Adaptive Graph Unlearning

Pengfei Ding, **Yan Wang***, **Guanfeng Liu**, **Jiajie Zhu**
Macquarie University, Sydney, Australia
{pengfei.ding, yan.wang, guanfeng.liu}@mq.edu.au, jiajie.zhu1@students.mq.edu.au

Abstract

Graph unlearning, which deletes graph elements such as nodes and edges from trained graph neural networks (GNNs), is crucial for real-world applications where graph data may contain outdated, inaccurate, or privacy-sensitive information. However, existing methods often suffer from (1) incomplete or over unlearning due to neglecting the distinct objectives of different unlearning tasks, and (2) inaccurate identification of neighbors affected by deleted elements across various GNN architectures. To address these limitations, we propose AGU, a novel Adaptive Graph Unlearning framework that flexibly adapts to diverse unlearning tasks and GNN architectures. AGU ensures the complete forgetting of deleted elements while preserving the integrity of the remaining graph. It also accurately identifies affected neighbors for each GNN architecture and prioritizes important ones to enhance unlearning performance. Extensive experiments on seven real-world graphs demonstrate that AGU outperforms existing methods in terms of effectiveness, efficiency, and unlearning capability.

1 Introduction

Machine unlearning (MU) [Sekhari *et al.*, 2021] has emerged as a pivotal step towards responsible AI, enabling the efficient removal of irrelevant, inaccurate, or privacy-sensitive data from trained models. While most MU methods focus on image or text data in Euclidean space, real-world data often appear in the form of graphs, prompting growing interest in unlearning for graph-based models, particularly graph neural networks (GNNs). For instance, fraudulent accounts in financial networks and outdated relations in knowledge graphs should be removed from their corresponding trained GNNs to prevent erroneous predictions. A naive approach is to retrain a new GNN on the updated graph, but this is costly and impractical for frequent unlearning requests. To address this issue, graph unlearning (GU) [Fan *et al.*, 2025] has been proposed to efficiently eliminate the influence of specific graph elements (e.g., nodes and edges) from trained GNNs. Because the effectiveness of GNNs relies on message propagation across local neighborhoods, effective GU requires not only forgetting information related to the *unlearning elements* (e.g., deleted nodes or edges in Fig. 1), but also minimizing their impact on neighboring nodes, referred to as *affected neighbors* (e.g., ego nodes, 1- and 2-hop neighbors in Fig. 1).

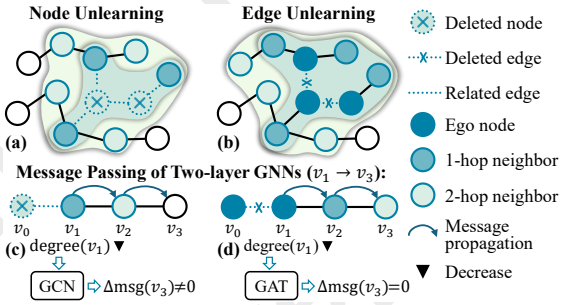

Figure 1: Examples of graph unlearning tasks. The deletion of node v_0 or edge (v_0, v_1) affects v_3 in GCN but not in GAT.

Recently, a wide range of GU studies [Wu *et al.*, 2023a; Tan *et al.*, 2024] focus on developing general frameworks that can address various GU tasks (e.g., node and edge unlearning) while supporting different backbone GNN architectures (e.g., GCN [Kipf and Welling, 2016] and GAT [Velickovic *et al.*, 2017]). However, these studies either overlook the distinct objectives of different GU tasks, or neglect the unique message-passing mechanisms of various GNN architectures, leading to two critical limitations as follows.

Limitation 1: Ineffective Forgetting of Unlearning Elements. To remove information about deleted elements from a trained GNN, a series of GU methods [Li *et al.*, 2024b; Kolipaka *et al.*, 2024] adopt a common MU strategy that penalizes the trained GNN for correctly predicting labels associated with deleted elements. However, this strategy is not universally effective across all GU tasks and often results in incomplete or over unlearning: (1) *Incomplete node-unlearning:* These methods input the remaining graph and deleted nodes into the trained GNN to generate labels for unlearning. However, since deleted nodes become isolated after removal, unlearning them in isolation only forgets their intrinsic attributes (e.g., node features), while ignoring the need to forget their original connections (e.g., related edges

*Corresponding author

Methods	Backbone GNNs	Node Unlearn.	Edge Unlearn.
Delete (2023)	GCN, GAT, GIN	K+1 hops	K hops
CEU (2023)	GCN, GIN, SAGE	-	K hops
GIF (2023)	GCN, SGC, GAT, GIN	K+1 hops	K hops
IDEA (2024)	GCN, SGC, GAT, GIN	K+1 hops	K hops
MEGU (2024)	GCN, SGC, GAT, GIN	K hops	K hops
Cognac (2024)	GCN, GAT	K+1 hops	K hops
ETR (2025)	GCN, GAT	K hops	K-1 hops
AGU (Ours)	GCN, SGC	K+1 hops	K hops
	GIN, GAT, SAGE	K hops	K-1 hops

Table 1: A summary of the affected ranges in recent GU studies. K denotes the number of layers in the trained GNN.

in Fig. 1(a)). (2) *Over edge-unlearning*: These methods transform edge-level GU tasks into node unlearning scenarios. Specifically, nodes connected to deleted edges (referred to as *ego nodes*, as shown in Fig. 1(b)) are treated as unlearning elements, while the topology of the remaining (undeleted) edges is preserved. However, this transformation inadvertently leads to the unlearning of these ego nodes' features, which should instead be retained in the remaining graph.

Limitation 2: Inaccurate Identification of Affected Neighbors. Existing GU methods have demonstrated that unlearning performance is heavily influenced by the identification of affected neighbors [Wu *et al.*, 2023b; Dong *et al.*, 2024]. However, these methods uniformly set the same affected range for all backbone GNNs. This setting overlooks the fact that different GNN architectures (e.g., GCN and GAT) employ distinct message-passing mechanisms, resulting in different degrees of influence on neighbors in node and edge unlearning. Table 1 compares the affected ranges used in existing methods with those identified in our study.

Figs. 1(c) and 1(d) illustrate why the affected ranges differ across various backbone GNNs. Consider the message-passing process of node v_3 in a two-layer GNN, where v_3 aggregates messages from its neighbors within two hops (e.g., node features of v_1 and v_2). If node v_0 or edge (v_0, v_1) is deleted, the degree of v_1 decreases. Since GCN normalizes neighbor features based on node degrees, this degree change affects the message aggregated by v_3, i.e., $\Delta\,\mathrm{msg}(v_3)\neq 0$. In contrast, GAT aggregates neighbor information without degree-based normalization, leaving the message aggregated at v_3 unaffected, i.e., $\Delta\,\mathrm{msg}(v_3)=0$. Consequently, the affected range in GAT is one hop smaller than that in GCN. Although some GU methods [Li *et al.*, 2024b] propose strategies to select important affected neighbors, they overlook the different impact of degree changes on various GNNs, limiting their ability to accurately identify truly affected neighbors.

Our Work. In this paper, we propose an **A**daptive **G**raph **U**nlearning framework (AGU) that flexibly adapts to various GU tasks and backbone GNNs. To address **Limitation 1**, we propose the *task-adaptive element forgetting*, which contains two basic unlearning modules that can be combined to effectively handle GU tasks at the node, edge, and feature levels. To address **Limitation 2**, we propose a *GNN-adaptive neighbor selection strategy*, which accurately identifies affected neighbors for each backbone GNN, and prioritizes highly affected neighbors across different GU tasks.

Contributions. (1) *Affected neighbor correction*. We identify and rectify inaccuracies in affected neighbor identification, a longstanding issue in the GU literature. (2) *Adaptive GU framework*. We propose a novel framework that flexibly adapts to various GU tasks and backbone GNNs. (3) *Universal selection strategy*. We propose a novel neighbor selection strategy that can be integrated into existing GU methods, improving both effectiveness and efficiency. (4) *SOTA performance*. Extensive experiments on seven real-world graphs show that AGU outperforms state-of-the-art methods in terms of effectiveness, efficiency, and unlearning capability.

2 Preliminaries

2.1 Notations and Background

Consider a graph $\mathcal{G}=\{\mathcal{V},\mathcal{E},\mathcal{X}\}$ with $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges. Each node $v_i \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_i \in \mathcal{X}$. In this paper, we focus on the node classification task. The training set $\mathcal{D}_0$ contains N samples $\{z_1, z_2, \ldots, z_N\}$, each annotated with a label $y \in \mathcal{Y}$. The objective is to train a GNN model f to predict the label of a given node $v \in \mathcal{V}$.

2.2 Graph Unlearning

In this part, we provide an overview of recent advances in GU. Early studies primarily focus on specific unlearning tasks and backbone GNN architectures. For instance, GraphEraser [Chen *et al.*, 2022] and GUIDE [Wang *et al.*, 2023] partition the graph into multiple shards and process the affected ones, but they only support node unlearning tasks. GraphEditor [Cong and Mahdavi, 2022], SGCunlearn [Chien *et al.*, 2023], and ScaleGUN [Yi and Wei, 2025] offer closed-form solutions with theoretical guarantees but are limited to linear GNNs. Recent advancements [Li *et al.*, 2024a; Yang *et al.*, 2025] extend GU to a wider range of tasks and backbone GNNs. For example, Delete [Cheng *et al.*, 2023] proposes general operators to remove deleted elements, while GIF [Wu *et al.*, 2023a] and IDEA [Dong *et al.*, 2024] estimate parameter changes for various GU tasks. MEGU [Li *et al.*, 2024b] improves performance by identifying highly affected neighbors. However, these methods either overlook the distinct objectives of different GU tasks or inaccurately identify affected neighbors, leading to suboptimal performance.

2.3 Graph Unlearning Formalization

Given a GNN model $f_{\mathcal{G}}$ trained on a graph $\mathcal{G}$, and an unlearning request $\Delta\mathcal{G}=\{\Delta\mathcal{V},\Delta\mathcal{E},\Delta\mathcal{X}\}$, the goal of GU is to derive a new GNN model $\hat{f}$ that closely approximates $f_{\mathcal{G}\backslash\Delta\mathcal{G}}$, which represents the model retrained from scratch on the remaining graph $\mathcal{G}\backslash\Delta\mathcal{G}$. GU tasks can be classified into three types: **Node Unlearning:** $\Delta\mathcal{G}=\{\Delta\mathcal{V},\emptyset,\emptyset\}$, where $\Delta\mathcal{V}$ is the set of nodes to be unlearned. **Edge Unlearning:** $\Delta\mathcal{G}=\{\emptyset,\Delta\mathcal{E},\emptyset\}$, where $\Delta\mathcal{E}$ is the set of edges to be unlearned. **Feature Unlearning:** $\Delta\mathcal{G}=\{\emptyset,\emptyset,\Delta\mathcal{X}\}$, where $\Delta\mathcal{X}$ denotes the set of features to be unlearned, which will be replaced with zeros.

3 Methodology

3.1 Framework Overview

The overall framework of AGU is illustrated in Fig. 2. To address incomplete or over unlearning across different GU tasks

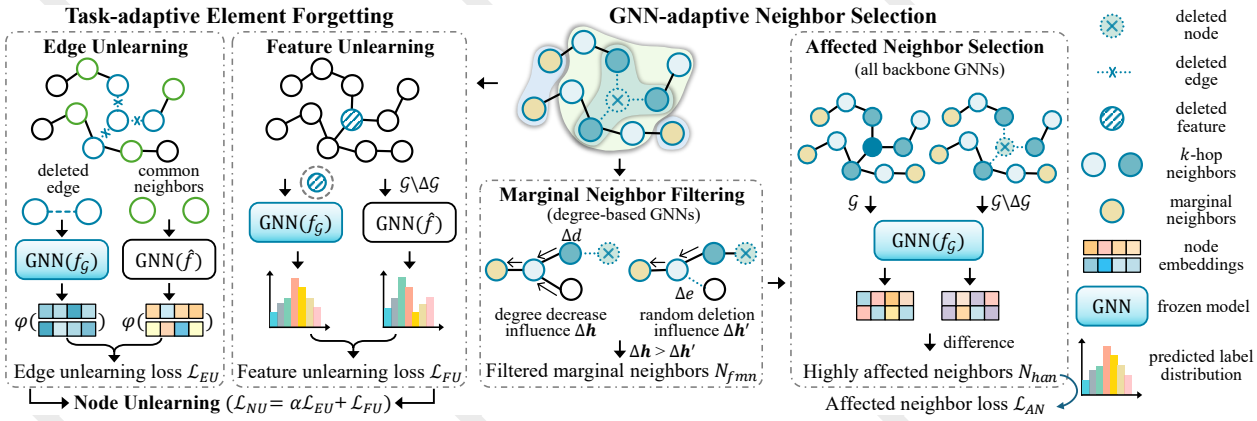

Figure 2: The overall architecture of our proposed AGU framework.

and accurately identify affected neighbors, AGU includes two key components: (1) *Task-adaptive element forgetting*, which contains two basic unlearning modules that can be combined to comprehensively forget information at the node, edge, and feature levels, respectively. (2) *GNN-adaptive neighbor selection*, which determines the accurate affected range for different GNNs and identifies highly affected neighbors. These components empower AGU to perform specific strategies for different GU tasks while effectively mitigating the influence on affected neighbors across various GNN architectures.

3.2 Task-adaptive Element Forgetting

GU tasks at the node, edge, and feature levels typically require distinct unlearning strategies, making it complex to address tasks across different levels. However, GU tasks at different levels share commonalities that can help reduce redundant unlearning processes and enable flexible adaptation to complex scenarios. In this section, we design two basic unlearning modules: *edge connection unlearning* and *individual feature unlearning*, which effectively address edge and feature unlearning, respectively. Moreover, their combination provides a comprehensive solution for node unlearning.

Edge Connection Unlearning. Given a GNN $f_{\mathcal{G}}$ trained on a graph $\mathcal{G}$ and an edge unlearning task $\Delta\mathcal{G}=\{\emptyset,\Delta\mathcal{E},\emptyset\}$, we aim to adjust $f_{\mathcal{G}}$ to obtain an unlearned model $\hat{f}$. Ideally, $\hat{f}$ should be unable to recognize whether a deleted edge $e_{uv}\in\Delta\mathcal{E}$ was originally part of $\mathcal{G}$. A common approach is to use the *deleted edge consistency* loss ($\mathcal{L}_{DEC}$) [Cheng *et al.*, 2023], which minimizes the difference between the representations of deleted edges and those of randomly selected node pairs:

$$\mathcal{L}_{DEC}=\mathrm{dis}(\{\varphi(\hat{\mathbf{h}}_u,\hat{\mathbf{h}}_v)|e_{uv}\in\Delta\mathcal{E}\},\{\varphi(\mathbf{h}_p,\mathbf{h}_q)|p,q\in_R\mathcal{V}\}), \tag{1}$$

where $\hat{\mathbf{h}}$ and $\mathbf{h}$ represent the node embeddings output by $\hat{f}$ and $f_{\mathcal{G}}$, respectively. $\varphi(\cdot)$ aggregates the embeddings of two nodes (e.g., via concatenation). $\mathrm{dis}(\cdot)$ is a function positively correlated with the difference. $\in_R$ indicates random selection.

However, enforcing the end nodes of edges in $\Delta\mathcal{E}$ to approximate random node pairs may not accurately reflect the actual post-deletion *homophily*, where connected nodes in a graph typically share similar attributes or belong to the same class [Zhu *et al.*, 2020]. Therefore, even after the removal of edges in $\Delta\mathcal{E}$, their end nodes are expected to retain homophily and should not be treated as randomly paired nodes.

To preserve the underlying homophily between nodes $(u,v)\in\Delta\mathcal{E}$, we refine Eq. (1) by proposing a new candidate set $\mathcal{V}_{uv}^k$ for selecting comparable node pairs. Specifically, $\mathcal{V}_{uv}^k$ consists of common neighbors within the k-hop range of u and v, i.e., $\mathcal{V}_{uv}^k=\mathcal{V}_u^k\cap\mathcal{V}_v^k$, where $\mathcal{V}_v^k$ denotes the set of nodes within k hops of v. We replace $\mathcal{V}$ in Eq. (1) with $\mathcal{V}_{uv}^k$ to define our edge unlearning loss, denoted as $\mathcal{L}_{EU}$. If $|\mathcal{V}_u^k\cap\mathcal{V}_v^k|<2$, we set $\mathcal{V}_{uv}^k$ as the union of the two sets, i.e., $\mathcal{V}_u^k\cup\mathcal{V}_v^k$.

Individual Feature Unlearning. Since $f_{\mathcal{G}}$ is trained using node labels as supervision signals, feature unlearning ($\Delta\mathcal{G}=\{\emptyset,\emptyset,\Delta\mathcal{X}\}$) requires the unlearned GNN $\hat{f}$ to dissociate the node features in $\Delta\mathcal{X}$ from their corresponding labels predicted by $f_{\mathcal{G}}$. Existing methods [Li *et al.*, 2024b; Kolipaka *et al.*, 2024] typically achieve this through *gradient ascent*, which penalizes $\hat{f}$ for predicting the same labels as $f_{\mathcal{G}}$ for nodes associated with $\Delta\mathcal{X}$.

However, since GNNs aggregate neighbor information to generate representations for prediction, directly applying gradient ascent can lead to the loss of valuable neighbor information, potentially degrading the performance of the unlearned GNN $\hat{f}$. For example, if $f_{\mathcal{G}}$ employs a weighted aggregation strategy that assigns lower weights to the target node v but higher weights to its neighbors, v's own features contribute less to its prediction. In this case, applying gradient ascent to v does not entirely cut the relationship between its features and the corresponding label. Instead, it may inadvertently degrade the prediction performance of other nodes that share similar neighbor information with v.

To address this issue, we propose a novel feature unlearning strategy that adaptively forgets deleted features while preserving valuable neighbor information. Specifically, we freeze the trained GNN $f_{\mathcal{G}}$ and input individual nodes without edge connections (i.e., $\mathcal{G}\backslash\mathcal{E}$) into $f_{\mathcal{G}}$ to obtain their independent label distributions $\mathbf{y}'$ as self-supervised signals. Then, we enforce the unlearned GNN $\hat{f}$ to output a distinct label distribution $\hat{\mathbf{y}}$ on the remaining graph $\mathcal{G}\backslash\Delta\mathcal{G}$:

$$\mathcal{L}_{FU}=-\sum\nolimits_{u\in\mathcal{V}_{\Delta\mathcal{X}}}\mathcal{L}_{KL}(\mathbf{y}'_u,\hat{\mathbf{y}}_u), \tag{2}$$

where $\mathcal{V}_{\Delta\mathcal{X}}$ denotes the set of nodes whose features are to be unlearned. $\mathcal{L}_{KL}(\cdot)$ is the Kullback-Leibler divergence. The rationale behind Eq. (2) is: (1) If $f_{\mathcal{G}}$ relies more on u's features for prediction, $\mathbf{y}'_u$ should approximate $\mathbf{y}_u$. Thus, enforcing a distributional difference between $\mathbf{y}'_u$ and $\hat{\mathbf{y}}_u$ encourages $\hat{f}$ to forget the correlation between u's features and the prediction made by $f_{\mathcal{G}}$. (2) If $f_{\mathcal{G}}$ relies more on u's neighbor information for prediction, $\mathbf{y}'_u$ should differ from $\mathbf{y}_u$. Thus, enforcing a distributional difference between $\mathbf{y}'_u$ and $\hat{\mathbf{y}}_u$ can forget u's own features while preserving valuable neighbor information.

Node Unlearning. A simple yet common strategy to forget deleted nodes $\Delta\mathcal{V}$ in a trained node classification model $f_{\mathcal{G}}$ is to apply a reverse cross-entropy loss: $-\sum_{u\in\Delta\mathcal{V}}\mathcal{L}_{CE}(y_u,\hat{y}_u)$, where y_u is the prediction of $f_{\mathcal{G}}$ on $\mathcal{G}$, and $\hat{y}_u$ is the prediction of $\hat{f}$ on the remaining graph $\mathcal{G}\backslash\Delta\mathcal{V}$. However, since the nodes in $\Delta\mathcal{V}$ become isolated after deletion, $\hat{y}_u$ depends only on u's own features and is no longer influenced by its original neighbors. Thus, applying this loss function cannot ensure $\hat{f}$ fully forgets the original connections between u and its neighbors.

For comprehensive node unlearning, we adopt a new perspective by transforming the task from $\{\Delta\mathcal{V},\emptyset,\emptyset\}$ to $\{\emptyset,\Delta\mathcal{E},\Delta\mathcal{X}\}$, where $\Delta\mathcal{E}$ represents the edges connected to nodes in $\Delta\mathcal{V}$, and $\Delta\mathcal{X}$ denotes node features of $\Delta\mathcal{V}$. This transformation allows us to combine our proposed edge and feature unlearning modules to derive the node unlearning loss $\mathcal{L}_{NU}$:

$$\mathcal{L}_{NU}=\alpha\mathcal{L}_{EU}+\mathcal{L}_{FU}, \tag{3}$$

where α is a loss coefficient hyper-parameter.

3.3 GNN-adaptive Neighbor Selection

Existing studies [Wu *et al.*, 2023b; Li *et al.*, 2024b] show that unlearning performance heavily depends on the identification of affected neighbors. Moreover, selecting highly affected neighbors can improve both the effectiveness and efficiency of unlearning. In this section, we address inaccuracies in existing methods for identifying affected neighbors and propose a novel GNN-adaptive selection strategy.

Accurate Affected Neighbor Identification. Given a k-layer trained GNN $f_{\mathcal{G}}$ on graph $\mathcal{G}$ and an unlearning request $\Delta\mathcal{G}$, existing GU methods identify the affected neighbors $\mathcal{N}_{aff}$ either directly based on the number of layers (i.e., k) [Dong *et al.*, 2024; Li *et al.*, 2024b] or through k-step normalized propagation [Wu *et al.*, 2023a; Chen *et al.*, 2023] as follows:

$$\mathcal{N}_{aff}=\left\{i\mid\Delta\mathbf{H}(\hat{\mathbf{A}},\mathbf{A})_i\neq 0,\ i\in\mathcal{G}\setminus\Delta\mathcal{G}\right\}, \tag{4}$$

$$\Delta\mathbf{H}(\hat{\mathbf{A}},\mathbf{A})=[(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}})^k-(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^k]\mathbf{X}, \tag{5}$$

where $\mathbf{A}$ and $\hat{\mathbf{A}}$ are the adjacency matrices of $\mathcal{G}$ and $\mathcal{G}\backslash\Delta\mathcal{G}$, respectively, with $\mathbf{D}$ and $\hat{\mathbf{D}}$ as their corresponding degree matrices. $\Delta\mathbf{H}(\hat{\mathbf{A}},\mathbf{A})$ quantifies the propagation change before and after deleting unlearning elements. Nodes outside $\Delta\mathcal{G}$ that satisfy $\Delta\mathbf{H}(\hat{\mathbf{A}},\mathbf{A})\neq 0$ are considered affected neighbors.

However, this identification method is limited to backbone GNNs that incorporate node degrees in message passing (e.g., GCN, SGC [Wu *et al.*, 2019]), which we refer to as *degree-based* GNNs. In contrast, *degree-free* GNNs (e.g., GAT, GIN [Xu *et al.*, 2018]) aggregate neighbor information without relying on node degrees, leading to a different set of affected neighbors. As discussed in Section 1, degree-free GNNs typically influence fewer nodes than their degree-based counterparts. We validate this difference through experiments (see Appendix [Ding *et al.*, 2025] for details). This influence discrepancy suggests that existing methods incorrectly set the same range of affected neighbors for all backbone GNNs. To address this issue, we summarize the correct affected range for popular GNNs in Table 1, and propose a simple yet general strategy to accurately identify affected neighbors without analyzing the architecture of the trained GNN $f_{\mathcal{G}}$:

$$\mathcal{N}_{ac}=\{v\mid f_{rand}(\mathcal{G})_v\neq f_{rand}(\mathcal{G}\setminus\Delta\mathcal{G})_v, v\in\mathcal{G}\setminus\Delta\mathcal{G}\}, \tag{6}$$

where f_{rand} denotes a randomly initialized GNN with same architecture as $f_{\mathcal{G}}$. Note that in practice, it is crucial to remove randomness from f_{rand}'s message-passing process to ensure accurate identification of affected neighbors.

Marginal Neighbor Filtering. For node and edge unlearning, degree-based GNNs affect an additional hop of neighbors compared to degree-free GNNs (as shown in Table 1). We refer to these extra affected neighbors as *marginal neighbors*. Note that for feature unlearning, since the graph structure remains unchanged after unlearning, the affected range is the same for all backbone GNNs.

We first analyze the impact of unlearning on marginal neighbors compared to other affected neighbors. Consider the message-passing process in a simple and classic degree-based GNN, SGC. The output for node i at the l-th layer is:

$$\mathbf{h}_i^{(l)}=\sum\nolimits_{j\in\mathcal{N}_i}\frac{\mathbf{h}_j^{(l\text{-}1)}}{\sqrt{d_i d_j}}=\sum\nolimits_{j\in\mathcal{N}_i^l}\frac{\mathbf{x}_j}{\prod_{t=1}^{l}\sqrt{d_{j_{t\text{-}1}}d_{j_t}}}, \tag{7}$$

where $\mathcal{N}_i^l$ denotes the set of i's neighbors at the l-th hop. d_i is the degree of node i. The term $\prod_{t=1}^{l}\sqrt{d_{j_{t\text{-}1}}d_{j_t}}$ represents the degree normalization over l layers.

In node unlearning, if i is a marginal neighbor, the deletion of unlearning nodes causes only a minor degree change, Δd, for some of i's neighbors l hops away (i.e., d_{j_l}), while leaving $\mathcal{N}_i^l$ unchanged. In contrast, for other affected neighbors, message passing is often obstructed due to a reduction in $\mathcal{N}_i^l$, leading to a more significant change in $\Delta\mathbf{h}$. Therefore, marginal neighbors tend to be less influenced than other affected nodes. To filter out marginal neighbors with minimal impact from degree changes, we propose the following filtering strategy: *A marginal neighbor should be considered only if the degree change in its neighbors exceeds the effect of randomly removing one of its neighbors:*

$$\mathcal{N}_{fmn}=\{i\mid|\Delta\mathbf{H}(\hat{\mathbf{A}},\mathbf{A})_i|-|\Delta\mathbf{H}(\mathbf{A}',\mathbf{A})_i|>\theta\}, \tag{8}$$

where $\mathbf{A}'$ is the adjacency matrix obtained after randomly deleting an edge within the k-hop neighborhood of each node in $\Delta\mathcal{G}$. θ represents the difference threshold parameter.

Affected Neighbor Selection. In this part, we propose a general strategy to select important affected neighbors. Existing GU methods typically employ *inverse feature propagation* on the original graph $\mathcal{G}$ to identify highly affected neighbors. Specifically, they perform feature propagation (e.g., $\mathbf{A}^k\mathbf{X}$) using the original feature matrix $\mathbf{X}$ and its inverse counterpart $\mathbf{X}'$=**1**-$\mathbf{X}$. By comparing propagation results for the same

Task	Method	Cora		Citeseer		PubMed		Photo		Computer		CS		Flickr	
	Bone	GCN	GIN	GCN	GIN	GCN	GIN	GCN	GIN	GCN	GIN	GCN	GIN	GCN	GIN
Node	Retrain	86.1±.4	83.5±.7	74.6±.3	73.6±.7	88.6±.7	84.9±.4	91.8±2	85.9±.4	83.7±.7	77.8±.8	92.1±.2	90.9±.3	49.1±.1	47.3±.4
	Delete	82.5±.5	76.2±1	71.4±.5	67.4±1	84.6±.2	76.1±3	89.2±.4	83.5±.0	84.2±.5	75.2±.0	91.4±.3	87.9±1	45.5±.4	40.4±.4
	GIF	84.5±.6	82.1±.7	73.2±.5	72.4±.7	86.5±.9	84.2±.3	89.7±.8	76.2±.6	83.5±.4	77.1±.3	91.7±.3	88.7±.4	47.4±.8	42.5±.5
	IDEA	80.2±2	82.3±.7	69.7±2	72.6±.7	82.3±3	84.3±.3	88.1±2	78.8±.4	83.3±.4	78.2±.2	91.6±.3	88.4±.4	47.1±.2	42.5±.6
	MEGU	85.4±.6	83.6±.3	74.8±.5	73.1±.7	87.1±.2	84.9±.3	91.9±.4	80.6±.4	85.8±.3	78.3±.2	92.2±.3	89.4±.4	49.3±.2	44.6±.3
	UTU	84.2±.0	80.5±.0	72.7±.0	70.6±.0	86.7±.0	83.6±.0	89.5±.0	83.8±.0	80.8±.0	77.3±.0	91.4±.0	88.3±.0	47.1±.0	41.8±.0
	ETR	84.9±.7	83.4±.8	74.2±.6	73.3±.6	87.3±.9	85.1±.3	92.3±.3	74.2±.6	85.7±.3	70.6±.2	91.8±.2	90.4±.5	48.1±.8	40.2±.2
	Cognac	85.1±.5	82.9±.8	74.0±.6	73.2±.9	87.4±.1	85.2±.1	92.2±.2	84.1±.4	85.2±.5	78.4±.3	92.1±.1	90.9±.2	48.7±.1	43.4±.3
	AGU	**85.9±.3**	**84.3±.2**	**76.2±.5**	**73.8±.7**	**88.4±.2**	**85.5±.3**	**92.6±.1**	**85.7±.3**	**86.3±.4**	**79.3±.5**	**92.8±.3**	**91.7±.4**	**50.2±.2**	**46.8±.5**
	Bone	SAGE	GAT	SAGE	GAT	SAGE	GAT	SAGE	GAT	SAGE	GAT	SAGE	GAT	SAGE	GAT
Edge	Retrain	86.8±.3	86.5±.2	78.0±.6	75.1±.2	89.1±.4	84.9±.4	94.3±2	90.1±.2	88.4±.8	84.6±.7	94.1±.2	91.3±.3	49.8±.8	49.2±.4
	Delete	84.7±.4	84.3±.7	73.7±.5	73.2±.1	88.9±.3	82.6±.9	92.6±.6	85.4±.4	86.5±.6	78.7±.4	92.8±.4	91.1±.2	47.2±.7	46.2±.5
	GIF	86.1±.1	85.2±.4	77.6±.5	73.7±.7	89.3±.0	85.4±.3	93.6±.6	87.8±.7	86.3±.4	78.2±.3	93.6±.1	90.7±.2	48.7±.4	46.7±.3
	IDEA	85.6±.7	85.2±.4	76.7±.5	73.8±.7	88.7±.3	85.2±.3	91.7±.5	87.6±.7	84.5±.7	78.3±.7	92.5±.2	90.9±.2	47.8±.8	46.7±.5
	MEGU	85.9±.3	86.2±.1	78.1±.1	74.1±.8	89.2±.3	86.4±.6	94.3±.2	89.7±.3	87.5±.1	82.8±.4	94.2±.1	92.1±.3	49.6±.3	45.8±.6
	UTU	83.8±.0	82.2±.0	77.5±.0	71.6±.0	88.2±.0	80.3±.0	93.7±.0	85.9±.0	87.1±.0	73.7±.0	94.1±.0	89.9±.0	47.5±.0	45.1±.0
	ETR	84.2±.2	85.5±.5	73.2±.3	74.3±.3	87.5±.2	86.4±.1	94.1±.6	90.9±.4	87.3±.4	83.5±.3	94.1±.2	91.1±1	49.5±,6	47.1±.7
	Cognac	83.9±.6	85.9±.6	77.9±.5	73.6±.4	88.4±.2	86.3±.2	92.5±.2	90.4±.5	87.6±.3	81.9±.5	93.9±.2	91.8±.2	49.4±.2	47.6±.2
	AGU	**87.4±.2**	**86.9±.1**	**78.6±.2**	**75.5±.3**	**90.2±.3**	**87.1±.4**	**95.3±.5**	**92.3±.6**	**88.7±.4**	**85.4±.3**	**94.5±.2**	**92.7±.2**	**50.3±.5**	**48.6±.1**

Table 2: Performance comparison in terms of F1 score. The best results are highlighted in **bold**, while the second-best results are underlined.

node, they identify highly affected neighbors as those with significant differences. However, this approach overlooks variations across different GU tasks and GNN architectures.

To address this limitation, we propose a task- and GNN-adaptive neighbor selection module. The idea is to utilize the trained GNN to directly identify important affected neighbors. Specifically, we freeze the trained model $f_{\mathcal{G}}$ and input $\mathcal{G}$ and $\mathcal{G}\backslash\Delta\mathcal{G}$ into $f_{\mathcal{G}}$. We then compare the differences in the node representations output by $f_{\mathcal{G}}$ for these two graphs:

$$\mathrm{diff}(v) = \mathrm{dis}(f_{\mathcal{G}}(\mathcal{G} \setminus \Delta\mathcal{G})_v, f_{\mathcal{G}}(\mathcal{G})_v), \tag{9}$$

where $f_{\mathcal{G}}(\cdot)_v$ denotes the representation of node v output by $f_{\mathcal{G}}$. $\mathrm{dis}(\cdot)$ is a function that positively correlates with the difference. Intuitively, nodes with higher $\mathrm{diff}(\cdot)$ values are more significantly affected by the unlearning elements. Thus, we select the top-k_{ans} nodes with the highest $\mathrm{diff}(\cdot)$ values as important affected neighbors, denoted as $\mathcal{N}_{han}$. For degree-based GNNs, we first apply marginal neighbor filtering, then use the remaining affected neighbors for further selection.

Optimization Objective. Unlike the labels of unlearning elements in $\Delta\mathcal{G}$, which are intended to be forgotten in the unlearned GNN $\hat{f}$, the predictive accuracy for the highly affected neighbors $\mathcal{N}_{han}$ should be preserved in $\hat{f}$. To this end, we follow [Li *et al.*, 2024b] and treat the predictions of the original model ($f_{\mathcal{G}}$) as self-supervised signals for $\mathcal{N}_{han}$. Specifically, we freeze the parameters of $f_{\mathcal{G}}$ and define a cross-entropy loss $\mathcal{L}_{AN}=\sum_{u\in\mathcal{N}_{han}}\mathcal{L}_{CE}(y_u,\hat{y}_u)$, where y_u is the prediction of $f_{\mathcal{G}}$ on $\mathcal{G}$, and $\hat{y}_u$ is the prediction of $\hat{f}$ on $\mathcal{G}\backslash\Delta\mathcal{G}$.

Finally, by combining $\mathcal{L}_{AN}$ with the loss on the unlearning elements $\Delta\mathcal{G}$, the overall loss is formulated as:

$$\mathcal{L} = \mathcal{L}_{EF} + \mathcal{L}_{AN}, \tag{10}$$

where $\mathcal{L}_{EF}$ is selected from $\{\mathcal{L}_{NU}, \mathcal{L}_{EU}, \mathcal{L}_{FU}\}$ based on the specific GU task.

4 Experiments

We conduct extensive experiments to answer the following research questions: **RQ1:** How does AGU perform compared to state-of-the-art methods? **RQ2:** How does each proposed module in AGU contribute to the overall performance? **RQ3:** Can our proposed neighbor selection strategies enhance the performance of existing methods? **RQ4:** How do different parameter settings influence AGU's performance?

4.1 Experimental Setup

Datasets. We select seven widely used datasets: Cora, Citeseer, PubMed [Yang *et al.*, 2016], Amazon-Photo, Amazon-Computers, Coauthor-CS [Shchur *et al.*, 2018], and Flickr [Zeng *et al.*, 2019]. The datasets are split following the guidelines of recent GU studies [Cheng *et al.*, 2023; Li *et al.*, 2024b], with 80% of the nodes used for training and 20% for testing. Detailed statistics and descriptions of these datasets are provided in the Appendix [Ding *et al.*, 2025].

Backbone GNNs and Baselines. We evaluate the adaptability of AGU using two **degree-based GNNs:** GCN [Kipf and Welling, 2016] and SGC [Wu *et al.*, 2019], and three **degree-free GNNs:** GAT [Velickovic *et al.*, 2017], SAGE [Hamilton *et al.*, 2017], and GIN [Xu *et al.*, 2018]. We compare AGU against **seven state-of-the-art baselines:** Delete [Cheng *et al.*, 2023], GIF [Wu *et al.*, 2023a], IDEA [Dong *et al.*, 2024], MEGU [Li *et al.*, 2024b], UTU [Tan *et al.*, 2024], ETR [Yang *et al.*, 2025], and Cognac [Kolipaka *et al.*, 2024]. Detailed descriptions of these GNNs and baselines are provided in the Appendix. For all methods, we set the embedding dimension to 64, and fix the number of GNN layers at 2. Baseline parameters are initialized using the values reported in the original papers and further fine-tuned for optimal performance. In AGU, the edge unlearning loss $\mathcal{L}_{EU}$ uses concatenation for $\varphi(\cdot)$ and mean-squared error for $\mathrm{dis}(\cdot)$, while Eq. (9) uses cosine similarity for $\mathrm{dis}(\cdot)$. The loss coefficient parameter α is set to 0.1. To ensure a fair comparison, each experiment is repeated 10 times and we report the average performance.

Unlearning Tasks. In our experiments, GU tasks are designed as follows: (1) **Node-level:** We randomly delete 5% of nodes from the training graph and remove their associated edges. (2) **Edge-level:** We randomly delete 5% of edges

Bone	Method	Cora		Citeseer		PubMed	
		F1	Time	F1	Time	F1	Time
SGC	Retrain	82.4±.1	5.49	72.2±.1	7.38	83.5±.1	24.1
	Delete	81.6±.2	1.75	70.4±.3	1.18	82.5±.1	2.96
	GIF	82.6±.2	0.23	70.2±.2	0.24	83.1±.1	1.36
	IDEA	83.8±.4	0.24	61.5±.7	0.24	82.3±.9	1.42
	MEGU	84.3±.2	0.14	73.1±.2	0.18	82.9±.1	0.27
	ETR	83.9±.4	**0.01**	72.8±.1	**0.02**	84.1±.4	**0.03**
	Cognac	83.4±.2	0.84	72.9±.2	0.91	83.9±.1	1.63
	AGU	**85.1±.2**	0.07	**73.8±.2**	0.12	**84.7±.1**	0.13
GIN	Retrain	84.5±.2	10.5	72.3±.4	16.3	84.1±.4	42.6
	Delete	83.4±.3	2.53	68.4±1	2.36	76.8±3	2.72
	GIF	85.4±.1	0.26	72.2±.1	0.26	84.2±.3	2.30
	IDEA	85.3±.4	0.25	71.9±.7	0.26	84.2±.7	2.22
	MEGU	85.6±.2	0.15	72.9±.3	0.15	84.9±.1	0.24
	ETR	84.3±.3	**0.01**	73.2±.2	**0.02**	84.7±.3	**0.03**
	Cognac	84.3±.4	0.83	71.6±.8	0.69	85.1±.2	1.76
	AGU	**86.7±.2**	0.07	**73.9±.4**	0.08	**85.5±.3**	0.08

Table 3: Performance comparison of feature unlearning.

Bone	Method	Cora	Citeseer	PubMed	Photo	Computer	CS	Flickr
GCN	Retrain	8.47	13.5	35.7	21.6	49.3	74.2	172
	GIF	0.29	0.26	0.56	0.49	0.29	0.31	1.22
	IDEA	0.27	0.26	0.55	0.49	0.27	0.28	1.19
	ETR	**0.02**	**0.03**	**0.05**	**0.03**	**0.06**	0.14	**0.22**
	Delete	2.14	2.01	2.57	2.18	2.26	2.74	8.32
	Cognac	1.01	0.83	2.15	1.27	3.34	4.71	4.33
	MEGU	0.18	0.16	0.25	0.22	0.25	0.26	1.82
	AGU	0.11	0.09	0.10	0.11	0.10	**0.12**	0.31
GAT	Retrain	13.8	17.1	47.2	23.1	50.1	77.4	196
	GIF	0.69	0.68	1.47	0.93	2.73	0.71	1.14
	IDEA	0.65	0.65	1.46	0.95	2.69	0.63	1.11
	ETR	**0.04**	**0.04**	**0.06**	**0.03**	**0.08**	**0.12**	0.26
	Delete	3.23	2.62	3.59	2.32	2.87	2.77	9.13
	Cognac	1.57	1.38	2.42	1.54	3.83	6.85	2.21
	MEGU	0.30	0.30	0.44	0.32	0.36	0.37	0.98
	AGU	0.16	0.15	0.18	0.13	0.15	0.17	**0.24**

Table 4: Comparison of node unlearning runtime.

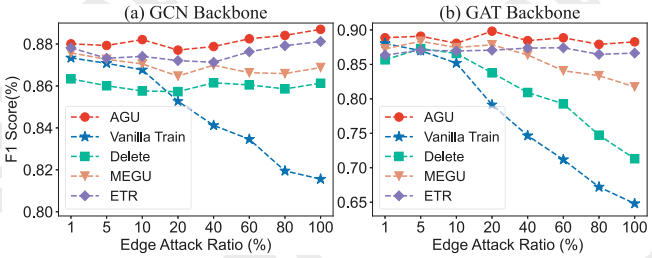

Figure 3: Edge attack performance on the Cora dataset.

from the training graph. (3) **Feature-level:** We randomly select 5% of nodes from the training graph and mask their full-dimensional features. Results for other unlearn ratios are provided in the Appendix. All methods are evaluated using the Micro-F1 score for node classification. To further evaluate the unlearning capability of GU methods, we adopt the widely used *Edge Attack* approach [Wu *et al.*, 2023a; Li *et al.*, 2024b]. Specifically, we add noisy edges into the training graph to mislead representation learning. Each added edge connects two nodes from different classes to maximize its adversarial effect. These noisy edges are then treated as unlearning elements. The unlearning capability of GU methods is evaluated by their ability to mitigate the negative impact of these noisy edges on predictive performance.

4.2 Performance Comparison (RQ1)

We compare the performance of all methods from three perspectives: *effectiveness*, *efficiency*, and *unlearning capability*.

Effectiveness. Tables 2 and 3 compare the node classification performance across various GU tasks and backbone GNNs. Notably, AGU consistently outperforms all baselines in F1-score across all settings. On average, AGU achieves a 1.37% improvement over the best-performing baseline in node unlearning (ranging from 0.35% to 4.93%), 1.21% in edge unlearning (0.31% to 2.28%), and 1.26% in feature unlearning (0.47% to 3.31%). These results demonstrate AGU's robustness across diverse GU tasks and backbone GNNs, highlighting the effectiveness of its task-adaptive element forgetting and GNN-adaptive neighbor selection. In contrast, existing methods adopt uniform unlearning strategies, leading to inferior overall performance. For instance, Delete, which adopts an edge-based unlearning strategy, performs well in edge-level GU tasks but struggles in other GU tasks. These findings further validate the necessity of designing task-specific unlearning strategies for different GU tasks.

Efficiency. We report the average unlearning time in Table 4. The results show that AGU outperforms other training-based methods (Delete, MEGU, and Cognac), achieving a speedup from 1.36× to 29×. Compared to training-free methods (GIF, IDEA, ETR), which directly adjust model parameters without training, AGU remains competitive, ranking second only to ETR in some cases. However, ETR requires additional time to extract subgraphs based on specific unlearning requests before initiating the unlearning process. Moreover, our experiments reveal that these training-free methods require careful parameter tuning to avoid model invalidation on the testing graph. In contrast, AGU ensures robust performance with minimal training epochs (20–30).

Unlearning Capability. Fig. 3 presents the node classification performance under different edge attack ratios. To provide a clear baseline for comparison, we introduce *vanilla train*, which retrains a new GNN directly on the noisy graph. The results demonstrate that AGU consistently outperforms other baselines and remains stable across varying attack ratios, particularly when using GAT as the backbone GNN. Notably, unlike Delete, which adopts a random node-pair strategy for contrastive edge unlearning, AGU leverages homophily consistency between connected nodes to more effectively distinguish noisy edges from real ones, thereby mitigating the impact of edge attacks.

4.3 Ablation Study (RQ2)

We conduct ablation studies by creating five AGU variants to evaluate the contributions of its key components: (1) **w/o Homo:** This variant replaces the homophily-based node-pair strategy with a random node-pair strategy for contrastive edge unlearning, allowing us to assess the effectiveness of the proposed edge unlearning module. (2) **w/o EU** and **w/o FU:** Since AGU combines the edge unlearning (EU) and feature

Bone	Method	Cora	Citeseer	PubMed	Photo	Computer
GCN	AGU	**85.94**	**76.22**	**88.38**	**92.61**	**86.34**
	w/o Homo	85.52	74.28	87.74	92.38	85.61
	w/o EU	85.07	73.81	87.52	92.06	85.48
	w/o FU	84.73	74.89	86.88	91.94	85.37
	w/o MNF	85.41	75.34	87.69	92.13	85.62
	w/o ANS	85.29	74.26	87.44	91.82	85.75
SAGE	AGU	**86.09**	**76.64**	**90.34**	**95.14**	**88.91**
	w/o Homo	85.42	75.71	90.03	94.78	88.27
	w/o EU	84.76	75.43	89.81	94.67	88.19
	w/o FU	85.75	75.49	90.01	94.69	87.72
	w/o ANS	85.43	75.24	89.98	94.42	88.16

Table 5: Node unlearning performance of AGU variants.

unlearning (FU) modules to address node unlearning tasks, these two variants independently evaluate the contributions of EU and FU to the overall unlearning performance. (3) **w/o MNF:** This variant removes the marginal neighbor filtering (MNF) module to evaluate its impact, particularly on degree-based GNNs (e.g., GCN). (4) **w/o ANS:** This variant excludes the affected neighbor selection (ANS) module to assess its influence on the unlearning process. The results in Table 5 show that AGU consistently outperforms all variants across all cases. This not only validates the importance of combining EU and FU for effective node unlearning, but also highlights the critical roles of MNF and ANS in identifying highly affected nodes for efficient and effective unlearning.

4.4 Strategy Generalizability (RQ3)

We evaluate the generalizability of our proposed neighbor selection strategies, MNF and ANS, by integrating them into existing GU methods: Delete, MEGU, and ETR. While MEGU and ETR have their own neighbor selection strategies, we replace them with MNF and ANS for comparison. In addition, we provide the "+AAN" variant, which selects accurate affected neighbors for degree-free GNNs (e.g., GAT). To reflect real-world scenarios with frequent but small-scale unlearning requests, we set the unlearn ratio to 1%. Our statistics show that in a 2-layer GNN, removing just 1% of nodes affects over 70% of the remaining nodes in the Photo and Computer datasets. We also conduct experiments on other datasets with various backbone GNNs and unlearn ratios (see Appendix). Our results demonstrate that MNF and ANS can be effectively integrated into existing GU frameworks, improving both effectiveness and efficiency across different datasets, backbone GNNs, and unlearn ratios. Specifically, the results in Table 6 show that (1) For degree-based GNNs (e.g., GCN), applying MNF or ANS consistently reduces unlearning time while preserving prediction performance in most cases. Combining both strategies further improves prediction performance and reduces unlearning time by 11% to 36%. (2) For degree-free GNNs (e.g., GAT), using ANS enhances prediction performance in most cases while achieving a comparable reduction in unlearning time (25% to 48%).

4.5 Parameter Study (RQ4)

We investigate the sensitivity of two key parameters θ and k_{ans} in AGU. The parameter θ controls the filtering of marginal neighbors ($\mathcal{N}_{fmn}$) when using degree-based GNNs as backbones, while k_{ans} determines the proportion of affected neighbors selected as highly affected neighbors ($\mathcal{N}_{han}$). As shown in Fig. 4, increasing θ generally improves unlearning performance. However, when θ becomes too large, the number of filtered marginal neighbors approaches zero, leading to a performance degradation. Thus, the optimal range for θ is between $5e-5$ and $5e-4$. For k_{ans}, a value around 40% can achieve a good trade-off between effectiveness and efficiency.

Bone	Method	Photo		Computer	
		F1	Time	F1	Time
GCN	MEGU	92.4±.3	0.34	85.8+.3	0.37
	+MNF	92.4±.7	0.29	85.1±.4	0.29
	+ANS	92.6±.6	0.30	85.6±.2	0.29
	+Both	**92.9±.2**	**0.28**	**86.0±.3**	**0.28**
	Delete	90.6±.3	1.85	83.9+.4	1.96
	+MNF	90.4±.7	1.81	84.3±.6	2.18
	+ANS	90.6±.5	1.79	84.9±.3	1.78
	+Both	**90.8±.2**	**1.63**	**85.2±.3**	**1.62**
	ETR	92.5±.3	0.07	85.8+.8	0.11
	+MNF	92.5±.8	0.07	85.2±.8	0.09
	+ANS	92.7±.4	0.07	85.8±.2	0.08
	+Both	**92.8±.3**	**0.06**	**85.9±.3**	**0.07**
GAT	MEGU	92.6±.4	0.48	86.0+.1	0.48
	+AAN	92.7±.4	0.31	**86.6±.5**	0.30
	+ANS	**93.1±.3**	**0.27**	86.4±.2	**0.25**
	Delete	90.9±.4	2.75	84.1+.3	2.74
	+AAN	90.9±.3	2.24	84.2±.5	2.43
	+ANS	**91.1±.4**	**1.98**	**84.7±.3**	**2.06**
	ETR	92.2±.7	0.09	85.3+.3	0.11
	+AAN	92.2±.8	0.06	85.1±.8	0.08
	+ANS	**92.7±.3**	**0.05**	**85.9±.5**	**0.07**

Table 6: Performance improvement of baselines. Unlearn ratio=1%.

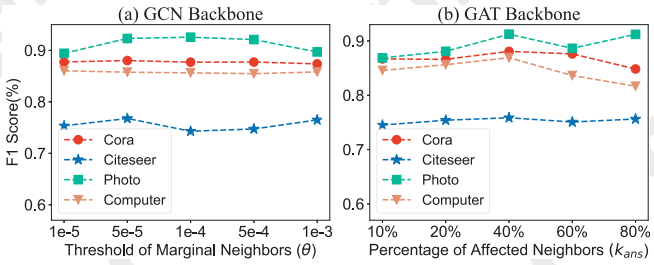

Figure 4: Node unlearning performance with varying parameters.

5 Conclusion

In this paper, we identify two significant limitations in existing GU studies: ineffective forgetting of unlearning elements and inaccurate identification of affected neighbors. These issues arise from the neglect of differences across various GU tasks and backbone GNNs. To address these issues, we propose AGU, an adaptive graph unlearning framework that unifies different GU tasks and develops general neighbor selection strategies adaptable to diverse backbone GNNs. Extensive experiments demonstrate the superior performance of AGU in terms of efficiency, effectiveness, and unlearning capability. In future work, we plan to extend our AGU framework to support various types of graph data, including heterogeneous graphs, knowledge graphs, and temporal graphs.

Acknowledgments

This work is supported by the Australian Research Council Discovery Project DP230100676.

References

[Chen *et al.*, 2022] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph unlearning. In *ACM CCS*, pages 499–513, 2022.

[Chen *et al.*, 2023] Zizhang Chen, Peizhao Li, Hongfu Liu, and Pengyu Hong. Characterizing the influence of graph elements. In *ICLR*, 2023.

[Cheng *et al.*, 2023] Jiali Cheng, George Dasoulas, Huan He, Chirag Agarwal, and Marinka Zitnik. GNNDelete: A general strategy for unlearning in graph neural networks. *ICLR*, 2023.

[Chien *et al.*, 2023] Eli Chien, Chao Pan, and Olgica Milenkovic. Efficient model updates for approximate unlearning of graph-structured data. In *ICLR*, 2023.

[Cong and Mahdavi, 2022] Weilin Cong and Mehrdad Mahdavi. Grapheditor: An efficient graph representation learning and unlearning approach. 2022.

[Ding *et al.*, 2025] Pengfei Ding, Yan Wang, Guanfeng Liu, and Jiajie Zhu. AGU Appendix. https://github.com/Aliezzz/AGU, 2025.

[Dong *et al.*, 2024] Yushun Dong, Binchi Zhang, Zhenyu Lei, Na Zou, and Jundong Li. Idea: A flexible framework of certified unlearning for graph neural networks. In *KDD*, pages 621–630, 2024.

[Fan *et al.*, 2025] Bowen Fan, Yuming Ai, Xunkai Li, Zhilin Guo, Rong-Hua Li, and Guoren Wang. OpenGU: A comprehensive benchmark for graph unlearning. *arXiv preprint arXiv:2501.02728*, 2025.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *NeurIPS*, 30, 2017.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Kolipaka *et al.*, 2024] Varshita Kolipaka, Akshit Sinha, Debangan Mishra, Sumit Kumar, Arvindh Arun, Shashwat Goel, and Ponnurangam Kumaraguru. A cognac shot to forget bad memories: Corrective unlearning in gnns. *arXiv preprint arXiv:2412.00789*, 2024.

[Li *et al.*, 2024a] Fan Li, Xiaoyang Wang, Dawei Cheng, Wenjie Zhang, Ying Zhang, and Xuemin Lin. TCGU: Data-centric graph unlearning based on transferable condensation. *arXiv preprint arXiv:2410.06480*, 2024.

[Li *et al.*, 2024b] Xunkai Li, Yulin Zhao, Zhengyu Wu, Wentao Zhang, Rong-Hua Li, and Guoren Wang. Towards effective and general graph unlearning via mutual evolution. In *AAAI*, pages 13682–13690, 2024.

[Sekhari *et al.*, 2021] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember what you want to forget: Algorithms for machine unlearning. *NeurIPS*, 34:18075–18086, 2021.

[Shchur *et al.*, 2018] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *NeurIPS Workshop*, 2018.

[Tan *et al.*, 2024] Jiajun Tan, Fei Sun, Ruichen Qiu, Du Su, and Huawei Shen. Unlink to unlearn: Simplifying edge unlearning in gnns. In *TheWebConf*, pages 489–492, 2024.

[Velickovic *et al.*, 2017] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[Wang *et al.*, 2023] Cheng-Long Wang, Mengdi Huai, and Di Wang. Inductive graph unlearning. In *USENIX Security*, pages 3205–3222, 2023.

[Wu *et al.*, 2019] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019.

[Wu *et al.*, 2023a] Jiancan Wu, Yi Yang, Yuchun Qian, Yongduo Sui, Xiang Wang, and Xiangnan He. Gif: A general graph unlearning strategy via influence function. In *TheWebConf*, pages 651–661, 2023.

[Wu *et al.*, 2023b] Kun Wu, Jie Shen, Yue Ning, Ting Wang, and Wendy Hui Wang. Certified edge unlearning for graph neural networks. In *KDD*, pages 2606–2617, 2023.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[Yang *et al.*, 2016] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICLR*, pages 40–48, 2016.

[Yang *et al.*, 2025] Zhe-Rui Yang, Jindong Han, Chang-Dong Wang, and Hao Liu. Erase then rectify: A training-free parameter editing approach for cost-effective graph unlearning. In *AAAI*, pages 13044–13051, 2025.

[Yi and Wei, 2025] Lu Yi and Zhewei Wei. Scalable and certifiable graph unlearning: Overcoming the approximation error barrier. In *ICLR*, 2025.

[Zeng *et al.*, 2019] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2019.

[Zhu *et al.*, 2020] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *NeurIPS*, 33:7793–7804, 2020.