dobriban, and S. B. Davidson. DeltaGrad: Rapid Retraining of Machine Learning Models. In *International Conference on Machine Learning (ICML)*, pages 10355–10366. PMLR, 2020.

[68] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[69] C. Yang, A. Pal, A. Zhai, N. Pancha, J. Han, C. Rosenberg, and J. Leskovec. MultiSage: Empowering GCN with Contextualized Multi-Embeddings on Web-Scale Multipartite Networks. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2434–2443. ACM, 2020.

[70] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *International Conference on Machine Learning (ICML)*, pages 40–48. JMLR, 2016.

[71] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 974–983. ACM, 2018.

[72] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations (ICLR)*, 2020.

[73] M. Zhang and Y. Chen. Weisfeiler-Lehman Neural Machine for Link Prediction. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 575–583. ACM, 2017.

[74] X. Zhang and M. E. J. Newman. Multiway Spectral Community Detection in Networks. *Physical Review E*, 2015.

[75] J. Zhou, C. Chen, L. Zheng, X. Zheng, B. Wu, Z. Liu, and L. Wang. Privacy-Preserving Graph Neural Network for Node Classification. *CoRR abs/2005.11903*, 2020.

A NOTATIONS

The frequently used notations in this paper are summarized in Table 9.

Table 9: Summary of the notations used in this paper.

Notation	Description
$\mathcal{G} = \langle \mathcal{V}, A, X \rangle$	Graph
$u, v \in \mathcal{V}$	Nodes in $\mathcal{G}$
$e_{u,v}$	Edge that connects u and v
A	Adjacency matrix of $\mathcal{G}$
X	Attributes associated with $\mathcal{V}$
$\mathcal{N}_u$	Neighborhood nodes of u
E_u	Node embedding of u
d_X / d_E	Dimension of attributes / embeddings
Φ	Aggregation operation in message passing
Ψ	Updating operation in message passing
$\mathbf{m}$	Message received from neighbors

B DETAILS OF GRAPH NEURAL NETWORKS

Aggregation Operations. We introduce four of the most widely used aggregation operations as follows:

- **Graph Isomorphism Networks (GIN) [68].** GIN directly sums up the embeddings of u's neighbors $\mathcal{N}_u$, i.e., $\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} E_v$.
- **Graph SAmple and aggreGatE (SAGE) [30].** The SAGE method takes an average over u's neighbors' embeddings rather than summing them up, i.e., $\mathbf{m}_{\mathcal{N}_u} = \frac{\sum_{v \in \mathcal{N}_u} E_v}{|\mathcal{N}_u|}$.
- **Graph Convolution Networks (GCN) [38].** The GCN method uses the symmetric normalization for aggregation, i.e., $\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} \frac{E_v}{\sqrt{|\mathcal{N}_u| \cdot |\mathcal{N}_v|}}$.
- **Graph Attention Networks (GAT) [62].** GAT assigns an attention weight or importance score to each neighbor during the aggregation, i.e., $\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} \alpha_{u,v} E_v$, where the attention weight $\alpha_{u,v}$ is defined as follows:

$$\alpha_{u,v} = \frac{exp(a^T [WE_u || WE_v])}{\sum_{v' \in \mathcal{N}_u} exp(a^T [WE_u || WE'_v])}$$

  Here, a is a learnable attention vector, W is a learnable matrix, and $||$ denotes the concatenation operation.

Updating Operation. We introduce three popular updating operations.

- **Linear Combination [53].** The most basic updating method is to calculate the linear combinations, i.e.,

$$\Psi_{linear} = \sigma(W_{self} E_u + W_{neigh} \mathbf{m}_{\mathcal{N}_u})$$

  where W_{self} and W_{neigh} are learned during the training process and σ is a non-linear activation function. The main issue of the basic method is over-smoothing. That is the embeddings of all nodes would be similar to each other after several steps of aggregation.
- **Concatenation [30].** One approach to handle the over-smoothing issue is to concatenate the result of the linear combination method with the current node embedding, i.e., $\Psi_{concat} = \Psi_{linear} || E_u$.
- **Interpolation [47].** Another method is to use the weighted average of the linear combination method and the current embedding for updating, i.e., $\Psi_{inter} = \alpha_1 \circ \Psi_{linear} + \alpha_2 \circ E_u$.

Implementation of GNN Models. Typically, each step of message passing is referred to as a *GNN module*, and a GNN model can be implemented by stacking multiple layers of the GNN module and one layer of the softmax module for node classification. We denote a GNN model by $\mathcal{F}$, which can take as input the feature matrix X and the adjacency matrix A of a set of nodes $\mathcal{V}$, and output a posterior matrix P. Here each row of P is the *posterior* of one node $u \in \mathcal{V}$, which is a vector of entries indicating the probability of node u belonging to a certain class. All values in each row of P sum up to 1 by definition.

C VISUALIZATION OF CLASSICAL LPA

Figure 5 shows the visualization of shards detected by classical LPA on the Cora dataset, where different colors stands for different shards. We use the red box and blue box to mark a large shard and a small shard. A clear unequal size of the two shards can be observed.

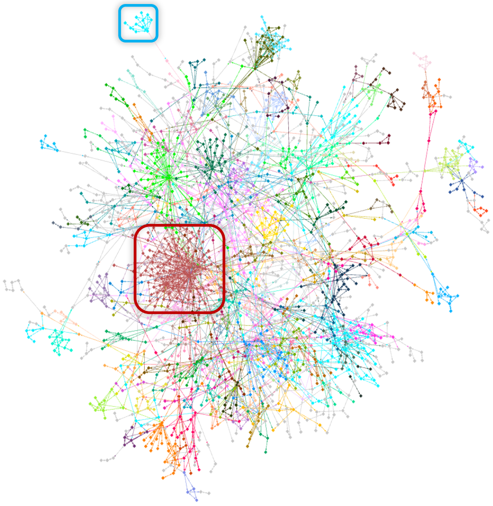
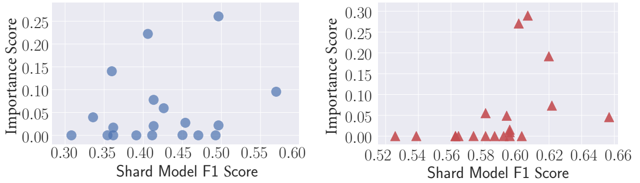


Figure 5: Visualization of imbalanced shards detected by classical LPA on the Cora dataset. Different colors stand for different shards, where the red box and blue box mark a large shard and a small shard, respectively.

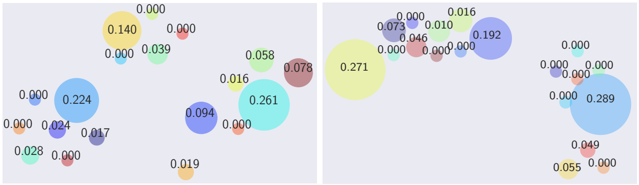
D CONVERGENCE ANALYSIS

As discussed in Section 4, it is difficult to theoretically prove the convergence of both BLPA and BEKM. In this section, we conduct empirical experiments to validate the convergence performance of both algorithms. An algorithm converges when the nodes of different shards between two consecutive iterations do not move. Figure 6 illustrates the ratio of moved nodes between different shards in each iteration. The experimental results show that the ratio of moved nodes gradually approximates to zero within 30 iterations for both algorithms on all five datasets. Therefore, we set the number of iterations T to 30 for all of our experiments.



(a) GraphEraser-BLPA **(b)** GraphEraser-BEKM

Figure 7: Correlation between the importance score of a shard model and its F1 score on the Cora dataset. The x-axis stands for the shard model's F1 score, and the y-axis stands for the importance score of that shard. We report the results of GAT model with GraphEraser-BLPA **and** GraphEraser-BEKM **unlearning methods.**

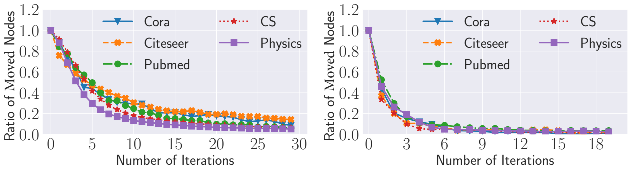


(a) BLPA **(b)** BEKM

Figure 8: The t-SNE plot of shard embeddings for the Cora dataset. Each circle represents the mean node embeddings of a shard, where the circle size is proportional to its importance score in annotations.

Table 10: Comparison of F1 scores of all graph unlearning methods for edge unlearning. BLPA **and** BEKM **stand for** GraphEraser-BLPA **and** GraphEraser-BEKM **unlearning methods, respectively. We highlight our recommended graph partition methods in the red ground and the best results in blue bold. In general, we reach similar conclusions as Table 3.**

Dataset	Model	Scratch	Random	BLPA	BEKM
Cora	GAT	0.823 ± 0.005	0.723 ± 0.009	**0.774 ± 0.008**	0.756 ± 0.005
	GCN	0.742 ± 0.004	0.448 ± 0.005	**0.657 ± 0.005**	0.474 ± 0.002
	GIN	0.786 ± 0.011	0.755 ± 0.007	0.762 ± 0.009	**0.768 ± 0.027**
	SAGE	0.827 ± 0.007	0.669 ± 0.005	0.721 ± 0.003	**0.731 ± 0.002**
Citeseer	GAT	0.706 ± 0.003	0.620 ± 0.017	**0.674 ± 0.002**	0.670 ± 0.001
	GCN	0.470 ± 0.005	0.464 ± 0.004	0.532 ± 0.008	**0.571 ± 0.017**
	GIN	0.610 ± 0.019	0.592 ± 0.015	0.632 ± 0.026	**0.736 ± 0.020**
	SAGE	0.667 ± 0.002	0.670 ± 0.012	0.680 ± 0.062	**0.711 ± 0.006**
Pubmed	GAT	0.844 ± 0.003	0.827 ± 0.002	0.848 ± 0.002	**0.854 ± 0.007**
	GCN	0.740 ± 0.001	0.549 ± 0.005	**0.716 ± 0.010**	0.578 ± 0.002
	GIN	0.846 ± 0.015	0.857 ± 0.050	0.865 ± 0.004	**0.859 ± 0.003**
	SAGE	0.873 ± 0.001	0.837 ± 0.002	**0.868 ± 0.002**	0.855 ± 0.002
CS	GAT	0.930 ± 0.004	0.882 ± 0.010	0.847 ± 0.002	**0.896 ± 0.001**
	GCN	0.905 ± 0.006	0.706 ± 0.018	**0.790 ± 0.003**	0.732 ± 0.022
	GIN	0.887 ± 0.005	0.858 ± 0.005	0.789 ± 0.013	**0.861 ± 0.002**
	SAGE	0.953 ± 0.004	0.898 ± 0.009	0.896 ± 0.015	**0.923 ± 0.001**
Physics	GAT	0.956 ± 0.002	0.910 ± 0.003	0.925 ± 0.002	**0.928 ± 0.003**
	GCN	0.942 ± 0.005	0.729 ± 0.013	**0.853 ± 0.007**	0.773 ± 0.002
	GIN	0.939 ± 0.003	0.910 ± 0.005	0.917 ± 0.003	**0.929 ± 0.002**
	SAGE	0.950 ± 0.005	0.817 ± 0.021	0.924 ± 0.001	**0.936 ± 0.001**



(a) BLPA **(b)** BEKM

Figure 6: Convergence evaluation of GraphEraser-BLPA **and** GraphEraser-BEKM **on five datasets.**

E EXPERIMENTAL DETAILS

Datasets Description. For the datasets in Table 1, Cora, Citeseer, and Pubmed are citation datasets, where the nodes represent the publications, and there are an edge between two publications if one cite the other. The node features are binary vectors indicating the presence of the keywords from a dictionary, and the class labels represent the publications' research field. CS and Physics are coauthor datasets, where two authors are connected if they collaborate on at least one paper. The node features represent paper keywords for each author's papers, and the class labels indicate most active fields of study for each author.

F CORRELATION BETWEEN IMPORTANCE SCORES AND SHARD PROPERTIES

To support the evidence of effectiveness of LBAggr in Section 6.4, we next investigate the influence of a shard's properties on its importance score determined by the LBAggr method.

Figure 7 depicts the correlation between each shard's F1 score and its importance score. Generally, shard models with more accurate predictions are assigned more significant importance scores. This demonstrates that LBAggr guides GraphEraser to choose the shards with the highest prediction capability.

We further investigate whether each shard's graph properties influence its importance score. To this end, we extract each shard's embedding by averaging all its nodes' embeddings obtained from the pretrained GNN model and project the shard embedding into a two-dimensional space using t-distributed stochastic neighbor embedding (t-SNE) [61]. The results are plotted in Figure 8. As we can see, shards with more significant importance scores are typically accompanied by shards with more miniature importance scores. This implies that for shards trained on similar graphs (similar shard embeddings in the two-dimensional space), our learning-based aggregation assigns a higher score to one of them. In another way, it also learns to discard redundant information to improve utility.

G ABLATION STUDY

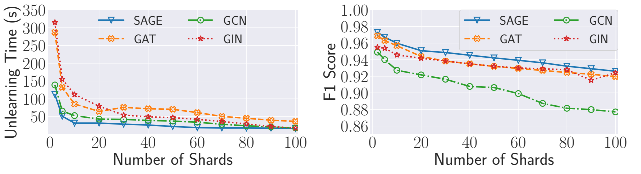
We now evaluate the impact of hyperparameters in the graph partitioning phase with regard to the performance of GraphEraser.

Number of Shards k. We conduct the experiments on the Physics dataset with four GNN models. We vary the number of shards from 2 to 100. As suggested in Section 6.3, we apply GraphEraser-BEKM for GIN, GAT and SAGE, and GraphEraser-BLPA for GCN.

The experimental results in Figure 9 show that the average unlearning time cost decreases when the number of shards increases for all the GNN models. This is expected since larger number of shards means smaller shard size, leading to higher unlearning efficiency. On the other hand, the F1 score of all the four GNN models slightly decrease. Comparing the four GNN models, the utility of GCN model drops the most. We suspect this is because the GCN model requires the node degree information for normalization which is severely reduced by the graph partitioning. The number of shards is an important hyperparameter for GraphEraser, in practice, it should be selected based on the size of the training graph.

Maximum Number of Nodes in Each Shard δ. It is an important parameter in both GraphEraser-BLPA and GraphEraser-BEKM for controlling the degree of balance of the partitioned graphs. The minimum value of δ is $\lceil \frac{n}{k} \rceil$, in which case the shards are balanced. The maximum value of δ is n, meaning no constraints are enforced to the shard size. In these cases, GraphEraser-BLPA and GraphEraser-BEKM fall back to the standard LPA and EKM (embedding clustering with original k-means), respectively.

Intuitively, we aim to make δ as close as $\lceil \frac{n}{k} \rceil$ to achieve balanced shards for efficiency. The remaining concern is what is the impact of δ on the model utility. To this end, we conduct experiments for both GraphEraser-BLPA and GraphEraser-BEKM on five datasets. To make the experiments across different datasets comparable, we introduce a scaling parameter γ in the range of $[0, 1]$ to regulate the choice of δ, i.e., $\delta = \lceil \frac{n}{k} \rceil + \gamma \cdot \left(n - \lceil \frac{n}{k} \rceil\right)$. When $\gamma = 0$, δ equals to $\lceil \frac{n}{k} \rceil$, which is the lower bound of δ; when $\gamma = 1$, δ equals to n, which is the upper bound of δ. Figure 10 illustrates the experimental results. In general, we observe that δ only has a slight impact on the model utility; thus, we set $\delta = \lceil \frac{n}{k} \rceil$ for all of our experiments which leads to the best efficiency.



(a) Efficiency

(b) Utility

Figure 9: Impact of the number of shards k on the unlearning efficiency and model utility on the Physics dataset.

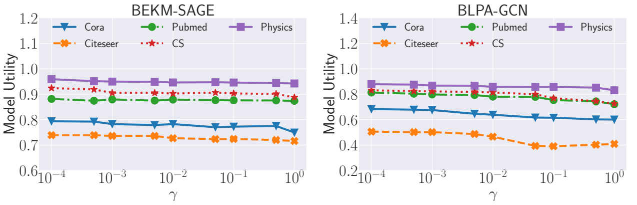
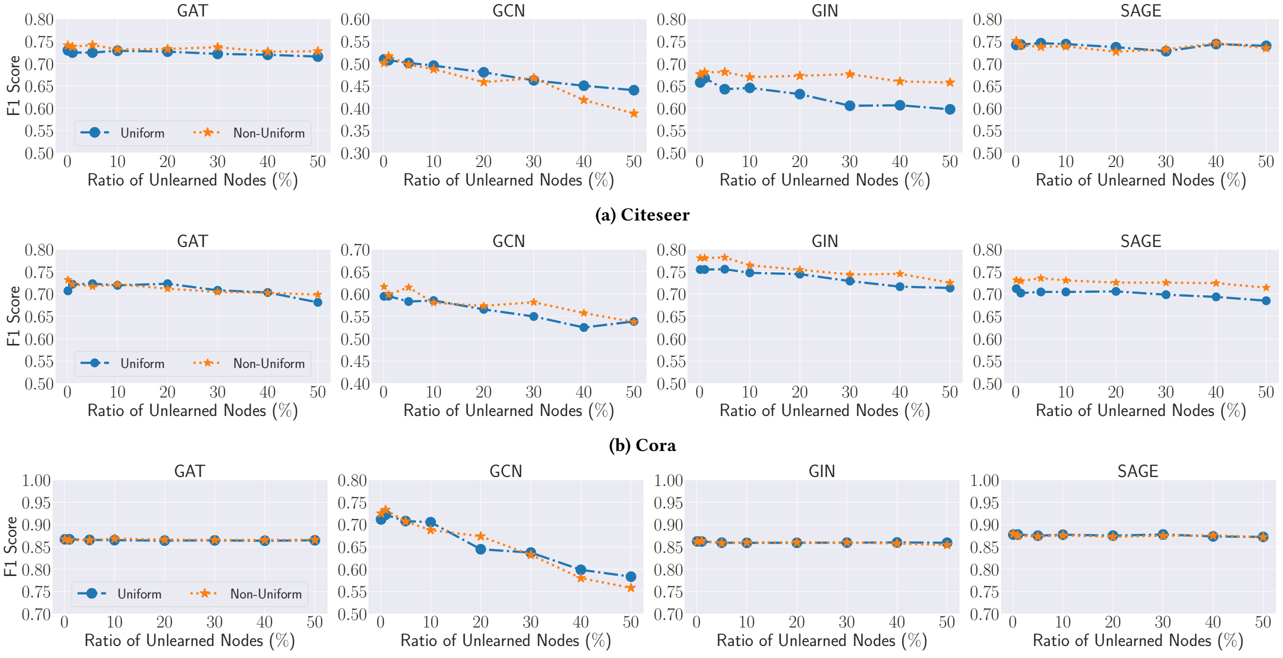


Figure 10: Impact of δ on GraphEraser-BEKM **and** GraphEraser-BLPA **for five datasets.**



(a) Citeseer

(b) Cora

(c) Pubmed

Figure 11: Impact of the ratio of unlearned nodes on the model utility. We evaluate on both uniform and non-uniform unlearning requests distribution.

H ROBUSTNESS OF GraphEraser

In this section, we investigate the impact of the number of unlearned nodes on the model utility of GraphEraser. We consider two distributions of node unlearning request: Uniform and non-uniform. For the uniform unlearning, we randomly delete nodes from all the shards. For non-uniform, we only delete nodes from half the shards with larger sizes.

Figure 11 illustrates the experimental results on three datasets. We first observe that the F1 scores of GraphEraser do not drop significantly in most of the settings when the ratio of unlearned nodes is less than 10%. When a larger ratio of nodes are deleted, we do observe utility degradation in certain cases. For instance, for GCN trained on Pubmed, when the ratio of deleted nodes are 50%, the utility drops from 0.72 to 0.56. Note that in practice, it is unlikely to happen that 50% of the nodes are deleted. In general, we conclude that GraphEraser is robust to a large number of nodes' deletion. Comparing the results of non-uniform and uniform unlearning, we further observe that the distributions of the deletion do not significantly affect the robustness.

I ROLE OF GRAPH STRUCTURE

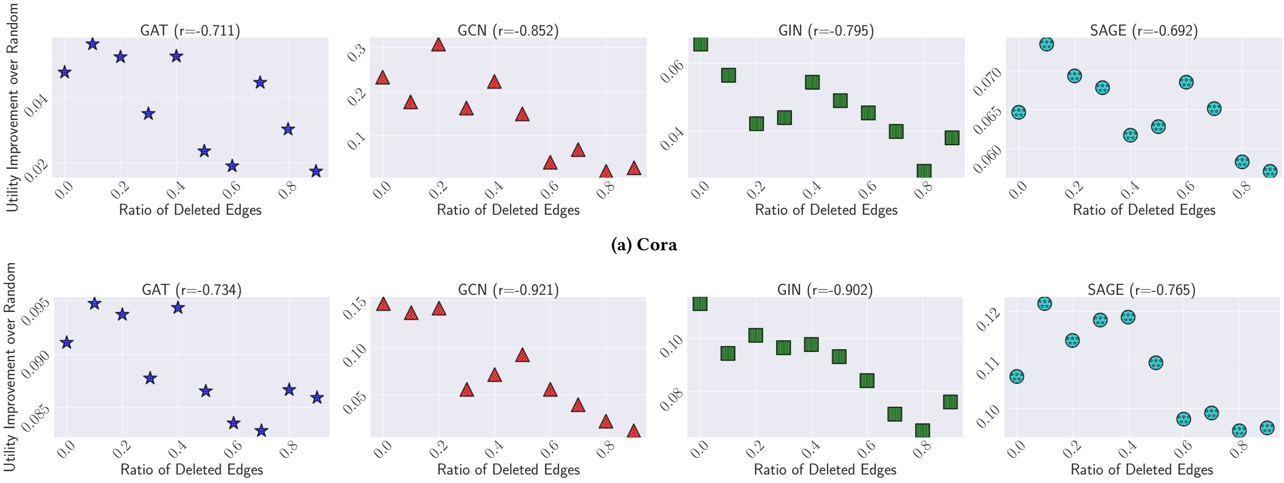
To better illustrate the correlation between the importance of the graph structure and the utility improvement of GraphEraser over Random (SISA), we performed another experiment on Cora and Citeseer. Concretely, we delete different fractions of edges from the training graph to model the significance of the graph structure, and then compare the performance gap between GraphEraser and Random. Figure 12 illustrates the experimental results. We vary the ratio of deleted edges (as shown in the x-axis) from 0% to 90% with a step of 10%. A higher deletion ratio reduces more graph structure information. And the y-axis stands for the utility improvement of GraphEraser over Random. Although there are some outliers, the overall trend (measured by the Pearson correlation score above each sub-figure) shows that when the graph structure is more important, the utility improvement of GraphEraser over Random is more significant in most of the cases.

J COMMUNITY DEPENDENT REMOVAL REQUESTS

Figure 13 illustrates the model utility when the removal requests come from specific community detected by vanilla LPA. The experimental results show that deleting nodes from a single community does not significantly affect the model utility of GraphEraser.

K ADDITIONAL EXPERIMENTAL RESULTS ON EDGE UNLEARNING

Similar to node unlearning, we conduct experiments to evaluate the unlearning efficiency (corresponding to Section 6.2) and model utility (corresponding to Section 6.3) for edge unlearning. Figure 14 and Table 10 illustrate the unlearning efficiency and model utility for edge unlearning, respectively. We reach similar conclusions for node unlearning.



(a) Cora

(b) Citeseer

Figure 12: Correlation between the importance of the graph structure (larger ratio of edge deletion indicates graph structure is less important) and the utility improvement of GraphEraser **over** Random**.**

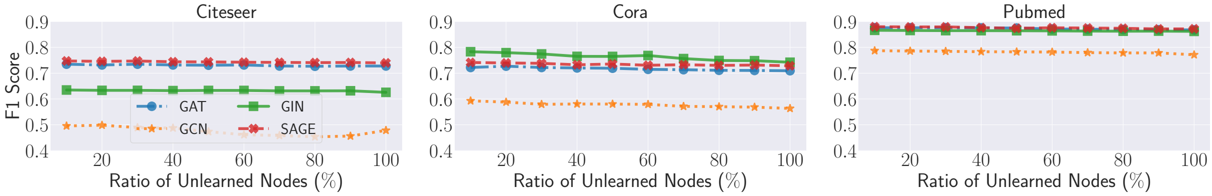


Figure 13: Model utility when the removal requests come from specific community detected by vanilla LPA. X-axis stands for the ratio of unlearned nodes in the selected community.

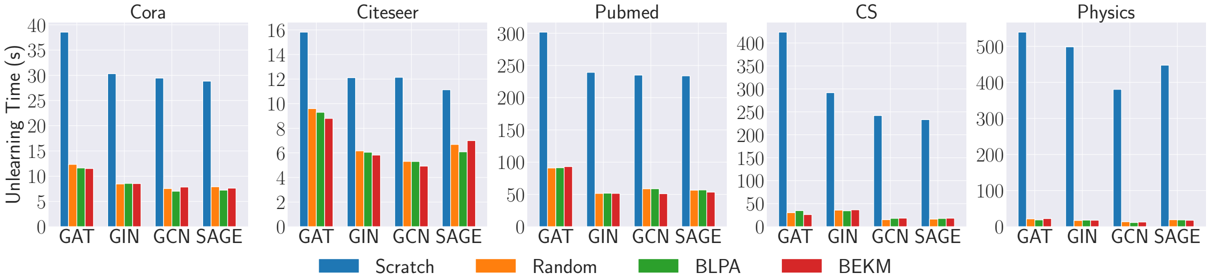


Figure 14: Comparison of edge unlearning efficiency for all graph unlearning methods. BLPA **and** BEKM **stand for** GraphEraser-BLPA **and** GraphEraser-BEKM **unlearning methods, respectively. We observe a similar trend as node unlearning, see Figure 4.**