

Graph Unlearning

Min Chen^{1*} Zhikun Zhang^{1*} Tianhao Wang² Michael Backes¹
Mathias Humbert³ Yang Zhang¹

¹*CISPA Helmholtz Center for Information Security*

²*Purdue University* ³*Cyber-Defence Campus, armasuisse S+T*

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 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 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 method 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 datasets to illustrate the unlearning efficiency and model utility of GraphEraser. We observe that GraphEraser achieves $2.06\times$ (small dataset) to $35.94\times$ (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.

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), the right to be forgotten implies that 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*.

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 [8, 9, 17, 19, 24, 28, 36, 54], among which the SISA method is the most general one in terms of ML models [8]. 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 [43], financial networks [35], biological networks [32], or transportation networks [13]. 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 [5, 12, 13, 26, 33, 40, 56, 60, 67]. 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 [40, 43], where the data subject may request to revoke their data. However, since GNNs rely on the graph structural information to learn the model, randomly partitioning the nodes into sub-graphs as in SISA could severely damage the resulting model utility. Therefore, there is a pressing need for novel methods for unlearning previously seen – but revoked – data samples in the context of GNNs.

1.1 Our Contributions

In this paper, we propose GraphEraser, an efficient unlearning method to achieve high unlearning efficiency and keep 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

*The first two authors made equal contributions.

unlearning in GNN models.

In order to permit efficient retraining while keeping the structural information of the graph, we propose two graph partition strategies. Our first strategy focuses on the graph structural information and tries to preserve it as much as possible by relying on community detection. Our second strategy considers both the graph structural information and the node features information. 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 and clustering methods might lead to highly unbalanced shard sizes due to the structural properties of real-world graphs [18, 70]. 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. In order to avoid this situation, we propose a general principle for balancing the shards resulting from the graph partition and instantiate it with two novel balanced graph partition algorithms. 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 $2.06\times$ shorter on the smallest dataset and up to $35.94\times$ shorter on the largest dataset compared to retraining from scratch. In addition, GraphEraser provides a much better 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 aggregation and majority vote aggregation methods. Our proposed learning-based aggregation achieves up to 112% higher F1 score than that of the majority vote aggregation and up to 93% higher F1 score than that of the mean 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 general principle to achieve balanced graph partitioning and instantiate it with two balanced graph partition algorithms.
- To further 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

Table 1: 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

illustrate the unlearning efficiency and model utility resulting from GraphEraser.

1.2 Organization

The rest of this paper is organized as follows. We introduce the background knowledge of GNNs and machine unlearning in Section 2. In Section 3, we formally define graph unlearning and introduce the general pipeline for GraphEraser. Section 4 introduces the graph partition module for GraphEraser. Section 5 introduces the learning-based aggregation method. We perform extensive experiments in Section 6 to evaluate the unlearning efficiency and model utility of GraphEraser. We summarize the related work in Section 7 and conclude the paper in Section 8.

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 in \mathcal{G} . The notations frequently used in this paper are summarized in Table 1.

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 [25, 40, 60], link prediction [63, 69], and graph classification [64, 66].

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 as described below.

Aggregation Operation. Aggregation is designed for aggregating the messages from a node u ’s neighbors. We introduce four of the most widely used aggregation operations as follows:

- **Graph Isomorphism Networks (GIN) [66].** The GIN method 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) [25].** Another approach, namely SAGE, 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) [33].** 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) [60].** The GAT method uses the attention mechanism (weighted average). It assigns an attention weight or importance score to each neighbor, 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. The updating operation Ψ combines the embeddings from node u and the message from \mathcal{N}_u . We introduce three popular updating operations.

- **Linear Combination [49].** The most basic updating method is to calculate the linear combinations, i.e.,

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

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 [25].** 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., $\Psi_{\text{concat}} = \Psi_{\text{linear}} || E_u$.
- **Interpolation [42].** Another method is to use the weighted average of the linear combination method and the current embedding for updating, i.e., $\Psi_{\text{inter}} = \alpha_1 \circ \Psi_{\text{linear}} + \alpha_2 \circ E_u$.

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 layers 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.

2.2 Machine Unlearning

Thanks to new legislations ensuring the “right to be forgotten”, individuals can now formally request the deletion of their personal data by the service provider controlling them. In the ML context, this implies that the model provider should delete the revoked sample from its training set, but it should also eliminate any influence of the revoked sample on the resulting ML model.

Retraining from Scratch. The simplest way to implement machine 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 method for unlearning and it is easy to implement. However, when the model is complex and the original training dataset is large, the computational overhead of retraining becomes prohibitive. In order to reduce the computational overhead, several approximation approaches have been proposed [6, 9, 15, 19, 20, 28, 50], among which SISA [8] is the most generic one.

SISA. It is an ensemble learning method that can handle different kinds of ML models. With this approach, the training set \mathcal{D}_o is first partitioned into k disjoint shards $\mathcal{D}_o^1, \mathcal{D}_o^2, \dots, \mathcal{D}_o^k$. These k shards are then used separately to train a set of ML models $\mathcal{F}_o^1, \mathcal{F}_o^2, \dots, \mathcal{F}_o^k$. 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 system, namely GraphEraser, to perform machine unlearning on GNNs. We

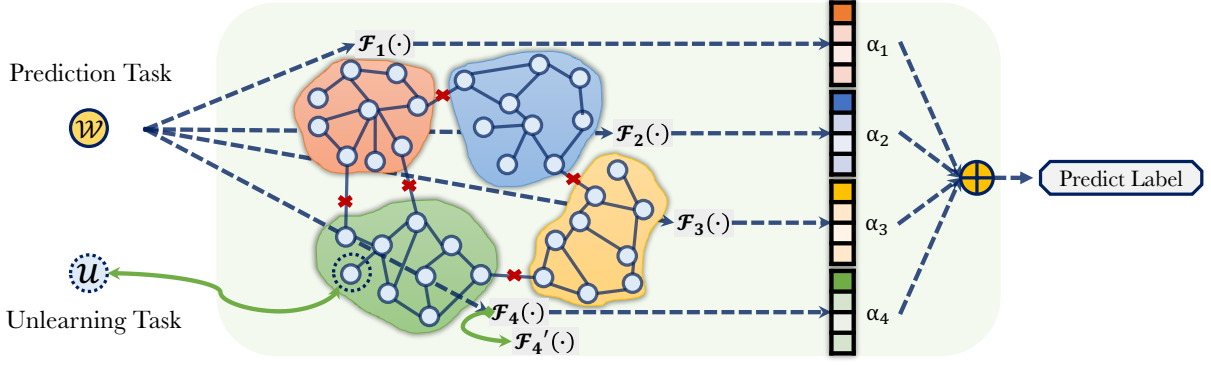


Figure 1: A schematic view of the pipeline of GraphEraser. It partitions the original training graph into disjoint shards, parallelly trains a set of shard models \mathcal{F}_i , and learns an optimal importance score α_i for each shard model. When a node w needs prediction, GraphEraser sends w to all 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 retraining the shard model.

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 need to consider two major factors: *unlearning efficiency* and *model utility*. The unlearning efficiency is related to the retraining time taken when receiving an unlearning request. This time should be as short

as possible. The model utility is related to the unlearned model’s prediction accuracy. Ideally, the prediction accuracy should be close to that of 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 [8], 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, 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 [18, 44, 70] (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 objective of high unlearning efficiency.
- **G2: Comparable Model Utility.** Graph structural information is the major factor that determines the perfor-

mance of GNN. To achieve good utility, i.e., good prediction accuracy in the context of node classification, each shard should preserve the structural properties of the training graph.

3.2 GraphEraser Pipeline

To address the challenge of unlearning in GNNs, we propose GraphEraser, which consists of the following three phases: Graph partition, shard model training, and shard model aggregation.

Graph Partition. We first propose two novel balanced graph partition methods to partition the original model’s training graph into different shards. Balanced graph partition is the key step of GraphEraser and it is designed to fulfill the two requirements defined in Section 3.1. We provide the details of our proposed 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 speed up the training process, the model owner can train different 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. The most straightforward aggregation strategy, also mainly used in [8] 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 name this aggregation strategy MeanAggr.

Observe 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. Therefore, we further propose OptAggr, a learning-based method to find the optimal importance score allocation for different shard models (see details in Section 5). This further improves the prediction accuracy of GraphEraser.

4 Graph Partition

In this section, we introduce the graph partition module for GraphEraser. Different from the image and text data in Euclidean space, the graph data contains rich structural information, thus the challenge is to partition the graph while preserving the structural information as much as possible. Considering the node features and graph structural information in graph data, we identify three graph partition strategies.

- **Strategy 0.** Consider the node features information only and randomly partition the nodes.

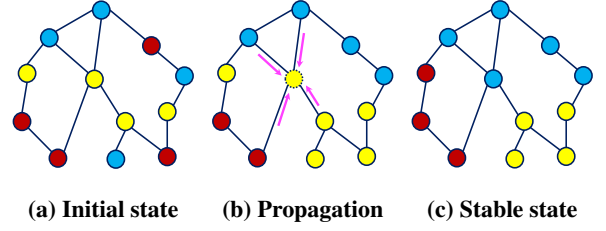


Figure 2: Workflow illustration of the LPA. Different colors represent different shards.

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**.

- **Strategy 1.** Consider the structural information only and try to preserve it as much as possible. One promising approach to do this is to rely on community detection [21, 61, 62].
- **Strategy 2.** Consider both the structural information and the node features. To do this, we can first embed the node features and graph structure into node embeddings using a pretrained GNN models introduced in Section 2.1, 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. 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 [70], Infomap [45], and Label Propagation Algorithm (LPA) [44, 61]. 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 2 gives an illustration of the workflow of LPA. At the initial state, each node is assigned a random shard label (Figure 2a). During the label propagation phase (Figure 2b \rightarrow Figure 2c), 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 2b will change its label to blue because the majority of its neighbors (two nodes

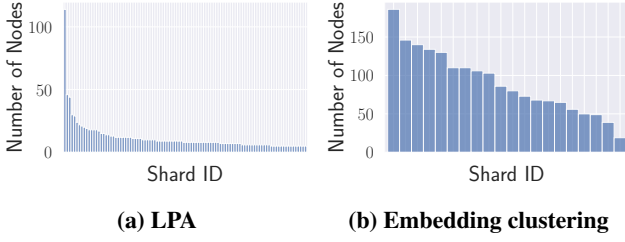


Figure 3: Distribution of shard sizes with classical partition methods on the Cora dataset. The classical LPA generates 341 shards, and we only show the top 100 shards in terms of their sizes.

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 3a shows the distribution of shard size on the Cora dataset [68] (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 A. Directly adopting the unbalanced shards detected by the classical LPA do not satisfy G1, which severely affect 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* pairs 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 1).
- **Step 2: Reassignment Profiles Calculation.** For each node u , we denote its *reassignment profile* by a tuple $\langle u, \mathbb{C}_{src}, \mathbb{C}_{dst}, \xi \rangle$, where \mathbb{C}_{src} and \mathbb{C}_{dst} are the current and

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 δ ; maximum iteration T ;
Output: Shards $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_k\}$;
1 **Initialization:** Randomly allocate all nodes into k shards and obtain $\mathbb{C}^0 = \{\mathbb{C}_1^0, \mathbb{C}_2^0, \dots, \mathbb{C}_k^0\}$, step $t = 0$;
Label Propagation: while True do
2 **foreach** node u in \mathcal{V} **do**
3 **foreach** shard \mathbb{C}_{dst} in $\{\mathbb{C} | v \in \mathbb{C}, v \in \mathcal{N}_u\}$ **do**
4 Store tuple $\langle u, \mathbb{C}_{src}, \mathbb{C}_{dst}, \xi \rangle$ in \mathbb{F} ;
5 Sort \mathbb{F} by ξ in descending order and obtain \mathbb{F}_s ;
6 **foreach** tuple in \mathbb{F}_s **do**
7 **if** $|\mathbb{C}_{dst}^t| < \delta$ **then**
8 $\mathbb{C}_{dst}^t \leftarrow \mathbb{C}_{dst}^{t-1} \cup u$;
9 $\mathbb{C}_{src}^t \leftarrow \mathbb{C}_{src}^{t-1} \setminus u$;
10 **if** $t > T$ or the shard does not change **then**
11 break;
12 $t \leftarrow t + 1$;
12 **return** \mathbb{C}^t ;

destination shards of node u , ξ is the neighbor counts of the destination shard \mathbb{C}_{dst} (Line 2 - Line 4). 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 5).
- **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 5 - Line 8).

The BLPA algorithm repeats steps 2-4 until the algorithm reaches the maximum iteration T , or the shard does not change (Line 13-14).

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 [31], 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

Algorithm 2: BEKM Algorithm

Input: Node embeddings $\mathbb{E} = \{E_1, E_2, \dots, E_n\}$, the number of clusters k , maximum number of nodes embedding in each cluster δ ; maximum number of iteration T ;

Output: Clusters $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_k\}$;

```
1 Initialization: Randomly select  $k$  centroids  
    $C^0 = \{C_1^0, C_2^0, \dots, C_k^0\}$ , step  $t = 0$ ; while True do  
2   Nodes Reassignment:  
3   foreach node embedding  $i \in \mathbb{E}$  do  
4     foreach centroid  $j \in \mathbb{C}$  do  
5       Store  $\|E_i - C_j\|_2$  in  $\mathbb{F}$ ;  
6   Sort  $\mathbb{F}$  in ascending order and obtain  $\mathbb{F}_s$ . foreach  
   node  $i$  and centroid  $j$  in  $\mathbb{F}_s$  do  
7     if  $|\mathbb{C}_j^t| < \delta$  then  
8        $\mathbb{C}_j^t \leftarrow \mathbb{C}_j^t \cup n_i$ ;  
9   Centroids Updating: foreach cluster  $j \in \mathbb{C}^t$  do  
10     $C_j^t = \frac{\sum_{i \in \mathbb{C}_j^t} E_i}{|\mathbb{C}_j^t|}$ ;  
11  if  $t > T$  or the centroid do not change then  
12    break;  
13   $t \leftarrow t + 1$ ;  
14 return  $\mathbb{C}^t$ ;
```

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 3b, 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 $\mathbb{E} = \{E_1, E_2, \dots, E_n\}$, 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 $C^0 = \{C_1^0, C_2^0, \dots, C_k^0\}$ (Line 1).
- **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 3 - Line 5).
- **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 6).

- **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 6 - Line 10). 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 11 - Line 12).

Note that the choice between BLPA and BEKM depends on the GNN structure. In Section 6, we will provide a guideline on which one to choose.

5 Optimal Aggregation (OptAggr)

As discussed in Section 3.2, various shard models may have different contributions with regard to the final prediction. We can assign different importance scores to different shard models. In this Section, we propose learning-based aggregation method OptAggr. It assigns importance scores for different shards based on the following optimization problem.

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

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. 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 Score Updating. Note that we use the nodes in the training graph to learn the optimal importance scores. However, these nodes can be requested to be revoked by their data subject as well. Therefore, we need to retrain our optimal importance scores if an unlearning request corresponds to a node used to train the optimal importance scores, and this training time is part of the unlearning time as well. To reduce this optimal importance scores retraining time, we propose to use a small random subset of nodes from the training graph. We empirically show in Section 6 that using a small portion of nodes is sufficient, and can effectively improve the model utility compared to other aggregation methods.

Table 2: Dataset statistics.

Dataset	Type	#. 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

6 Evaluation

In this section, we first evaluate the unlearning efficiency and model utility of GraphEraser. Second, we conduct experiments to show the superiority of our proposed learning-based aggregation method OptAggr. Third, we investigate the correlation between the properties of the shard models and the importance scores resulting from OptAggr. Fourth, we evaluate the impact of the number of shards on the unlearning efficiency and model utility. Finally, we show the robustness of GraphEraser to the number of unlearned nodes/edges.

6.1 Experimental Setup

Datasets. We conduct our experiments on five public graph datasets, including Cora [68], Citeseer [68], Pubmed [68], CS [51], and Physics [51]. These datasets are widely used as benchmark datasets for evaluating the performance of GNN models [33, 52, 73]. Among them, Cora, Citeseer, and Pubmed are citation datasets; CS and Physics are coauthor datasets. Table 2 provides the statistics of all the datasets.

GNN Models. We evaluate the efficiency and accuracy of GraphEraser on four state-of-the-art GNN models, including SAGE, GCN, GAT, and GIN (discussed in Section 2.1). For each GNN model, we stack two layers of GNN modules. All the models are implemented with the PyTorch Geometric¹ library. All the GNN models (including the shard models) considered in this paper are trained for 100 epochs.

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 *F1 score* to measure the model utility, which is widely used to evaluate the prediction ability of GNN models on multi-class classification [22]. The F1 score is a harmonic mean of *precision* and *recall*, and can provide a better measure of the incorrectly classified cases than the accuracy metric.

Competitors. We have two natural baselines in our experiments: 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 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 [25], 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.

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 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. The training graphs of different datasets are partitioned into different number of shards, based on the size of the original dataset, which ensures each shard is trained on a reasonable number of nodes and edges. Specifically, we partition Cora, Citeseer, Pubmed, CS, and Physics into 20, 20, 50, 50, and 100 shards, respectively.

Figure 4 illustrates the node unlearning efficiency for different graph unlearning methods. The experimental results show that the shard-based methods, i.e., Random, GraphEraser-BLPA, and GraphEraser-BEKM, 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, we observe that 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 $4.16\times$ for the Cora dataset, $3.08\times$ for the Citeseer dataset, $5.40\times$ for the Pubmed dataset, $19.25\times$ for the CS dataset, and $35.9\times$ for the Physics datasets. This is expected. From the Scratch method perspective, 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).

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

¹https://github.com/rustyls/pytorch_geometric

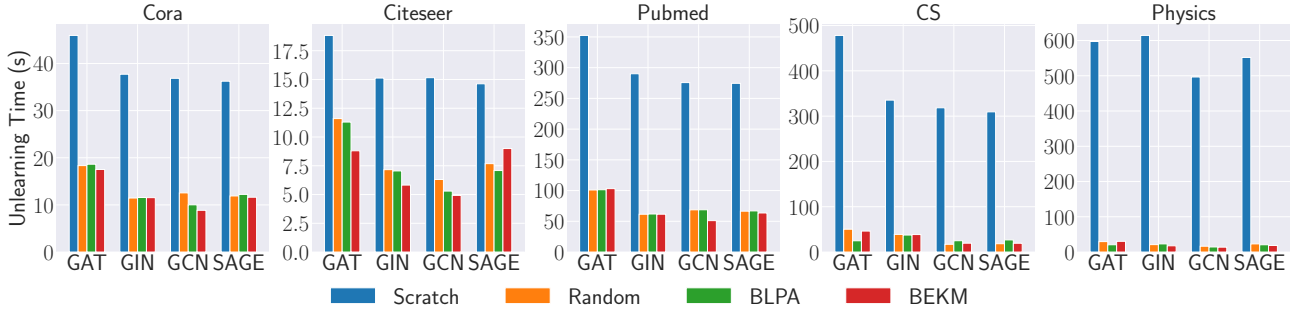


Figure 4: Comparison of node unlearning efficiency for all graph unlearning methods. BLPA and BEKM stand for GraphEraser-BLPA and GraphEraser-BEKM unlearning methods, respectively.

Table 3: Comparison of F1 scores for all graph unlearning methods for node unlearning. BLPA and BEKM stand for GraphEraser-BLPA and GraphEraser-BEKM unlearning methods, respectively. Best results are highlighted in bold.

	Model	Scratch	Random	BLPA	BEKM
Cora	GAT	0.823 \pm 0.006	0.726 \pm 0.004	0.727 \pm 0.009	0.754 \pm 0.009
	GCN	0.739 \pm 0.003	0.509 \pm 0.009	0.676 \pm 0.004	0.493 \pm 0.006
	GIN	0.787 \pm 0.013	0.766 \pm 0.021	0.753 \pm 0.015	0.801 \pm 0.018
	SAGE	0.824 \pm 0.004	0.682 \pm 0.013	0.684 \pm 0.014	0.740 \pm 0.013
Citeseer	GAT	0.691 \pm 0.015	0.631 \pm 0.015	0.676 \pm 0.004	0.746 \pm 0.006
	GCN	0.493 \pm 0.006	0.277 \pm 0.009	0.450 \pm 0.006	0.332 \pm 0.006
	GIN	0.587 \pm 0.031	0.626 \pm 0.022	0.612 \pm 0.026	0.739 \pm 0.020
	SAGE	0.668 \pm 0.013	0.623 \pm 0.014	0.657 \pm 0.012	0.716 \pm 0.007
Pubmed	GAT	0.851 \pm 0.004	0.857 \pm 0.002	0.858 \pm 0.003	0.860 \pm 0.003
	GCN	0.748 \pm 0.017	0.551 \pm 0.005	0.718 \pm 0.010	0.482 \pm 0.003
	GIN	0.837 \pm 0.015	0.856 \pm 0.003	0.855 \pm 0.004	0.859 \pm 0.003
	SAGE	0.874 \pm 0.003	0.857 \pm 0.002	0.863 \pm 0.002	0.862 \pm 0.002
CS	GAT	0.919 \pm 0.004	0.882 \pm 0.000	0.858 \pm 0.004	0.886 \pm 0.002
	GCN	0.903 \pm 0.006	0.706 \pm 0.008	0.750 \pm 0.023	0.671 \pm 0.012
	GIN	0.867 \pm 0.005	0.858 \pm 0.005	0.789 \pm 0.013	0.861 \pm 0.002
	SAGE	0.932 \pm 0.004	0.905 \pm 0.004	0.886 \pm 0.010	0.913 \pm 0.002
Physics	GAT	0.955 \pm 0.005	0.920 \pm 0.002	0.921 \pm 0.004	0.925 \pm 0.001
	GCN	0.947 \pm 0.002	0.747 \pm 0.010	0.858 \pm 0.008	0.750 \pm 0.001
	GIN	0.934 \pm 0.003	0.921 \pm 0.002	0.907 \pm 0.003	0.926 \pm 0.001
	SAGE	0.956 \pm 0.005	0.823 \pm 0.011	0.922 \pm 0.001	0.933 \pm 0.001

6.3 Evaluation of Model Utility

Next, we evaluate the model utility of different graph unlearning methods. We conduct the experiments on five datasets and four GNN models. For a fair comparison, we also apply OptAggr for Random.

Table 3 shows the experimental results for node unlearning. We 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.

Influence of Datasets. 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 on these datasets. 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 this, we introduce a baseline that uses a 3-layer MLP (multi-layer 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 almost the same 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.

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 first compare the F1 score of MLP and GNN before choosing the most suitable graph unlearning methods. Concretely, if the gap in the F1 score between MLP and GNN is small, then 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 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 due to the fact that 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.

We reach similar conclusions for edge unlearning whose

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

results are shown in [Appendix B](#). 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.

6.4 Effectiveness of OptAggr

To validate the effectiveness of the OptAggr method proposed in [Section 5](#), we compare with MeanAggr and MajAggr by conducting experiments on five datasets and four GNN models. [Table 5](#) illustrates the F1 scores of different aggregation methods for Scratch, GraphEraser-BLPA, and GraphEraser-BEKM. Due to space limitation, we defer the results of Random to [Appendix C](#). In general, the OptAggr 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, OptAggr achieves 0.357 higher F1 score than that of MajAggr for the GCN model. Comparing different GNN models, the GIN model benefits the least from OptAggr.

We observe that, in terms of model utility, the GraphEraser-BLPA method benefits the most from OptAggr. We conjecture that it is due to the BLPA partition method can capture the local structural information while losing some of the global structural information of the training graph [\[52, 62\]](#). Using OptAggr helps better capture the global structural information by assigning different importance scores to shard models.

We also observe that the MajAggr method performs the worst in most cases. We posit it is due to the fact that 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.

6.5 Correlation between Importance Scores and Shard Properties

Next, we investigate the influence of a shard’s properties on its importance score determined by the OptAggr method.

[Figure 5](#) depicts the correlation between each shard’s F1 score and its importance score. In general, shard models with more accurate predictions are assigned larger importance scores. This demonstrates that OptAggr indeed guides

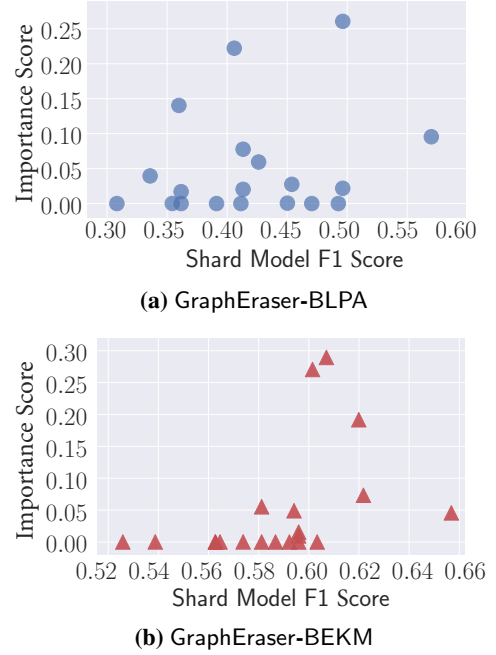


Figure 5: Correlation between the importance score of a shard model and its F1 score on the Cora dataset. The x-axis stands for the shard model’s F1 score, and the y-axis stands for the importance score of that shard. We report the results of GAT model with GraphEraser-BLPA and GraphEraser-BEKM unlearning methods.

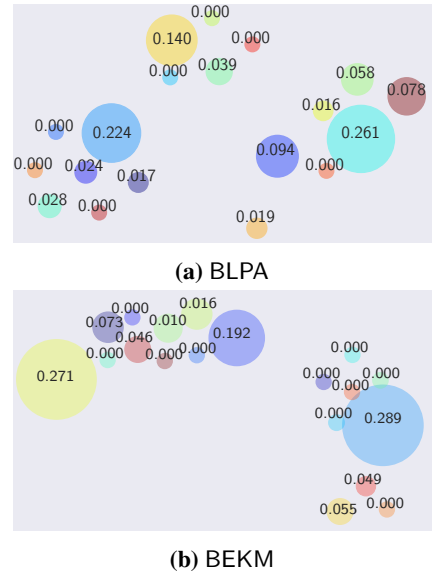


Figure 6: The t-SNE plot of shard embeddings for the Cora dataset. Each circle represents the mean node embeddings of a shard, where the circle size is proportional to its importance score in annotations.

Table 5: Comparison of F1 scores for different aggregation methods. Note that the Scratch method does not need aggregation. Best results are highlighted in bold.

	Dataset/ Model	Scratch	GraphEraser-BLPA			GraphEraser-BEKM		
			MeanAggr	MajAggr	OptAggr	MeanAggr	MajAggr	OptAggr
Cora	GAT	0.823 ± 0.006	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	0.590 ± 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	0.681 ± 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	0.354 ± 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	0.504 ± 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	0.372 ± 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	0.451 ± 0.062	0.447 ± 0.032	0.612 ± 0.026	0.725 ± 0.016	0.690 ± 0.010	0.739 ± 0.020
	SAGE	0.668 ± 0.013	0.447 ± 0.007	0.472 ± 0.024	0.657 ± 0.012	0.715 ± 0.003	0.710 ± 0.004	0.716 ± 0.007
Pubmed	GAT	0.851 ± 0.004	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	0.644 ± 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	0.849 ± 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	0.841 ± 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	0.664 ± 0.015	0.662 ± 0.009	0.858 ± 0.004	0.885 ± 0.001	0.882 ± 0.003	0.886 ± 0.002
	GCN	0.903 ± 0.006	0.658 ± 0.004	0.440 ± 0.003	0.750 ± 0.023	0.620 ± 0.003	0.502 ± 0.003	0.671 ± 0.012
	GIN	0.867 ± 0.005	0.655 ± 0.024	0.691 ± 0.011	0.789 ± 0.013	0.857 ± 0.005	0.844 ± 0.005	0.861 ± 0.002
	SAGE	0.932 ± 0.004	0.745 ± 0.009	0.679 ± 0.003	0.886 ± 0.010	0.904 ± 0.007	0.903 ± 0.001	0.913 ± 0.002
Physics	GAT	0.955 ± 0.005	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	0.817 ± 0.003	0.770 ± 0.001	0.858 ± 0.008	0.575 ± 0.003	0.506 ± 0.001	0.750 ± 0.001
	GIN	0.934 ± 0.003	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	0.905 ± 0.003	0.894 ± 0.003	0.922 ± 0.001	0.926 ± 0.003	0.924 ± 0.002	0.933 ± 0.001

GraphEraser to choose the shards with highest prediction capability.

We further investigate whether each shard’s graph properties can influence its importance score. To this end, we extract each shard’s embedding by averaging all its nodes’ embeddings obtained from pretrained GNN model, and project the shard embedding into a two-dimensional space using t-distributed stochastic neighbor embedding (t-SNE) [59]. The results are plotted in Figure 6. As we can see, shards with larger importance scores are normally accompanied by shards with smaller importance scores. This implies that for shards trained on similar graphs (similar shard embeddings in the two-dimensional space), our optimal aggregation assigns a higher score to one of them. In another way, our optimal aggregation also learns to discard redundant information to improve utility.

6.6 Impact of the Number of Shards

In this section, we investigate the impact of the number of shards on the unlearning efficiency and model utility. We conduct the experiments on the Physics dataset with four GNN models. We vary the number of shards from 2 to 100.

The experimental results in Figure 7 show that the average unlearning time 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

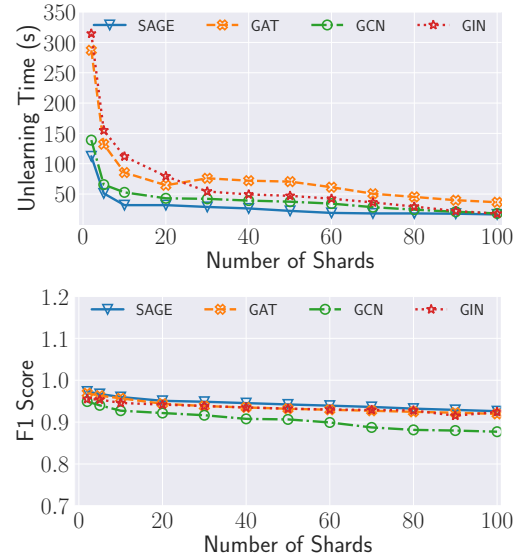


Figure 7: Impact of the number of shards on the unlearning efficiency and model utility on the Physics dataset. As suggested in Section 6.3, we apply GraphEraser-BEKM for GIN, GAT and SAGE, and GraphEraser-BLPA for GCN.

should be selected based on the size of the training graph.

6.7 Robustness of GraphEraser

In this section, we investigate the impact of the number of unlearned nodes and the number of unlearned edges on the model utility of GraphEraser. The experiments are conducted on the Citeseer dataset with four GNN models.

Impact of the Number of Unlearned Nodes. Figure 8a illustrates the impact of the number of unlearned nodes on

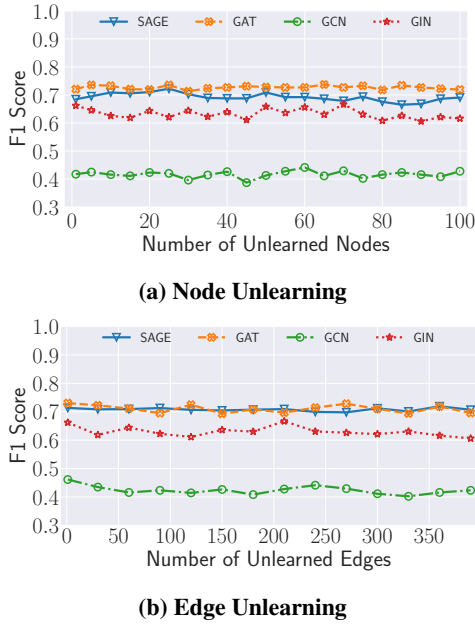


Figure 8: Impact of the number of unlearned nodes and edges on the model utility on the Citeseer dataset. As suggested in Section 6.3, we apply GraphEraser-BEKM for GIN, GAT and SAGE, and GraphEraser-BLPA for GCN.

the model utility for node unlearning. The number of unlearned nodes ranges from 1 to 100 (5% of the whole training graph). We observe that all the graph unlearning methods are robust to node deletion, even when the number of deleted nodes reaches 5% of the whole training graph.

Impact of the Number of Unlearned Edges. Figure 8b shows the impact of the number of unlearned edges on the model utility for edge unlearning. We reach similar conclusions with node unlearning: GraphEraser is robust to edge unlearning, even when 10% (400) edges are removed from the training graph.

7 Related Work

Machine Unlearning. The notion of machine unlearning was first proposed by Cao et al. [9]. The most straightforward approach to implement machine unlearning is to remove the revoked samples from the original training set and retrain the ML model from scratch. However, retraining from scratch is computationally infeasible when the dataset scale is large. Thus, most of the previous studies in machine unlearning focus on reducing the computational overhead of the unlearning process [6, 8, 9, 28, 50].

Cao et al. [9] proposed to transform the learning algorithms into a summation form that follows statistical query learning, breaking down the dependencies of training data. To remove a data sample, the model owner only needs to remove the transformations of this data sample from the summations that depend on this sample. However, the algorithm in [9] is not applicable to learning algorithms that cannot be transformed into summation form, such as neural networks. Thus, Bourtole et al. [8] proposed a more general algorithm

named SISA. The main idea of SISA is to split the training data into several disjoint shards, with each shard training one sub-model. To remove a specific sample, the model owner only needs to retrain the sub-model that contains this sample. To further speed up the unlearning process, the authors proposed to split each shard into several slices and store the intermediate model parameters when the model is updated by each slice. However, most of the existing unlearning methods focus on image and text data in the Euclidean space and cannot be directly applied to graph data.

It is worth noting that Chen et al. in [11] have shown that the machine unlearning process can leak information about the unlearned sample (with membership inference) when the adversary has access to both the original model and the unlearned model. In the future, we plan to investigate whether GraphEraser is prone to such leakage as well.

Privacy and Security in ML Systems. Another line of research relates to the various attacks on ML systems. For the general ML models, one type of such attack aims to mislead the behavior of the ML models by perturbing the data samples, such as *adversarial examples* [7, 10, 41, 57] and *data poisoning* [23, 29, 37, 47]. Another major type of attacks aim to infer information from the ML model, including the model’s training dataset (membership inference [48, 53], model inversion [16, 71], and attribute inference [38, 55]) or the model’s parameters (model stealing [39, 58]).

There are several recent studies aiming to attack GNN models. He et al. [27] propose a link stealing attack that aims to infer whether there is a link between two nodes in the semi-supervised setting. There are also works that aim to infer the information of the graph using graph embedding [14, 34]. To mitigate the privacy risks in GNN models, several studies [30, 46, 73] propose privacy-preserving mechanisms tailored to GNN models. Another type of attack against GNN models is the backdoor attack [65, 72] which leads the targeted ML model to predict a specific class when there is a predefined “trigger” in the data sample.

8 Conclusion

In this paper, we propose the first machine unlearning algorithm 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 partitioning 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.

References

- [1] <https://gdpr-info.eu/>. 1

- [2] <https://oag.ca.gov/privacy/ccpa>. 1
- [3] <https://laws-lois.justice.gc.ca/ENG/ACTS/P-8.6/index.html>. 1
- [4] https://iapp.org/media/pdf/resource_center/Brazilian_General_Data_Protection_Law.pdf. 1
- [5] James Atwood and Don Towsley. Diffusion-Convolutional Neural Networks. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1993–2001. NIPS, 2016. 1
- [6] Thomas Baumhauer, Pascal Schöttle, and Matthias Zeppelzauer. Machine Unlearning: Linear Filtration for Logit-based Classifier. *CoRR abs/2002.02730*, 2020. 3, 12
- [7] Battista Biggio, Iginio Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion Attacks against Machine Learning at Test Time. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, pages 387–402. Springer, 2013. 12
- [8] Lucas Bourtole, Varun Chandrasekaran, Christopher Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine Unlearning. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2021. 1, 3, 4, 5, 12
- [9] Yinzhi Cao and Junfeng Yang. Towards Making Systems Forget with Machine Unlearning. In *IEEE Symposium on Security and Privacy (S&P)*, pages 463–480. IEEE, 2015. 1, 3, 12
- [10] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy (S&P)*, pages 39–57. IEEE, 2017. 12
- [11] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. When Machine Unlearning Jeopardizes Privacy. *CoRR abs/2005.02205*, 2020. 12
- [12] Michaël Defferrard, Xavier Bresson, and Pierre 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. 1
- [13] Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. Dynamic Spatial-Temporal Graph Convolutional Neural Networks for Traffic Forecasting. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 890–897. AAAI, 2019. 1
- [14] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. Quantifying Privacy Leakage in Graph Embedding. *CoRR abs/2010.00906*, 2020. 12
- [15] Daniel L. Felps, Amelia D. Schwickerath, Joyce D. Williams, Trung N. Vuong, Alan Briggs, Matthew Hunt, Evan Sakmar, David D. Saranchak, and Tyler Shumaker. Class Clown: Data Redaction in Machine Unlearning at Enterprise Scale. *CoRR abs/2012.04699*, 2020. 3
- [16] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1322–1333. ACM, 2015. 12
- [17] Antonio A. Ginart, Melody Y. Guan, Gregory Valiant, and James Zou. Making AI Forget You: Data Deletion in Machine Learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 3513–3526. NeurIPS, 2019. 1
- [18] Michelle Girvan and M. E. J. Newman. Community Structure in Social and Biological Networks. *Proceedings of the National Academy of Sciences*, 2002. 2, 4
- [19] Aditya Golatkar, Alessandro Achille, and Stefano 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. 1, 3
- [20] Aditya Golatkar, Alessandro Achille, and Stefano 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. 3
- [21] Steve Gregory. Finding Overlapping Communities in Networks by Label Propagation. *New Journal of Physics*, 2010. 5
- [22] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 855–864. ACM, 2016. 8
- [23] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *CoRR abs/1708.06733*, 2017. 12
- [24] Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. Certified Data Removal from Machine Learning Models. In *International Conference on Machine Learning (ICML)*, pages 3832–3842. PMLR, 2020. 1
- [25] William L. Hamilton, Zitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1025–1035. NIPS, 2017. 2, 3, 8
- [26] Songtao He, Fayyen Bastani, Satvat Jagwani, Edward Park, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Samuel Madden, and Mohammad Amin Sadeghi. RoadTagger: Robust Road Attribute Inference with Graph Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 10965–10972. AAAI, 2020. 1

- [27] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. Stealing Links from Graph Neural Networks. In *USENIX Security Symposium (USENIX Security)*. USENIX, 2021. 12
- [28] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. Approximate Data Deletion from Machine Learning Models: Algorithms and Evaluations. *CoRR abs/2002.10077*, 2020. 1, 3, 12
- [29] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In *IEEE Symposium on Security and Privacy (S&P)*, pages 19–35. IEEE, 2018. 12
- [30] Meng Jiang, Taeho Jung, Ryan Karl, and Tong Zhao. Federated Dynamic GNN with Secure Aggregation. *CoRR abs/2009.07351*, 2020. 12
- [31] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002. 6
- [32] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular Graph Convolutions: Moving Beyond Fingerprints. *Journal of Computer-Aided Molecular Design*, 2016. 1
- [33] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*, 2017. 1, 3, 8
- [34] Kaiyang Li, Guangchun Luo, Yang Ye, Wei Li, Shihao Ji, and Zhipeng Cai. Adversarial Privacy Preserving Graph Embedding against Inference Attack. *CoRR abs/2008.13072*, 2020. 12
- [35] Xiaoxiao Li, João Saúde, Prashant Reddy, and Manuela 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. 1
- [36] Yang Liu, Zhuo Ma, Ximeng Liu, and Jianfeng Ma. Learn to Forget: User-Level Memorization Elimination in Federated Learning. *CoRR abs/2003.10933*, 2020. 1
- [37] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019. 12
- [38] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE Symposium on Security and Privacy (S&P)*, pages 497–512. IEEE, 2019. 12
- [39] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff Nets: Stealing Functionality of Black-Box Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4954–4963. IEEE, 2019. 12
- [40] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. Pin-Sage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2311–2320. ACM, 2020. 1, 2
- [41] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *IEEE Symposium on Security and Privacy (S&P)*, pages 582–597. IEEE, 2016. 12
- [42] Trang Pham, Truyen Tran, Dinh Q. Phung, and Svetha Venkatesh. Column Networks for Collective Classification. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2485–2491. AAAI, 2017. 3
- [43] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. DeepInf: Social Influence Prediction with Deep Learning. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2110–2119. ACM, 2018. 1
- [44] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks. *Physical Review E*, 2007. 4, 5
- [45] Martin Rosvall and Carl T. Bergstrom. Maps of Random Walks on Complex Networks Reveal Community Structure. *Proceedings of the National Academy of Sciences*, 2008. 5
- [46] Sina Sajadmanesh and Daniel Gatica-Perez. Locally Private Graph Neural Networks. *CoRR abs/2006.05535*, 2020. 12
- [47] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic Backdoor Attacks Against Machine Learning Models. *CoRR abs/2003.03675*, 2020. 12
- [48] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019. 12
- [49] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 2009. 3
- [50] Sebastian Schelter. “Amnesia” - Towards Machine Learning Models That Can Forget User Data Very Fast. In *Annual Conference on Innovative Data Systems Research (CIDR)*, 2020. 3, 12

- [51] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of Graph Neural Network Evaluation. *CoRR abs/1811.05868*, 2018. 8
- [52] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. *CoRR abs/2009.03509*, 2020. 8, 10
- [53] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *IEEE Symposium on Security and Privacy (S&P)*, pages 3–18. IEEE, 2017. 12
- [54] David Marco Sommer, Liwei Song, Sameer Wagh, and Prateek Mittal. Towards Probabilistic Verification of Machine Unlearning. *CoRR abs/2003.04247*, 2020. 1
- [55] Congzheng Song and Vitaly Shmatikov. Overlearning Reveals Sensitive Attributes. In *International Conference on Learning Representations (ICLR)*, 2020. 12
- [56] Wen Torng and Russ B. Altman. Graph Convolutional Neural Networks for Predicting Drug-Target Interactions. *Journal of Chemical Information and Modeling*, 2019. 1
- [57] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble Adversarial Training: Attacks and Defenses. In *International Conference on Learning Representations (ICLR)*, 2017. 12
- [58] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing Machine Learning Models via Prediction APIs. In *USENIX Security Symposium (USENIX Security)*, pages 601–618. USENIX, 2016. 12
- [59] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 2008. 11
- [60] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 2, 3
- [61] Fei Wang and Changshui Zhang. Label Propagation through Linear Neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 2008. 5
- [62] Hongwei Wang and Jure Leskovec. Unifying Graph Convolutional Neural Networks and Label Propagation. *CoRR abs/2002.06755*, 2020. 5, 10
- [63] Xiaokai Wei, Linchuan Xu, Bokai Cao, and Philip Yu. Cross View Link Prediction by Learning Noise-resilient Representation Consensus. In *International Conference on World Wide Web (WWW)*, pages 1611–1619. ACM, 2017. 2
- [64] Jun Wu, Jingrui He, and Jiejun 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. 2
- [65] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph Backdoor. *CoRR abs/2006.11890*, 2020. 12
- [66] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations (ICLR)*, 2019. 2, 3
- [67] Carl Yang, Aditya Pal, Andrew Zhai, Nikil Pancha, Jiawei Han, Charles Rosenberg, and Jure 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. 1
- [68] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *International Conference on Machine Learning (ICML)*, pages 40–48. JMLR, 2016. 6, 8
- [69] Muhan Zhang and Yixin Chen. Weisfeiler-Lehman Neural Machine for Link Prediction. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, pages 575–583. ACM, 2017. 2
- [70] Xiao Zhang and M. E. J. Newman. Multiway Spectral Community Detection in Networks. *Physical Review E*, 2015. 2, 4, 5
- [71] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 250–258. IEEE, 2020. 12
- [72] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor Attacks to Graph Neural Networks. *CoRR abs/2006.11165*, 2020. 12
- [73] Jun Zhou, Chaochao Chen, Longfei Zheng, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. Privacy-Preserving Graph Neural Network for Node Classification. *CoRR abs/2005.11903*, 2020. 8, 12

A Visualization of LPA

Figure 10 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.

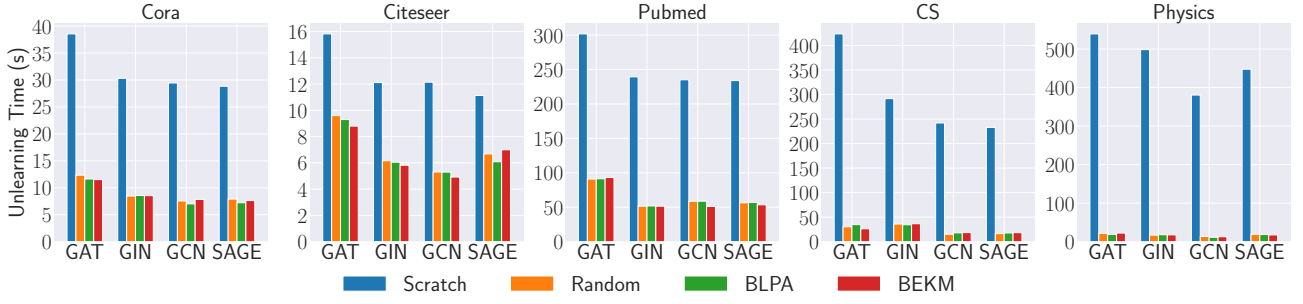


Figure 9: Comparison of edge unlearning efficiency for all graph unlearning methods. BLPA and BEKM stand for GraphEraser-BLPA and GraphEraser-BEKM unlearning methods, respectively.

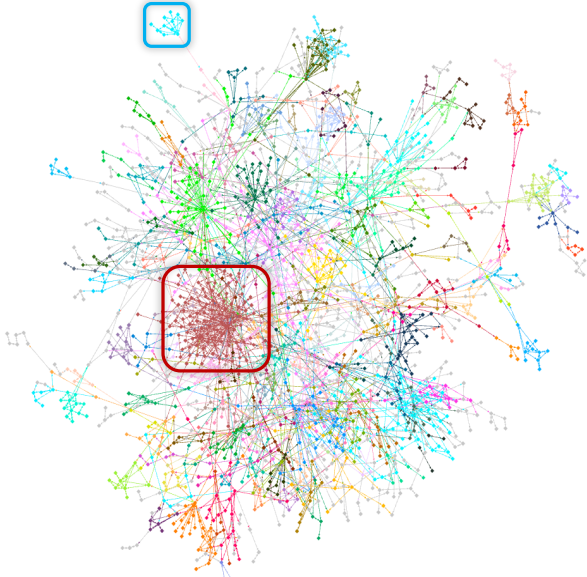


Figure 10: Visualization of shards detected by classical LPA on the Cora dataset. Different colors stand for different shards, where the red box and blue box mark a large shard and a small shard, respectively.

B Experimental Results for 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 9](#) and [Table 6](#) illustrate the unlearning efficiency and model utility for edge unlearning, respectively. We reach similar conclusions for node unlearning.

C Comparison of Aggregation Methods for Random

[Table 7](#) illustrates the F1 scores of different aggregation methods for the Random method. In general, the OptAggr method can also effectively improve the model utility of Random compared to MeanAggr and MajAggr.

Table 6: 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. Best results are highlighted in bold.

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

Table 7: Comparison of F1 scores for different aggregation methods. Note that the Scratch method does not need aggregation. Best results are highlighted in bold.

Dataset/ Model		Scratch	MeanAggr	Random MajAggr	OptAggr
Cora	GAT	0.823 ± 0.006	0.649 ± 0.006	0.638 ± 0.010	0.726 ± 0.004
	GCN	0.739 ± 0.003	0.337 ± 0.006	0.188 ± 0.004	0.509 ± 0.009
	GIN	0.787 ± 0.013	0.760 ± 0.030	0.702 ± 0.033	0.766 ± 0.021
	SAGE	0.824 ± 0.004	0.583 ± 0.009	0.572 ± 0.012	0.682 ± 0.013
Citeseer	GAT	0.691 ± 0.015	0.502 ± 0.012	0.507 ± 0.016	0.631 ± 0.015
	GCN	0.493 ± 0.006	0.263 ± 0.014	0.157 ± 0.011	0.277 ± 0.009
	GIN	0.587 ± 0.031	0.611 ± 0.028	0.540 ± 0.056	0.626 ± 0.022
	SAGE	0.668 ± 0.013	0.519 ± 0.024	0.536 ± 0.026	0.623 ± 0.014
Pubmed	GAT	0.851 ± 0.004	0.852 ± 0.001	0.851 ± 0.002	0.857 ± 0.002
	GCN	0.748 ± 0.017	0.484 ± 0.004	0.207 ± 0.000	0.551 ± 0.005
	GIN	0.837 ± 0.015	0.859 ± 0.003	0.855 ± 0.003	0.856 ± 0.003
	SAGE	0.874 ± 0.003	0.854 ± 0.002	0.852 ± 0.003	0.857 ± 0.002
CS	GAT	0.919 ± 0.004	0.880 ± 0.001	0.877 ± 0.001	0.882 ± 0.000
	GCN	0.903 ± 0.006	0.644 ± 0.002	0.528 ± 0.001	0.706 ± 0.008
	GIN	0.867 ± 0.005	0.856 ± 0.006	0.839 ± 0.004	0.858 ± 0.005
	SAGE	0.932 ± 0.004	0.896 ± 0.005	0.896 ± 0.003	0.905 ± 0.004
Physics	GAT	0.955 ± 0.005	0.917 ± 0.001	0.915 ± 0.001	0.920 ± 0.002
	GCN	0.947 ± 0.002	0.597 ± 0.001	0.533 ± 0.001	0.747 ± 0.010
	GIN	0.934 ± 0.003	0.923 ± 0.002	0.916 ± 0.001	0.921 ± 0.002
	SAGE	0.956 ± 0.005	0.712 ± 0.003	0.717 ± 0.002	0.823 ± 0.011