OpenGU: A Comprehensive Benchmark for Graph Unlearning

Bowen Fan
Beijing Institute of Technology
fan1085165825@gmail.com

Yuming Ai
Beijing Institute of Technology
3120251027@bit.edu.cn

Xunkai Li
Beijing Institute of Technology
cs.xunkai.li@gmail.com

Zhilin Guo
Shandong University
frank04180@outlook.com

Rong-Hua Li
Beijing Institute of Technology
lironghuabit@126.com

Guoren Wang
Beijing Institute of Technology
wanggrbit@gmail.com

Abstract

Graph Machine Learning is essential for understanding and analyzing relational data. However, privacy-sensitive applications demand the ability to efficiently remove sensitive information from trained graph neural networks (GNNs), avoiding the unnecessary time and space overhead caused by retraining models from scratch. To address this issue, Graph Unlearning (GU) has emerged as a critical solution to support dynamic graph updates while ensuring privacy compliance. Unlike machine unlearning in computer vision or other fields, GU faces unique difficulties due to the non-Euclidean nature of graph data and the recursive message-passing mechanism of GNNs. Additionally, the diversity of downstream tasks and the complexity of unlearning requests further amplify these challenges. Despite the proliferation of diverse GU strategies, the absence of a benchmark providing fair comparisons for GU, and the limited flexibility in combining downstream tasks and unlearning requests, have yielded inconsistencies in evaluations, hindering the development of this domain. To fill this gap, we present OpenGU, the first GU benchmark, where 16 SOTA GU algorithms and 37 multi-domain datasets are integrated, enabling various downstream tasks with 13 GNN backbones when responding to flexible unlearning requests. Through extensive experimentation, we have drawn 10 crucial conclusions about existing GU methods, while also gaining valuable insights into their limitations, shedding light on potential avenues for future research. Our code is available at https://github.com/bwfan-bit/OpenGU.

1 Introduction

Graphs are versatile mathematical structures that represent complex interactions and relationships between entities, offering an abstract yet intuitive framework for modeling real-world systems. To effectively capture the rich and interconnected information inherent in graph data, GNNs [21, 46, 60] have emerged as transformative tools, achieving remarkable success in data management [1, 2], social networks [39, 69], recommendation systems [6, 29, 59], and biological networks [24, 93, 55].

However, the majority of machine learning paradigms including GNNs are primarily driven by data for the acquisition of knowledge, fundamentally transforming decision-making processes across various fields [3, 76, 91]. Therefore, the indiscriminate use of data has intensified the conflict between data rights and user privacy [44, 63, 71]. In response to these pressing issues, a range of pioneering regulatory frameworks has been proposed, including the European Union's General Data Protection Regulation (GDPR) [53], and the California Consumer Privacy Act (CCPA) [50] in California. Among these regulations, one of the most critical and contentious provisions is *the right*

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Track on Datasets and Benchmarks.

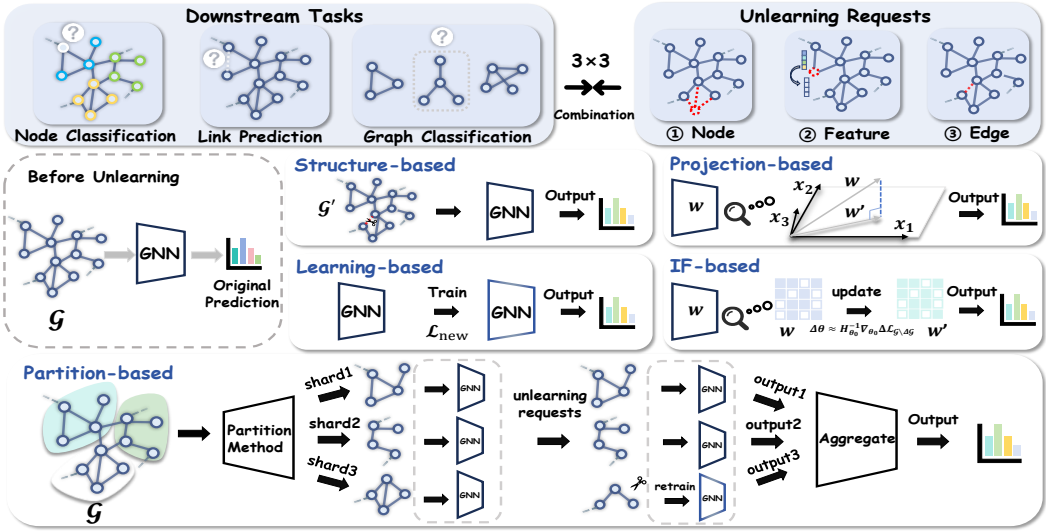

Figure 1: An overview of OpenGU framework, illustrating the key components and methodologies.

to be forgotten [35]. To technically align the effectiveness of machine learning with data rights and privacy regulations, a revolutionary frontier—machine unlearning [7, 77] has emerged.

Despite advances in traditional machine unlearning for independently distributed data, the growing reliance on graph-structured data in various fields highlights the significance of GU. As graphs underpin relational schemas and mined patterns, GU enables selective removal of sensitive information while preserving the integrity of analytical results. Unlike other domains, removing specific information from a graph requires not only modifying individual nodes or edges but also considering how these changes ripple through the entire graph. In addition, GU is jointly determined by downstream tasks (See Appendix A.2) and unlearning requests (See Appendix A.3). The interplay of these two aspects makes GU even more complex. These challenges position GU as a particularly demanding area of research, requiring tailored solutions for its structural complexities and task-specific needs.

Recently, diverse GU methods have surfaced, paving the way for more responsible and privacy-compliant graph learning via data-level techniques, model modifications, or parameter updates. Though these methods have made significant strides, there are still deficiencies in achieving unity:
(1) **Datasets**: The utilization of different datasets varies significantly in terms of domain characteristics and scale. This diversity, combined with the rigidity of processing data regarding splitting and inference settings, fundamentally makes it difficult to analyze the results from a unified perspective.
(2) **Backbones**: The lack of harmonization among GNN backbones between different methods creates barriers for fair comparisons, as experiments for many approaches are confined to specific GNNs, limiting their applicability to broader architectural variations and hindering cross-method evaluations.
(3) **Experiments**: Typically, experiments are conducted on a single downstream task under one type of unlearning request, neglecting the exploration of diverse task-request combinations. Besides, varying configurations and differing evaluation metrics lead to inconsistent interpretations of results.

To address the aforementioned challenges, we propose OpenGU, to our knowledge, the first comprehensive benchmark specifically designed for graph unlearning. OpenGU integrates 16 SOTA algorithms and 37 multi-domain datasets, enabling the flexible 3×3 combinations of unlearning requests and downstream tasks. OpenGU implements these components with unified APIs to standardize method invocation. Based on this design, we conduct an in-depth investigation of GU algorithms, providing valuable insights across three dimensions: effectiveness, efficiency, and robustness.
For **effectiveness**, OpenGU offers a comprehensive evaluation in two aspects: model updates and inference safety. We assess the forgetting ability for unlearning entities and the reasoning capability for retained data to determine if methods achieve effective trade-off between forgetting and reasoning.
For **efficiency**, OpenGU provides insights into scalability by evaluating how GU methods handle increasing amounts of data and more complex graph structures, showing their practical applicability. Furthermore, we analyze the computational resources required by various graph unlearning algorithms, focusing on time and space complexity from both theoretical and empirical perspectives.
For **robustness**, we focus on noise and sparsity scenarios in real-world applications, and delve into

the algorithms' capacity to maintain performance stability and integrity amid different unlearning intensities, particularly under varying levels of data perturbation.

Our contributions. We propose OpenGU in this paper, the inaugural benchmark specifically tailored for GU domain. Our contributions are outlined as follows:
(1) *Comprehensive Benchmark.* OpenGU integrates 16 SOTA algorithms and 37 popular multi-domain datasets, establishing a thorough evaluation framework for a fair comparison. Moreover, OpenGU expands and standardizes experimental task requirements, encompassing unlearning requests and downstream tasks, thereby facilitating a code-level implementation of 3×3 cross-experiments.
(2) *Analytical Insights.* Through extensive empirical experiments centered around algorithms' forgetting capability and reasoning capability, we conduct a thorough analysis from three perspectives and summarize 10 key conclusions, envisioning our conclusions as the catalyst for graph unlearning field.
(3) *Open-sourced Benchmark Library.* OpenGU is designed as an open-source benchmark library, providing researchers and practitioners with accessible tools and resources for expanding the exploration of graph unlearning. Additionally, comprehensive documentation and user-friendly interfaces foster collaboration and innovation within the GU community.

2 Definitions and Background

In this section, we will briefly review several key concepts and definitions to better explain the fundamentals of GU. Further details on GNNs and GU mechanisms can be found in Appendix A.1.

Graph Notations. Generally, we define a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{V}$ represents a set of nodes with $|\mathcal{V}| = n$, and $\mathcal{E}$ denotes the edge set containing $|\mathcal{E}| = m$ edges. The feature matrix, $\mathcal{X} \in \mathbb{R}^{n \times d}$, represents the feature vectors of all nodes, and d represents the dimension of node features. We also define $\mathcal{Y} = \{y_1, y_2, ..., y_n\}$ as the label set, where each y_i corresponds to the node $v_i \in \mathcal{V}$ in a one-to-one manner. Apart from the aforementioned attributes, the graph is also characterized by its adjacency matrix $A \in \mathbb{R}^{n \times n}$, where each element A_{ij} indicates the edge between nodes i and j. Formally, we define $\mathcal{M}$ as the original model trained on a complete graph $\mathcal{G}$ or a graph dataset G.

Unlearning Requests. When receiving unlearning request $\Delta\mathcal{G} = \{\Delta\mathcal{V}, \Delta\mathcal{E}, \Delta\mathcal{X}\}$, the retrained model $\hat{\mathcal{M}}$ is trained from scratch on the pruned graph after the deletion and the unlearning model $\mathcal{M}'$ updates its parameters from original W to W' based on the unlearning algorithm. The goal of GU is to minimize the discrepancy between $\mathcal{M}'$ and $\hat{\mathcal{M}}$. Common unlearning requests in graph unlearning typically fall into three categories: node-level $\Delta\mathcal{G} = \{\Delta\mathcal{V}, \varnothing, \varnothing\}$, edge-level $\Delta\mathcal{G} = \{\varnothing, \Delta\mathcal{E}, \varnothing\}$, and feature-level $\Delta\mathcal{G} = \{\varnothing, \varnothing, \Delta\mathcal{X}\}$. Noting that $\Delta\mathcal{V}$ represents only the unlearning request received by the model, the edges connected to the nodes must also be considered during graph modification.

GU Taxonomy. To provide a comprehensive understanding, we categorize existing methodologies into five types based on their operational frameworks: (1) Partition-based algorithms, (2) Influence Function-based (IF-based) unlearning algorithms, (3) Learning-based algorithms, (4) Projection-based algorithms, and (5) Structure-based algorithms. This categorization provides a structured landscape of GU, as illustrated in Figure 1, with classification details provided in Appendix A.4.

3 Benchmark Design

In this section, we present a comprehensive overview of OpenGU, emphasizing the key aspects of the benchmark design, as outlined in Table 1. First, we detail the datasets and the associated preprocessing techniques (Sec. 3.1), followed by an examination of the various GU methods (Sec. 3.2). Lastly, we outline the evaluation metrics and experimental configurations (Sec. 3.3) that structure the benchmarking process, offering the core principles and the holistic view of OpenGU [1].

3.1 Dataset Overview for OpenGU

OpenGU rigorously selects domain-diverse datasets to evaluate graph understanding tasks: 19 datasets for node or edge-related work [33] (e.g., citation networks: Cora, PubMed [82]; co-author networks: CS and Physics [57]; e-commerce: ogbn-products [32]; webpage networks: Squirrel [51]) and 17 datasets for graph classification (e.g., compound networks: MUTAG [20], ogbg-molhiv [32]; protein networks: ENZYMES [8]; social networks: IMDB-BINARY [80]; 3D networks: ShapeNet [83]).

[1] Due to their specialized nature, the Projector and UtU methods are not categorized within mainstream GU algorithms in the table.

Table 1: An overview of OpenGU.

Datasets	
Type	Homophily, Heterophily, Node, Edge, Graph
Domain	Citation, Co-author, Social, Wiki, Image, Protein, Movie, ...
Preprocess	Transductive/Inductive, Label-Balanced/Label-Random, Noise, Sparsity
GNN Algorithms	
Decoupled	SGC, SSGC, SIGN, APPNP
Sampling	GraphSAGE, GraphSAINT, Cluster-gcn
Traditional	GCN, GCNII, LightGCN, GAT, GATv2, GIN
Mainstream GU Algorithms	
Partition-based	GraphEraser, GUIDE, GraphRevoker
IF-based	GIF, CGU, CEU, GST, IDEA, ScaleGUN
Learning-based	GNNDelete, MEGU, SGU, D2DGN, GUKD
Evaluations	
Attack	Membership Inference Attack, Poisoning Attack
Effectiveness	Accuracy, Precision, F1-score, AUC-ROC
Efficiency	Time, Memory, Theoretical Algorithm Complexity
Robustness	Deletion Intensity, GU Scenario Noise and Sparsity
Unlearning Request	Node-level Request, Feature-level Request, Edge-level Request
Downstream Task	Node Classification, Link Prediction, Graph Classification

To provide a clearer understanding of the datasets, a detailed overview is shown in Tables 4 and 5. Further details and preprocessing about datasets can be found in Appendix B.2 and B.3.

3.2 Algorithm Framework for OpenGU

GNN Backbones. To evaluate the generalizability of GU algorithms, we incorporate three predominant paradigms of GNNs within OpenGU: traditional GNNs [33, 65, 78], sampling GNNs [14, 27, 86], and decoupled GNNs [25, 72, 92]. This multi-faceted coverage enables rigorous benchmarking of GU algorithms across varying computational constraints, scalability demands, and representational capacities. More detailed descriptions of these GNN backbones are provided in Appendix C.

GU Algorithms. For GU algorithms, our framework encompasses 16 methods, each meticulously reproduced based on source code or detailed descriptions in the relevant publications. The detailed descriptions of GU can be found in Appendix D. Moreover, we deliver a unified interface for GU methods, merging them under a cohesive API to facilitate easier access, experimentation, and future expansion. By standardizing these methods within OpenGU, we provide a streamlined and highly efficient platform for researchers and practitioners to conduct robust, reliable, and reproducible benchmarking studies. More detailed information about OpenGU API can be found in Appendix E.1.

3.3 Evaluation Strategy for OpenGU

To assess GU algorithms in diverse real-world scenarios, OpenGU evaluation spans three dimensions tailored to GU contexts: effectiveness, efficiency, and robustness. Each dimension includes tailored evaluation methods reflecting OpenGU's mission to serve as a flexible, high-standard benchmark.

Cross-over Design. In previous GU studies, node and feature unlearning typically align with node classification tasks, while edge unlearning is often evaluated in the context of link prediction. However, real-world applications frequently demand the removal of data in scenarios where unlearning requests and downstream tasks intersect. To address this gap, we systematically modified and extended existing GU methods by decoupling their task-specific components and enabling cross-task compatibility through unified code interfaces. This approach provides a comprehensive and practical evaluation framework to assess the flexibility of GU algorithms in real-world applications. Further detailed information regarding the codebase implementation are provided in Appendix E.2.

Effectiveness. For the effectiveness of algorithms within OpenGU, we evaluate model performance on key downstream tasks while explicitly measuring unlearning impacts. We leverage F1-score, AUC-ROC, and Accuracy to evaluate the model's predictive performance at the node, edge, and graph levels, respectively. To rigorously validate unlearning effects, we employ membership inference attacks (MIA) [19, 48, 89] and poisoning attacks (PA) [94, 95], which systematically verify the erasure of target data. This multi-faceted approach provides a thorough assessment of GU effectiveness,

ensuring comprehensive evaluation of both retained and unlearned information. Rationales for metric selection and detailed methodological descriptions are provided in Appendix F.

Efficiency. In evaluating GU efficiency, we assess scalability, time complexity, and space complexity. Scalability examines adaptability to varying dataset sizes, reflecting performance stability across graph scales. Time complexity combines theoretical and empirical analysis to gauge computational demands. Space complexity focuses on memory efficiency, measuring peak usage and storage needs during unlearning to identify viability in resource-constrained settings. Together, these metrics provide a comprehensive view of each method's suitability for real-time and scalable deployment.

Robustness. To evaluate the robustness of algorithms in OpenGU, we systematically examine model performance under varying levels of deletion intensity, noise and sparsity. This involves assessing how different proportions of data perturbation affect the model's predictive and unlearning capabilities. Robust GU algorithms should ideally demonstrate minimal performance degradation as deletion intensity increases, reflecting strong resilience in maintaining optimum performance.

4 Experiments and Analyses

In this section, we delve into a series of experiments designed to rigorously evaluate the effectiveness, efficiency, and robustness of algorithms within OpenGU. By posing key questions, we aim to uncover insights into how these algorithms respond to diverse unlearning scenarios, data complexities, and practical deployment challenges, ultimately providing a comprehensive understanding of their efficacy. More detailed information about the experimental environment is presented in Appendix G.1.

For **effectiveness**, **Q1:** How effective are GU algorithms in predicting retained data under different unlearning requests? **Q2:** Do existing GU strategies achieve forgetting in response to unlearning requests? **Q3:** Do current GU algorithms effectively balance the trade-off between forgetting and reasoning? For **efficiency**,**Q4:** How do GU algorithms perform in terms of space and time complexity theoretically? **Q5:** How do the GU algorithms perform regarding space and time consumption in practical scenarios? For **robustness**, **Q6:** As the intensity of forgetting requests increases, can the GU algorithms still maintain their original performance levels? **Q7:** How do GU algorithms perform under sparse and noisy settings?

4.1 Reasoning Performance Comparison

To address **Q1**, we conducted a comprehensive comparison and analysis of existing methods across three representative combinations of downstream tasks and unlearning requests. For clarity, we denote these combinations as downstream task-unlearning request (e.g. node-node). For dataset configuration, we adopt an 80%/20% training-test split across all tasks. Unlearning requests are systematically defined: (1) node-level: removing 10% of the total nodes from the training set, (2) edge-level: deleting 10% of edges, and (3) graph-feature-level: selecting 50% of training graphs and setting 10% of their node features to zero vectors. More information about standardized experimental protocol and experimental settings can be found in Appendix G.2 and H.

Node-Node Experiment. From Table 2, we can reveal several key observations: (1) The best and second-best results are predominantly achieved by Learning-based methods, with SGU and D2DGN standing out. This indicates that the tailored strategies and loss function designs in Learning-based approaches effectively maintain SOTA performance for predicting retained data. (2) On smaller, commonly used datasets such as Cora, CiteSeer, and PubMed, most methods deliver relatively strong performance. However, on larger datasets like ogbn-arxiv, several methods, including CGU, GNNDelete, GUKD, and Projector, encounter challenges such as out-of-memory or out-of-time.

Edge-Edge Experiment. Based on the results presented in Table 9, several key observations can be drawn: (1) IF-based methods achieve superior performance in edge-edge experiments, retaining high link prediction accuracy after unlearning, with GIF and IDEA standing out. This demonstrates that leveraging influence functions enables strong generalizability. (2) Partition-based methods face challenges in effectively addressing link prediction tasks. This is primarily due to the inherent sparsity of edges in most datasets, where partitioning further reduces the number of meaningful edges. (3) While Learning-based methods excel in node-node scenarios, most of them fall significantly behind SOTA in edge-edge settings. However, GNNDelete consistently maintains high accuracy across all tested datasets. This gap likely arises because Learning-based approaches prioritize node-level optimization, neglecting explicit modeling of structural dependencies critical to edge removal.

Table 2: F1 ± STD comparison(%) for transductive node classification with node unlearning. The highest results are highlighted in bold, while the second-highest results are marked with underline.

Node-Level	Cora	CiteSeer	PubMed	ogbn-arxiv	CS	Flickr	Chameleon	Minesweeper	Tolokers
Retrain	90.41±0.13	77.18±0.24	86.89±0.19	70.39±0.04	93.51±0.06	49.03±0.32	61.75±0.18	81.29±0.05	79.26±0.05
GraphEraser	81.14±1.00	73.57±1.25	84.68±0.54	62.72±0.18	91.24±0.08	46.93±0.14	45.18±1.46	80.45±0.08	78.36±0.30
GUIDE	73.89±2.18	63.50±0.71	84.02±0.15	OOM	86.96±0.15	OOM	43.37±0.04	44.71±0.00	44.11±0.00
GraphRevoker	81.09±1.63	73.45±0.61	84.94±0.14	62.72±0.18	91.26±0.10	46.91±0.14	44.87±1.72	80.44±0.12	78.36±0.30
GIF	81.75±1.09	62.58±0.67	78.60±0.22	65.52±0.17	91.87±0.22	47.56±0.12	55.09±1.23	77.83±0.09	78.44±0.10
CGU	86.37±0.78	75.62±0.41	76.07±0.15	OOT	OOT	OOT	35.83±0.74	80.85±0.00	78.91±1.49
ScaleGUN	80.26±0.00	72.87±0.06	80.55±0.01	58.76 ±0.01	90.13±0.01	43.58±0.01	49.61±0.58	69.80±0.04	72.59±0.04
IDEA	87.71±0.25	63.66±0.49	80.44±0.13	64.22±0.17	89.47±0.22	42.01±0.04	52.89±0.80	77.93±0.04	78.72±0.15
CEU	87.12±0.07	71.56±0.15	**86.91±0.06**	57.80±0.83	92.23±0.07	**49.47±0.19**	**61.89±0.54**	81.05±0.04	78.72±0.03
GNNDelete	74.78±5.49	64.26±3.82	84.82±1.36	OOM	76.26±2.73	42.03±0.00	54.43±1.87	81.04±0.19	79.12±0.16
MEGU	82.68±1.56	63.60±1.11	79.68±0.60	66.15±0.29	91.69±0.08	47.97±0.13	53.82±1.68	77.64±0.02	**79.29±0.10**
SGU	**89.26±0.49**	72.04±1.70	86.61±0.02	**67.20±0.08**	**93.20±0.07**	48.70±0.21	60.18±0.26	81.11±0.02	79.18±0.04
D2DGN	88.41±0.20	73.81±0.28	86.06±0.05	66.08±0.18	92.99±0.03	48.01±0.03	53.25±0.70	**81.19±0.04**	79.18±0.05
GUKD	79.65±1.98	70.63±0.93	83.37±0.60	OOM	75.04±0.82	42.03±0.04	36.62±2.12	80.89±0.04	78.91±0.00
Projector	86.79±2.37	**77.00±0.65**	83.97±0.30	OOT	88.40±0.63	OOT	43.29±1.17	80.85±0.00	78.91±0.00

Graph-Feature Experiment. From the results presented in Table 10, we observe that: (1) IF-based methods remain effective for graph classification task, achieving commendable performance in most cases. Though the Partition-based approach is relatively straightforward, GraphEraser shows competitive performance across these datasets. (2) Although OpenGU provides comprehensive graph-level dataset interfaces, existing algorithms struggle to handle large-scale datasets due to inherent computational bottlenecks particularly in terms of memory scalability.

Through our exploration of **Q1**, we derive three key conclusions. **C1**: *Partition and IF-based methods demonstrate broad applicability to most tasks, while Learning-based methods, Projector, and UtU are more specialized, limiting their scalability and generalization* [17, 88]. **C2**: *Learning-based methods excel in targeted tasks, often achieving SOTA performance, while IF-based methods offer flexibility and maintain competitiveness in more tasks* [43]. **C3**: *The scalability gaps of GU algorithms manifest in two dimensions: memory bottlenecks during graph task, and the inherent incompatibility of current unlearning paradigms with feature-level unlearning requirements* [28, 70].

4.2 Forgetting Performance Comparison

To address **Q2**, we evaluate whether GU algorithms effectively forget the unlearning entity by employing commonly used attack strategies in the GU domain, including Membership Inference Attack and Poisoning Attack. These assessments are conducted from both node and edge perspectives to determine whether existing GU methods can genuinely prevent information leakage and protect privacy. The experimental setup in this section is identical to Section 4.1. More details about experimental setup and results can be found in Appendix I.

In node-node experiments, MIA assesses GU algorithms' privacy protection, with an AUC near 0.5 indicating predictions akin to random guesses, suggesting minimal information leakage and effective sensitive data removal. We assess various GU methods across multiple datasets, selecting CiteSeer as a representative case (Figure 2). The results reveal that: For Partition-based methods, GraphEraser underperforms with the BEKM partitioning strategy, while GUIDE with SR and Fast strategies excels in privacy protection. Learning-based methods like SGU, D2DGN, and GNNDelete achieve AUC values near 0.5, demonstrating robust unlearning capabilities. Although IF-based methods theoretically limit differences between original and unlearned models, CGU fails to meet the complete unlearning benchmark, as confirmed by MIA.

In edge-edge experiments, PA assesses GU methods' performance by introducing 10% heterophilic edges as poisoned data. The effectiveness of unlearning is measured by the improvement in link prediction AUC, with a larger increase indicating successful removal of poisoned edge information [2]. We evaluated various GU methods across datasets, selecting Cora as a representative case (Figure 3). The findings show that all Partition-based and IF-based methods achieve an increase in AUC

[2]The histogram column on the left represents the result before unlearning, and the corresponding one on the right represents the result after unlearning. And for convenience, UtU is presented in IF-based.

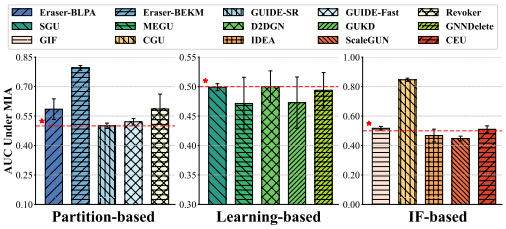

Figure 2: AUC-ROC ± STD comparison under MIA for node-node task with SGC backbone.

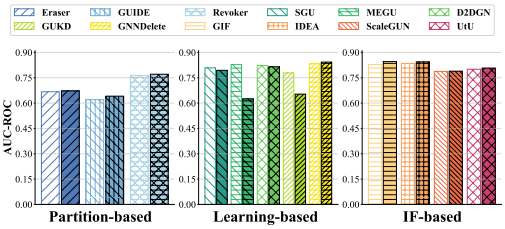

Figure 3: AUC-ROC comparison under PA for edge-edge task with GraphSAGE backbone.

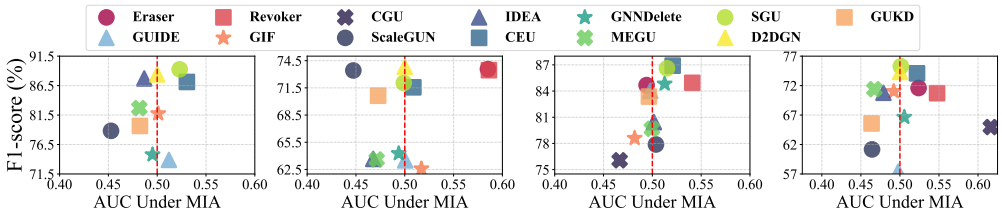

Figure 4: Trade-off between forgetting and reasoning on Cora, CiteSeer, PubMed and in average.

after unlearning, demonstrating their capability to address edge-level unlearning requests in link prediction tasks. Among these, GIF and IDEA stand out by not only effectively removing the harmful edges but also achieving high AUC scores. On the other hand, among the Learning-based methods, only GNNDelete successfully handles the removal of poisoned edges during unlearning, achieving a notably high AUC. Other Learning-based approaches exhibit varying levels of performance decline, suggesting that the unlearning process might inadvertently affect critical information.

Based on our analysis of **Q2**, we draw the conclusion **C4**: *For learning-based methods, only GNNDelete shows superiority in edge-level tasks, highlighting the need for further development in edge-level learning-based approaches* [38]. **C5**: *IF-based methods provide theoretical unlearning guarantees, but MIA reveals that some fail to fully eliminate privacy leaks, necessitating the integration of theoretical analysis and empirical evaluation* [10].

4.3 Trade-off between Forgetting and Reasoning

To address **Q3**, we synthesize insights from the preceding questions and adopt a unified perspective to analyze their interplay, providing a clear overall performance of various methods in balancing forgetting and reasoning (Figure 4). In this part, we focus on node classification on Cora, CiteSeer and PubMed, using SGC as the backbone for consistent comparisons, with the experimental setup adhering to configurations in Appendix H.1. For additional experimental details, refer to Appendix J.

Given that an AUC of 0.5 under the MIA serves as the theoretical baseline for indistinguishable inference, GU methods positioned closer to the red centerline and higher on the graph exhibit superior overall performance in balancing unlearning and retention. From a vertical perspective, which reflects reasoning capabilities, SGU, D2DGN, and CEU emerge as top performers, demonstrating exceptional strength in maintaining predictive accuracy on retained data. In contrast, GUIDE displays comparatively weaker reasoning performance, struggling to uphold effectiveness in this aspect. Horizontally, the distribution of AUC scores under MIA reveals distinct trends: apart from CGU, which attains an AUC exceeding 0.6, markedly deviating from the theoretical baseline, most methods cluster within a narrow range of 0.45 to 0.55. This concentration underscores a complex trade-off between unlearning efficacy and retention fidelity, evidencing phenomena such as over-forgetting, where excessive unlearning impairs utility, and under-forgetting, where incomplete unlearning falls short of privacy objectives. Among the evaluated methods, SGU, D2DGN, GNNDelete, and GUIDE distinguish themselves with robust unlearning capabilities, effectively reducing the influence of unlearned data, though their success in navigating these competing demands varies.

Upon exploring **Q3**, we conclude that **C6**: *Future algorithms should prioritize improving the balance between forgetting and reasoning trade-offs. The issues of under-forgetting and over-forgetting constitute critical challenges, requiring further research to enhance effectiveness* [16, 45].

Table 3: Algorithm complexity analysis for existing prevalent GU studies.

Method	Preprocessing	Training	Unlearning	Inference	Memory
GUIDE	$O(ktn^2+kctn)$	$O(Lfn/k+Lf^2n)$	$O(Lk'fn^2/k^2+Lkf^2n/k)$	$O(Lfm+Lf^2n+L_kfn)$	$O(n^2/k^2+Lfn+kn)$
GraphEraser	$O(kdtn+ktn\log(kn))$	$O(Lfn/k+Lf^2n)$	$O(Lk'fn^2/k^2+Lkf^2n/k)$	$O(Lfm+Lf^2n+Ln_sf/k+Ln_sf^2)$	$O(n^2/k^2+Lfn+kn)$
GraphRevoker	$O(k(d+c)n)$	$O(Lfn/k+Lf^2n)$	$O(Lk'fn^2/k^2+Lkf^2n/k)$	$O(Lfm+Lf^2n+kf^2n_s)$	$O(n^2/k^2+Lfn+kn)$
GIF	-	$O(Lfm+Lf^2n)$	$O(n\|\theta\|)$	$O(Lfm+Lf^2n)$	$O(\|\theta\|)$
CGU	-	$O(Lfm+f^2n)$	$O((Lmf+f^2n)u)$	$O(Lfm+f^2n)$	$O(m+f^2+fn)$
CEU	-	$O(Lfm+Lf^2n)$	$O(t\|\theta\|+u\|\theta\|)$	$O(Lfm+Lf^2n)$	$O(\|\theta\|)$
GST	-	$O(\sum_{i=0}^{N} pg_i^2)$	$O(((p+\|\theta\|)\sum_{i=0}^{N} g_i^2+\|\theta\|^3)u)$	$O(\sum_{i=0}^{N} pg_i^2)$	$O(pfn)$
IDEA	-	$O(Lfm+Lf^2n)$	$O(n\|\theta\|)$	$O(Lfm+Lf^2n)$	$O(\|\theta\|)$
ScaleGUN	-	$O(Lfm+f^2n)$	$O(L^2d^2u)$	$O(Lfm+f^2n)$	$O(m+f^2+fn)$
SGU	$O(Bf^2+Bn_s+f^2u)$	$O(Lfm+f^2n)$	$O(Lf^2+Bn_sf+(c+f)u)$	$O(Lfm+f^2n)$	$O(Bfn_s+f^2)$
MEGU	$O(Lfm+Lf^2n+d^2u)$	$O(Lfm+Lf^2n+d^2u)$	$O(d^2cu)$	$O(Lfm+Lf^2n)$	$O(Lfm+f^2n)$
GUKD	$O(Lfm+Lf^2n)$	$O(L_tfm+Lf^2n)$	$O(c(n-u))$	$O(Lfm+Lf^2n)$	$O(f^2+fn)$
D2DGN	$O(Lfm+Lf^2n)$	$O(Lfm+Lf^2n)$	$O(c(n-u))$	$O(Lfm+Lf^2n)$	$O(f^2+fn)$
GNNDelete	$O(Lfm+Lf^2n+d^2u)$	$O(Lfm+Lf^2n)$	$O(d^2fu)$	$O(Lfm+Lf^2n)$	$O(fu+d^2fu)$
Projector	-	$O(Lfm+Lf^2n)$	$O(f^2n+\max\{u^3,f^2u\})$	$O(Lfm+Lf^2n)$	$O(f^2+fn)$
UtU	-	$O(Lfm+Lf^2n)$	$O(u)$	$O(Lfm+Lf^2n)$	$O(m+Lfn)$

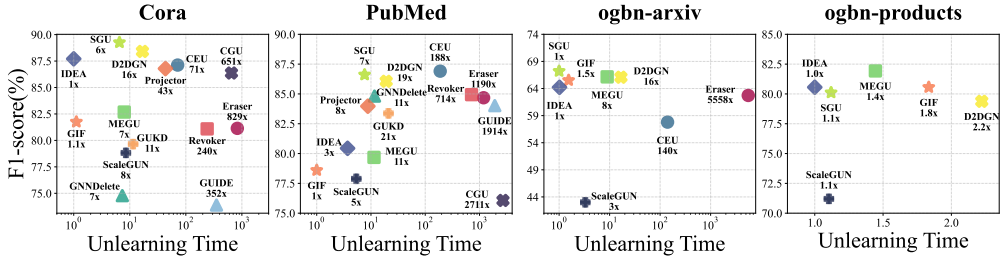

Figure 5: Unlearning Time Performance on Cora, PubMed, ogbn-arxiv and ogbn-products.

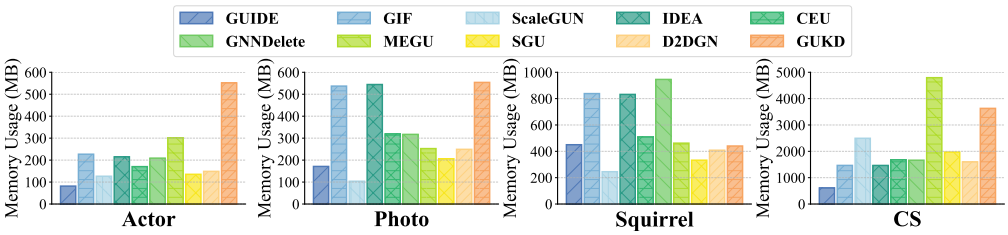

Figure 6: Memory Usage Performance on Various Datasets.

4.4 Algorithm Complexity and Practical Efficiency Analyses

Since **Q4** and **Q5** are closely interconnected, we integrate their analyses in this section. More details regarding notation definitions and experimental specifics are provided in Appendix K.

To address **Q4**, we analyze the time complexity from the perspectives of preprocessing, training, unlearning, and inference. As detailed in Table 3, the training and inference phases reveal that GST deviates from the complexity of standard GNN training, while other methods align with conventional expectations. In the unlearning phase, both time and space complexity vary based on method-specific characteristics: Partition-based methods exhibit complexities comparable to those of training, IF-based methods are predominantly governed by the parameter size $|\theta|$ and constrained by the extent of optimization applied to Hessian matrix computations, and Learning-based methods scale with data attributes, such as the feature dimension f. For **Q5**, Figure 5 illustrates that Partition-based methods incur substantial time overheads due to the combined demands of partitioning, aggregation, and shard training. In contrast, most IF-based and Learning-based methods demonstrate greater temporal efficiency. When evaluated on large-scale datasets like ogbn-products, only six GU methods completed execution successfully with competitive performance, as others encountered timeouts or memory overflows. As depicted in Figure 6, GUIDE consistently exhibits low memory overhead across various datasets, while GraphEraser (without visualization) incurs higher costs, exceeding 4000MB even on smallest dataset Actor. Notably, scalable methods such as ScaleGUN and SGU exhibit consistent memory usage across all datasets, underscoring their robustness and scalability.

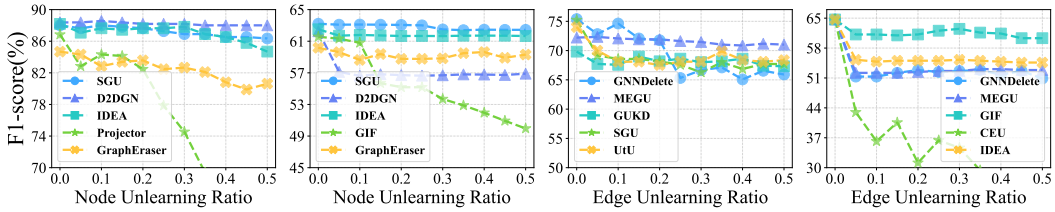

Figure 7: Performance under Different Unlearning Intensities.

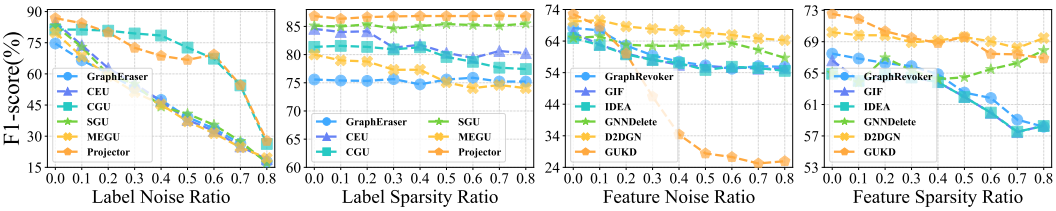

Figure 8: Performance under Different Noise and Sparsity Ratios at Label and Feature Levels.

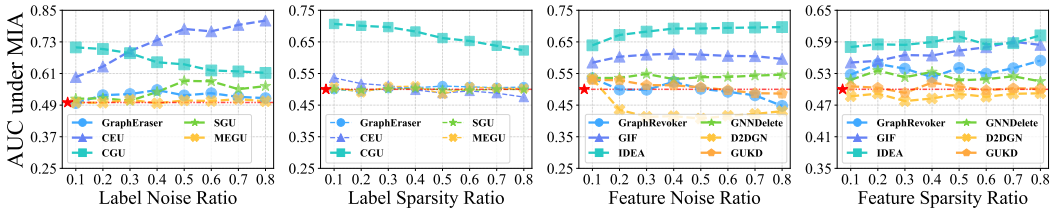

Figure 9: MIA under Different Noise and Sparsity Ratios at Label and Feature Levels.

Based on the analysis, we conclude **C7**: *To reduce time costs, partition-based methods require optimization in partitioning, IF-based methods need to minimize the computational overhead of Hessian matrix calculations in GU scenarios, and Learning-based approaches demand enhanced efficiency in preprocessing* [9, 79]. **C8**: *Existing GU methods require further optimization to effectively reduce time and space overhead, emphasizing the need for more efficient implementations* [26, 64].

4.5 Robustness Analyses

To investigate the robustness of GU algorithms, we designed comprehensive experiments targeting unlearning intensity and noise perturbations. These experiments systematically assess the algorithms' performance in challenging scenarios, with further details provided in Appendix L.

To address **Q6**, we perform two types of unlearning experiments. The first investigates the influence of varying unlearning ratios in a single batch, while the second explores the behavior of models under incremental unlearning requests. For the first experiment, we conducted node unlearning experiments on Cora and ogbn-arxiv and edge unlearning experiments on CiteSeer and Chameleon, employing various backbones for the node classification task. The unlearning ratio was incrementally increased from 0 to 0.5, and the results are presented in Figure 7. The illustration reveals a consistent downward trend in performance for all GU methods as the unlearning ratio increases, indicating that higher deletion intensities negatively impact prediction capabilities. Notably, methods such as Projector, GIF, and CEU exhibit greater sensitivity to changes in certain datasets, while Learning-based methods demonstrate a more gradual decline, highlighting their robustness under higher unlearning intensities. However, we also observe that even with minimal deletion ratios, many methods experience significant performance degradation during unlearning, particularly on the Chameleon dataset. This suggests that current GU algorithms need to take the deletion ratio into account to better reduce the gap in predictive performance between the unlearned and original model. To examine the second scenario, we perform incremental unlearning experiments, where 5% of the target nodes are sequentially removed in multiple rounds. As shown in Figure 10, all methods experience a sharp initial drop followed by a gradual decline, reflecting partial adaptation to continuous unlearning and distinct robustness differences among algorithms. In **Q7**, we simulate more realistic noise and sparsity scenarios by introducing perturbations at both the label and feature levels to comprehensively evaluate

the robustness of existing GU methods. For label noise, a certain proportion of training samples are randomly assigned incorrect labels, while for feature noise, Gaussian noise is injected based on the dimensionality of node features. Sparsity is introduced by varying the proportion of training nodes and simulating partial feature absence. Given the large number of GU methods, we select representative approaches for analysis based on their categories, using the same settings as the node-node experiments in Q1 and adopting SSGC as the backbone. Experiments on the Cora and CiteSeer datasets show that simulated noise and sparsity cause a performance drop across all methods, with label noise exhibiting a notably stronger effect (as illustrated in Figure 8). For label noise, CGU and Projector degrade slowly then sharply, unlike the steady and steep drop in other methods. Label sparsity has a milder effect, with some methods retaining performance. Feature noise markedly impairs GUKD, revealing its vulnerability, while feature sparsity reduces performance in GIF, IDEA, and GraphRevoker, yet GNNDelete improves unexpectedly. As indicated by Figure 9, representative methods are comparatively less affected by Label Sparsity Ratio. In contrast, most GU methods fail to exhibit a distinct or unified trend when faced with other forms of perturbations.

Based on analyses, we derive conclusion **C9**: *While most GU algorithms perform reasonably well across varying unlearning intensities, there remains a critical need to enhance their robustness across diverse datasets* [75]. **C10**: *Current GU methods exhibit insufficient robustness to noise and sparsity, particularly in the presence of label noise, posing a challenge to enhance the overall robustness* [54].

5 Conclusion and Future Directions

In this paper, we first review advancements in GU, detailing its applications and classifying algorithms by technical traits. Then we introduce OpenGU, the first unified and comprehensive benchmark for GU, which integrates 16 GU algorithms and datasets across multiple domains, supporting flexible combinations for downstream tasks and unlearning requests. Through OpenGU's standardized evaluation, we extensively assess GU methods' effectiveness, efficiency, and robustness, summarizing 10 insightful conclusions. To inspire further research, we outline the major challenges currently faced by GU and propose promising directions for future exploration.

Designing Generalized GU Frameworks for Diverse Tasks (*C1*, *C2* and *C3*). While some existing methods demonstrate strong predictive performance for specific tasks, their underlying principles often make it difficult to adapt to varying downstream tasks and unlearning requests. Furthermore, the current design of GU algorithms remains relatively simplistic, whereas real-world applications often require handling mixed unlearning requests or diverse graph types, such as dynamic graphs, spatialtemporal graphs, or knowledge graphs. Achieving consistently superior predictive capabilities across such complex and varied scenarios remains a considerable challenge.

Unified Metrics for Evaluating Forgetting (*C4*, *C5* and *C6*). The current methods for assessing the forgetting capability remain insufficient, as they are tightly coupled with specific unlearning requests and downstream tasks, making them less effective in handling diverse combinations of scenarios. Future research should also move beyond the current paradigm of independently assessing these two aspects (forgetting and reasoning), striving instead for a unified metric that evaluates models from an integrated perspective, ensuring a comprehensive understanding of their capabilities.

Enhancing Algorithm Efficiency (*C7* and *C8*). While theoretical analysis provides valuable insights, the practical performance of current GU methods often falls short, especially when scaling to large datasets with millions of nodes. Current methods commonly encounter OOT or OOM issues. To enable GU's effective deployment in large-scale scenarios, algorithms must be optimized for both efficiency and scalability to avoid these performance bottlenecks.

Addressing Realistic Scenarios (*C9* and *C10*). In practical applications, the presence of noise and incomplete datasets is an unavoidable challenge. However, current GU algorithms lack sufficient exploration and adaptation to such scenarios. Experimental results highlight significant weaknesses when dealing with noise and sparsity, particularly in terms of label and feature robustness. Future research should aim to broaden the scope of investigation, extending robustness analysis to encompass a wider range of real-world challenges and data imperfections.

6 Acknowledgement

This work was supported by the NSFC Grants U2241211, U24A20255, and 62427808. Rong-Hua Li is the corresponding author of this paper.

References

[1] Charu C Aggarwal and Haixun Wang. Graph data management and mining: A survey of algorithms and applications. *Managing and mining graph data*, pages 13–68, 2010.

[2] Renzo Angles and Claudio Gutierrez. An introduction to graph data management. *Graph Data Management: Fundamental Issues and Recent Developments*, pages 1–32, 2018.

[3] Alaa Bessadok, Mohamed Ali Mahjoub, and Islem Rekik. Graph neural networks in network neuroscience. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5833–5848, 2022.

[4] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159, 2021.

[5] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *International Conference on Learning Representations, ICLR*, 2022.

[6] Xuheng Cai, Chao Huang, Lianghao Xia, and Xubin Ren. Lightgcl: Simple yet effective graph contrastive learning for recommendation. In *International Conference on Learning Representations, ICLR*, 2023.

[7] Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480, 2015.

[8] Chuan Chen, Ziyue Xu, Weibo Hu, Zibin Zheng, and Jie Zhang. Fedgl: Federated graph learning framework with global self-supervision. *Information Sciences*, 657:119976, 2024.

[9] Jiaao Chen and Diyi Yang. Unlearn what you want to forget: Efficient unlearning for llms. *arXiv preprint arXiv:2310.20150*, 2023.

[10] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. When machine unlearning jeopardizes privacy. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 896–911, 2021.

[11] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph unlearning. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2022.

[12] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning, ICML*, 2020.

[13] Jiali Cheng, George Dasoulas, Huan He, Chirag Agarwal, and Marinka Zitnik. Gnndelete: A general strategy for unlearning in graph neural networks. In *International Conference on Learning Representations, ICLR*, 2023.

[14] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*, 2019.

[15] Eli Chien, Chao Pan, and Olgica Milenkovic. Certified graph unlearning. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, 2022.

[16] Dasol Choi and Dongbin Na. Distribution-level feature distancing for machine unlearning: Towards a better trade-off between model utility and forgetting, 2024.

[17] Somnath Basu Roy Chowdhury, Krzysztof Choromanski, Arijit Sehanobish, Avinava Dubey, and Snigdha Chaturvedi. Towards scalable exact machine unlearning using parameter-efficient fine-tuning, 2024.

[18] Weilin Cong and Mehrdad Mahdavi. Efficiently forgetting what you have learned in graph representation learning via projection. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, 2023.

[19] Mauro Conti, Jiaxin Li, Stjepan Picek, and Jing Xu. Label-only membership inference attack against node-level graph neural networks. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security (AISec'22)*, pages 1–12. Association for Computing Machinery, 2022.

[20] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[21] Michal Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems, NeurIPS*, 2016.

[22] Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330 4:771–83, 2003.

[23] Yushun Dong, Binchi Zhang, Zhenyu Lei, Na Zou, and Jundong Li. Idea: A flexible framework of certified unlearning for graph neural networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 621–630. Association for Computing Machinery, 2024.

[24] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[25] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[26] Tao Guo, Song Guo, Jiewei Zhang, Wenchao Xu, and Junxiao Wang. Efficient attribute unlearning: Towards selective removal of input attributes from feature representations. *arXiv preprint arXiv:2202.13295*, 2022.

[27] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems, NeurIPS*, 2017.

[28] Lei Han, Jiaxing Xu, Jinjie Ni, and Yiping Ke. Rethinking the message passing for graph-level classification tasks in a category-based view. *Eng. Appl. Artif. Intell.*, 143:109897, 2025.

[29] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 639–648. Association for Computing Machinery, 2020.

[30] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 01 2001.

[31] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000-2001. *Bioinform.*, 17:107–108, 2001.

[32] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

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

[34] J. Klicpera, A. Bojchevski, and S Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations, ICLR*, 2019.

[35] Chanhee Kwak, Junyeong Lee, Kyuhong Park, and Heeseok Lee. Let machines unlearn–machine unlearning and the right to be forgotten. 2017.

[36] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. page 177–187. Association for Computing Machinery, 2005.

[37] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. Datasets: A community library for natural language processing. *arXiv preprint arXiv:2109.02846*, 2021.

[38] Wenqin Li, Xinrong Zheng, Ruihong Huang, Mingwei Lin, Jun Shen, and Jiayin Lin. Enhancing privacy protection for online learning resource recommendation with machine unlearning. In *2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 3282–3287, 2024.

[39] Xuefeng Li, Yang Xin, Chensu Zhao, Yixian Yang, and Yuling Chen. Graph convolutional networks for privacy metrics in online social networks. *Applied Sciences*, 10(4), 2020.

[40] Xunkai Li, Bowen Fan, Zhengyu Wu, Zhiyu Li, Rong-Hua Li, and Guoren Wang. Toward scalable graph unlearning: A node influence maximization based approach, 2025.

[41] Xunkai Li, Yulin Zhao, Zhengyu Wu, Wentao Zhang, Rong-Hua Li, and Guoren Wang. Towards effective and general graph unlearning via mutual evolution. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(12):13682–13690, Mar. 2024.

[42] Daniil Likhobaba, Nikita Pavlichenko, and Dmitry Ustalov. Toloker Graph: Interaction of Crowd Annotators, 2023.

[43] Jiaqi Liu, Jian Lou, Zhan Qin, and Kui Ren. Certified minimax unlearning with generalization rates and deletion capacity. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 62821–62852. Curran Associates, Inc., 2023.

[44] Ximeng Liu, Lehui Xie, Yaopeng Wang, Jian Zou, Jinbo Xiong, Zuobin Ying, and Athanasios V Vasilakos. Privacy and security issues in deep learning: A survey. *IEEE Access*, 9:4566–4593, 2020.

[45] Zheyuan Liu, Guangyao Dou, Eli Chien, Chunhui Zhang, Yijun Tian, and Ziwei Zhu. Breaking the trilemma of privacy, utility, and efficiency via controllable machine unlearning. In *Proceedings of the ACM on Web Conference 2024*, pages 1260–1271, 2024.

[46] Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic gnns are strong baselines: Reassessing gnns for node classification. *arXiv preprint arXiv:2406.08993*, 2024.

[47] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2016.

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

[49] Chao Pan, Eli Chien, and Olgica Milenkovic. Unlearning graph classifiers with limited data resources. In *Proceedings of the ACM Web Conference, WWW*, 2023.

[50] Stuart L Pardau. The california consumer privacy act: Towards a european-style privacy regime in the united states. *J. Tech. L. & Pol'y*, 23:68, 2018.

[51] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations, ICLR*, 2020.

[52] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of gnns under heterophily: are we really making progress? *International Conference on Learning Representations, ICLR*, 2023.

[53] Eugenia Politou, Alexandra Michota, Efthimios Alepis, Matthias Pocs, and Constantinos Patsakis. Backups and the right to be forgotten in the gdpr: An uneasy relationship. *Computer Law Security Review*, 34(6):1247–1257, 2018.

[54] Wei Qian, Chenxu Zhao, Wei Le, Meiyi Ma, and Mengdi Huai. Towards understanding and enhancing robustness of deep learning models against malicious unlearning attacks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1932–1942, 2023.

[55] Zongshuai Qu, Tao Yao, Xinghui Liu, and Gang Wang. A graph convolutional network based on univariate neurodegeneration biomarker for alzheimer's disease diagnosis. *IEEE Journal of Translational Engineering in Health and Medicine*, 2023.

[56] Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, SSPR & SPR '08, page 287–297, Berlin, Heidelberg, 2008. Springer-Verlag.

[57] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[58] Yash Sinha, Murari Mandal, and Mohan Kankanhalli. Distill to delete: Unlearning in graph networks with knowledge distillation, 2024.

[59] Daohan Su, Bowen Fan, Zhi Zhang, Haoyan Fu, and Zhida Qin. Dcl: Diversified graph recommendation with contrastive learning. *IEEE Transactions on Computational Social Systems*, 2024.

[60] Henan Sun, Xunkai Li, Zhengyu Wu, Daohan Su, Rong-Hua Li, and Guoren Wang. Breaking the entanglement of homophily and heterophily in semi-supervised node classification. *arXiv preprint arXiv:2312.04111*, 2023.

[61] Jeffrey J. Sutherland, Lee A. O'Brien, and Donald F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure activity relationships. *Journal of Chemical Information and Computer Sciences*, 43(6):1906–1915, 2003.

[62] Jiajun Tan, Fei Sun, Ruichen Qiu, Du Su, and Huawei Shen. Unlink to unlearn: Simplifying edge unlearning in gnns. In *Companion Proceedings of the ACM Web Conference 2024*, page 489–492. Association for Computing Machinery, 2024.

[63] Harry Chandra Tanuwidjaja, Rakyong Choi, Seunggeun Baek, and Kwangjo Kim. Privacy-preserving deep learning on machine learning as a service—a comprehensive survey. *IEEE Access*, 8:167425–167447, 2020.

[64] Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast yet effective machine unlearning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[65] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations, ICLR*, 2018.

[66] Nikil Wale and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, page 678–689, USA, 2006. IEEE Computer Society.

[67] Cheng-Long Wang, Mengdi Huai, and Di Wang. Inductive graph unlearning. *arXiv preprint arXiv:2304.03093*, 2023.

[68] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020.

[69] Zhouxia Wang, Tianshui Chen, Jimmy Ren, Weihao Yu, Hui Cheng, and Liang Lin. Deep reasoning with knowledge graph for social relationship understanding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1021–1028. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[70] Alexander Warnecke, Lukas Pirch, Christian Wressnegger, and Konrad Rieck. Machine unlearning of features and labels, 2023.

[71] Bingzhe Wu, Jintang Li, Junchi Yu, Yatao Bian, Hengtong Zhang, CHaochao Chen, Chengbin Hou, Guoji Fu, Liang Chen, Tingyang Xu, et al. A survey of trustworthy graph learning: Reliability, explainability, and privacy protection. *arXiv preprint arXiv:2205.10014*, 2022.

[72] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning, ICML*, 2019.

[73] Jiancan Wu, Yi Yang, Yuchun Qian, Yongduo Sui, Xiang Wang, and Xiangnan He. Gif: A general graph unlearning strategy via influence function. In *Proceedings of the ACM Web Conference, WWW*, 2023.

[74] Kun Wu, Jie Shen, Yue Ning, Ting Wang, and Wendy Hui Wang. Certified edge unlearning for graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 2606–2617. Association for Computing Machinery, 2023.

[75] Tao Wu, Xinwen Cao, Chao Wang, Shaojie Qiao, Xingping Xian, Lin Yuan, Canyixing Cui, and Yanbing Liu. Graphmu: Repairing robustness of graph neural networks via machine unlearning, 2024.

[76] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

[77] Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, and Philip S. Yu. Machine unlearning: A survey. *ACM Comput. Surv.*, 56(1), August 2023.

[78] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? International Conference on Learning Representations, ICLR, 2019.

[79] Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. Arcane: An efficient architecture for exact machine unlearning. In *IJCAI*, volume 6, page 19, 2022.

[80] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 1365–1374. Association for Computing Machinery, 2015.

[81] Tzu-Hsuan Yang and Cheng-Te Li. When contrastive learning meets graph unlearning: Graph contrastive unlearning for link prediction. In *2023 IEEE International Conference on Big Data (BigData)*, pages 6025–6032, 2023.

[82] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning, ICML*, 2016.

[83] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Trans. Graph.*, 35(6), 2016.

[84] Lu Yi and Zhewei Wei. Scalable and certifiable graph unlearning: Overcoming the approximation error barrier, 2024.

[85] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. *Graph transformer networks*. Curran Associates Inc., 2019.

[86] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International conference on learning representations, ICLR*, 2020.

[87] Jiahao Zhang. Graph unlearning with efficient partial retraining. In *Companion Proceedings of the ACM Web Conference 2024*, page 1218–1221. Association for Computing Machinery, 2024.

[88] Zhexin Zhang, Junxiao Yang, Pei Ke, Shiyao Cui, Chujie Zheng, Hongning Wang, and Minlie Huang. Safe unlearning: A surprisingly effective and generalizable solution to defend against jailbreak attacks, 2024.

[89] Zhikun Zhang, Min Chen, Michael Backes, Yun Shen, and Yang Zhang. Inference attacks against graph neural networks. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4543–4560. USENIX Association, 2022.

[90] Wenyue Zheng, Ximeng Liu, Yuyang Wang, and Xuanwei Lin. Graph unlearning using knowledge distillation. In *Information and Communications Security: 25th International Conference, ICICS 2023, Tianjin, China, November 18–20, 2023, Proceedings*, page 485–501. Springer-Verlag, 2023.

[91] Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and TechnoLoGy*, 13(1):1–54, 2022.

[92] Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations, ICLR*, 2021.

[93] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 06 2018.

[94] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2847–2856. Association for Computing Machinery, 2018.

[95] Daniel Zügner, Oliver Borchert, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on graph neural networks: Perturbations and their patterns. *ACM Transactions on Knowledge Discovery from Data*, 14(5):1–31, 2020.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We ensure that the abstract and introduction accurately reflect the paper's contributions, comprehensively outlining the current development of the graph unlearning field and establishing an innovative benchmark, with details of contributions provided in the final paragraph of the introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our work in the conclusion section of the paper.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide theoretical analysis for the GU algorithm's complexity, supported by detailed proofs and comprehensive explanations in the paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We thoroughly describe the experimental setup for the OpenGU algorithm (including datasets, hyperparameters, and optimizers) and reproduction steps, providing anonymized code and structured documentation in the supplemental material to ensure reproducibility.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide an anonymized access link for the OpenGU code, accompanied by structured documentation detailing the runtime environment, hyperparameters, and reproduction scripts. Additionally, we include detailed hyperparameter spaces for each GU algorithm in the appendix to ensure faithful reproduction of experimental results.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We detail data splits, model architecture, and optimizer types in the appendix, and provide hyperparameters and their selection methods in Table 8, ensuring results are understandable and reproducible.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report the mean and standard deviation of multiple runs in all relevant figures and tables of the experimental section, with their interpretation explained in the main text.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: In the appendix of our paper, we provide a detailed description of the hardware and software used in the experiments, including GPU type, memory, etc.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in this paper conforms in every respect with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: The main focus of this paper is a review of existing GU algorithms and the introduction of an innovative benchmark. As this work does not directly address concrete societal applications at this stage, it has no immediate societal impact.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper presents the research on the OpenGU Benchmark and does not pose any potential risks of misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All existing assets used in this paper (including code and datasets) have been properly credited to their creators or original owners, and the license and terms of use for each asset have been explicitly mentioned.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: This paper introduces new assets that are accompanied by comprehensive documentation detailing code usage instructions, model training details, license information, etc. These documents and the code have been provided via an anonymized URL.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper's research does not involve any crowdsourcing experiments or studies with human subjects. Our work primarily focuses on the field of graph unlearning.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Our research presented in this paper does not include any studies with human subjects. The main focus of our work is in the area of graph unlearning.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The core methodology, code development, and manuscript writing of this paper were entirely completed independently by the authors. Large language models were only used to enhance the quality of the writing.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Contents

A Background

A.1 Graph Neural Networks Propagation Mechanism

In this section, we primarily explore Message Passing Neural Networks, a prevalent paradigm in GNNs for effectively learning graph-structured data. These models harness the inherent structural properties of graphs to derive robust node representations. Broadly, such GNNs are characterized by three fundamental components: initialization, aggregation, and update, providing a cohesive framework to capture and model relational dependencies within graphs.

1.Initialization. To initialize node representations in a graph, each node is typically assigned an embedding based on its features or attributes. This process can be formalized as:

$$h_v = x_v \quad \forall v \in \mathcal{V}. \tag{1}$$

2.Aggregation. To capture information from the neighborhood of node u, an aggregation function $f_{Agg}^{(k-1)}$ is applied to combine the embeddings of $\mathcal{N}(u)$ from the previous iteration. Common aggregators include permutation-invariant functions like sum, mean, or max, ensuring that the aggregated result is independent of the order of neighbors. This operation can be expressed as:

$$m_u^{(k)} = f_{Agg}^{(k-1)}(\{h_v^{(k-1)}, v \in \mathcal{N}(u)\}). \tag{2}$$

3.Update. This process enables each node to refine its representation using $f_{Up}^{(k-1)}$, which combines the node's embedding and the neighborhood's aggregated message. Specifically, the updated embedding for node u is computed as follows:

$$h_u^{(k)} = f_{Up}^{(k-1)}(h_u^{(k-1)}, m_{\mathcal{N}(u)}^{(k-1)}). \tag{3}$$

The combination of initialization, aggregation, and update functions forms the foundation of GNNs. Through iterative refinement, node embeddings progressively incorporate structural and feature information from their local neighborhoods. Consequently, this enables GNNs to effectively learn comprehensive and highly expressive representations for a diverse range of downstream tasks.

A.2 Graph Neural Networks Downstream Tasks

Building upon the iterative process of node representation, the resulting node embeddings $h_u^{(k)}$ at the final iteration k serve as the foundation for downstream tasks. These refined embeddings are rich in information and capture both local and global graph structures, making them highly suitable for various graph-based prediction tasks. The downstream tasks typically involve node classification, link prediction, and graph classification, each utilizing these node embeddings in different ways.

Node classification aims to assign labels to individual nodes in a graph by utilizing their learned representations $h^{(k)}$. In this task, a GNN generates these representations by iteratively refining node features based on neighborhood information across k layers. The final representations $h^{(k)}$ encapsulate both node-specific attributes and relational patterns, which are then processed by a classifier—typically a fully connected layer—to predict labels. This approach excels in capturing dependencies within graph-structured data. For instance, in financial transaction networks, where nodes represent user accounts and edges signify monetary transfers, node classification can detect fraudulent behavior. By leveraging $h^{(k)}$ derived from account features (e.g., transaction volume, frequency) and connectivity (e.g., links to previously flagged accounts), the classifier can accurately identify suspicious nodes, improving fraud detection in real-world financial systems.

Link prediction aims to infer the likelihood of edges between nodes by leveraging their embeddings and the graph's structural properties. In this task, GNNs utilize node embeddings, such as $h^{(k)}$, which encode both individual node features and relational information derived from the graph's topology. These embeddings are typically combined—via concatenation, dot product, or distance metrics—and processed by a predictive model to estimate the probability of an edge existing between node pairs. This capability is pivotal for uncovering latent relationships in graph-structured data. For example, in biomedical networks, where nodes represent drugs and edges denote experimentally validated interactions, link prediction can identify potential drug-drug interactions. By analyzing embeddings that capture drug properties (e.g., chemical composition, target proteins) and network patterns (e.g., shared interaction profiles with other drugs), GNNs can predict previously unobserved edges, facilitating the discovery of novel therapeutic combinations, such as synergistic drug pairs for complex diseases like cancer or diabetes, thus supporting drug development and clinical research.

Graph classification entails predicting a label for an entire graph by synthesizing node-level information into a comprehensive representation. GNNs accomplish this by producing node embeddings $h^{(k)}$ that encode individual features and structural relationships, then aggregating these embeddings through a readout function f_{Read}, such as summation, averaging, or advanced pooling techniques, to form a graph-level embedding. This unified representation is subsequently processed by a classifier to assign the graph's label, enabling the model to capture global characteristics from local interactions. For instance, in chemical compound classification, molecules are represented as graphs, with nodes corresponding to atoms and edges indicating chemical bonds. Atoms like carbon or nitrogen contribute properties such as electronegativity and atomic radius, while bonds like covalent links define the molecular structure. Graph classification can predict toxicity by leveraging embeddings that reflect these atomic properties and structural configurations, including aromatic rings or functional groups, to distinguish toxic compounds from non-toxic ones. This approach aids applications like drug safety evaluation or environmental hazard screening by identifying harmful substances in chemical libraries.

A.3 Unlearning Requests

Node unlearning targets the selective removal of individual nodes from a graph, addressing the distinct challenge of excising specific entities, along with their associated features and incident edges, while preserving the graph's remaining structure and functionality. Unlike edge unlearning, which modifies relationships, or feature unlearning, which alters attributes, node unlearning entails the complete elimination of a node's representation, ensuring its influence is fully eradicated from the trained GNN model. This process adjusts the model's parameters or embeddings to exclude the target node's contributions to predictions, such as node labels or graph-level outputs, without

disrupting the connectivity or learned patterns of unaffected nodes. Its importance stems from enabling precise, entity-level data deletion, critical for privacy-sensitive applications. For instance, in social network analysis, where nodes represent individual users and edges denote interactions like follows or messages, node unlearning ensures that upon a user's account deletion request, their profile data, such as posts and preferences, and network ties, such as connections to other users, are entirely removed from the model's recommendation system. This prevents the deleted user's behavior from influencing future outputs, aligning with privacy regulations like the GDPR's "right to be forgotten". By isolating node-specific effects, this method sustains model utility for the retained graph, underscoring its distinct role in graph unlearning and its applicability to domains requiring entity-level control, such as personalized marketing or healthcare patient record management.

Edge unlearning focuses on the selective removal of relationships between nodes within a graph, targeting the erasure of specific edges while preserving node attributes and the broader graph structure. Distinct from node unlearning, which eliminates entire entities, edge unlearning isolates and excises connections between nodes, adjusting the trained GNN model to exclude their influence on relational predictions. This process involves updating the model's parameters or adjacency representation to reflect the absence of targeted edges, ensuring that the altered connectivity no longer affects downstream tasks. Its significance lies in its ability to refine graph relationships dynamically without disrupting the integrity of individual nodes. For instance, in financial fraud detection systems, where nodes represent user accounts and edges denote monetary transactions, edge unlearning proves invaluable when certain transactions are flagged as erroneous, fraudulent, or compromised due to security breaches. Consider a scenario where an edge links two accounts involved in a suspected phishing scam: edge unlearning can remove this transactional relationship from the GNN-based fraud detection model, preventing it from influencing risk assessments or clustering patterns, while retaining the accounts' individual features, such as transaction history or balance data. This targeted removal enhances model accuracy by eliminating misleading connections and supports compliance with data protection standards. By maintaining node-level information intact, edge unlearning ensures the model's utility for legitimate transactions, highlighting its unique role in graph unlearning and its applicability to domains like social network moderation or supply chain management.

Feature unlearning targets the removal of specific node features from a graph, enabling the selective deletion of attribute-based information tied to individual nodes while preserving their structural roles and other properties. Distinct from node unlearning, which excises entire entities, or edge unlearning, which removes connections, feature unlearning eliminates sensitive or extraneous attributes from a node's feature set, adjusting the trained GNN model to exclude their influence on predictions without altering graph topology. This can involve partial removal, targeting specific features like personal identifiers, or full deletion of a node's attribute vector, depending on the unlearning goal. Its significance lies in providing fine-grained control over data, essential for privacy and relevance in predictive tasks. For example, in healthcare networks, where nodes represent patients and edges denote shared treatments, feature unlearning can erase sensitive attributes, such as genetic data, from a GNN-based risk prediction model, ensuring compliance with privacy regulations while retaining features like age for continued utility in disease forecasting. This selective approach maintains model effectiveness, highlighting feature unlearning's unique role in graph unlearning and its value in domains requiring attribute-specific adjustments, such as advertising or cybersecurity.

A.4 GU Method Taxonomy

In the context of GU, a diverse array of specialized methods has been developed to address the complex challenges of selectively forgetting specific nodes, edges, or features in a trained model while preserving the integrity of the remaining graph information. In Partition-based methods, GraphEraser [11], GUIDE [67], and GraphRevoker [87] draw inspiration from SISA [4] to implement different partitioning strategies, enabling training and unlearning on independent shards. In IF-based methods, GIF [73], CGU [15], CEU [74], and others [49, 23, 84] leverage rigorous mathematical formulations to quantify the impact of data removal on model, allowing for efficient model updates without full retraining. In Learning-based algorithms, GNNDelete [13], MEGU [41], SGU [40], and others [58, 90, 81] achieve a trade-off between forgetting and reasoning with specialized loss functions. As for Projection-based algorithm, Projector [18] adapts to unlearning by orthogonally projecting the weights into a different subspace, thereby mitigating the influence of targeted data. Finally, Structure-based method UtU [62] manipulate the graph structure directly without the extensive retraining or complex optimizations procedures. These methods collectively aim to balance effectiveness

Table 4: Statistical Overview of Datasets for Node and Edge-Level Tasks in OpenGU Benchmark.

Datasets	Nodes	Edges	Features	Classes	Type	Description
Cora	2,708	5,278	1,433	7	Homophily	Citation Network
CiteSeer	3,327	4,732	3,703	6	Homophily	Citation Network
PubMed	19,717	44,338	500	3	Homophily	Citation Network
DBLP	17,716	52,867	1,639	4	Heterophily	Citation Network
ogbn-arxiv	169,343	1,166,243	128	40	Homophily	Citation Network
CS	18,333	81,894	6,805	15	Homophily	Co-author Network
Physics	34,493	247,962	8,415	5	Homophily	Co-author Network
Photo	7,487	119,043	745	8	Homophily	Co-purchasing Network
Computers	13,381	245,778	767	10	Homophily	Co-purchasing Network
ogbn-products	2,449,029	61,859,140	100	47	Homophily	Co-purchasing Network
Chameleon	2,277	36,101	2,325	5	Heterophily	Wiki-page Network
Squirrel	5,201	216,933	2,089	5	Heterophily	Wiki-page Network
Actor	7,600	29,926	931	5	Heterophily	Actor Network
Minesweeper	10,000	39,402	7	2	Homophily	Game Synthetic Network
Tolokers	11,758	519,000	10	2	Homophily	Crowd-sourcing Network
Roman-empire	22,662	32,927	300	18	Heterophily	Article Syntax Network
Amazon-ratings	24,492	93,050	300	5	Heterophily	Rating Network
Questions	48,921	153,540	301	2	Homophily	Social Network
Flickr	89,250	899,756	500	7	Heterophily	Image Network

in unlearning with model efficiency and robustness, ensuring minimal degradation in predictive performance for retained data. Below, we provide a comprehensive overview of these approaches, highlighting their mechanisms and contributions to advancing GU capabilities.

B Datasets

In this section, we focus on the datasets relevant to graph unlearning, beginning with an introduction to the diverse datasets integrated within OpenGU. These datasets comprehensively address the requirements for node, edge, and graph level tasks in graph unlearning scenarios. We also provide detailed data statistics (see Table 4 and Table 5) and thorough descriptions of these datasets. Furthermore, we offer an in-depth explanation of the two commonly used dataset partitioning approaches (e.g., transductive and inductive) elucidating their characteristics and applications.

B.1 Dataset Overview

Datasets for Node-level and Edge-level Tasks

Graph unlearning scenarios are inherently data-driven, underscoring the critical need for a meticulous selection of datasets to effectively evaluate GU strategies. The performance of GU methods, whether applied to node-related or edge-related tasks, hinges on the diversity and relevance of the data used for assessment. To this end, we have carefully curated a collection of 19 datasets, each chosen to represent distinct domains and graph structures [33]. Citation Networks, including Cora, CiteSeer, and PubMed [82], offer a robust platform for studying academic citation patterns, characterized by extensive node interconnections. Co-author Networks, represented by CS and Physics [57], model collaborative relationships among researchers, providing insights into scholarly authorship networks. For visual data, Image Networks such as Flickr [86] capture connections within image-sharing communities. E-commerce and Product Networks, encompassing Photo, Computers [57], ogbn-products [32], and Amazon-ratings [52], reflect consumer-product interactions, highlighting purchasing behaviors and preferences. Scientific Networks, such as DBLP [85] and ogbn-arxiv [32], focus on publication ecosystems, revealing trends in research collaboration and dissemination. To widen the evaluation scope, we incorporate Squirrel and Chameleon [51], which represent webpage networks with complex hyperlink structures, and Actor [51], which maps film industry relationships through actor collaborations. Digital engagement is addressed with Minesweeper [52], an online gaming dataset capturing player interactions, and Tolokers [52], drawn from a crowdsourcing platform

Table 5: Statistical Overview of Datasets for Graph-Level Tasks in OpenGU Benchmark.

Datasets	Graphs	Nodes	Edges	Features	Classes	Description
MUTAG	188	17.93	19.79	7	2	Compounds Network
PTC-MR	344	14.29	14.69	18	2	Compounds Network
BZR	405	35.75	38.36	56	2	Compounds Network
COX2	467	41.22	43.45	38	2	Compounds Network
DHFR	467	42.43	44.54	56	2	Compounds Network
AIDS	2,000	15.69	16.20	42	2	Compounds Network
NCI1	4,110	29.87	32.30	37	2	Compounds Network
ogbg-molhiv	41,127	25.50	27.50	9	2	Compounds Network
ogbg-molpcba	437,929	26.00	28.10	9	2	Compounds Network
ENZYMES	600	32.63	62.14	21	6	Protein Network
DD	1,178	284.32	715.66	89	2	Protein Network
PROTEINS	1,113	39.06	72.82	4	2	Protein Network
ogbg-ppa	158,100	243.40	2,266.10	4	37	Protein Network
IMDB-BINARY	1,000	19.77	96.53	degree	2	Movie Network
IMDB-MULTI	1,500	13.00	65.94	degree	3	Movie Network
COLLAB	5,000	74.49	2,457.78	degree	3	Collaboration Network
ShapeNet	16,881	2,616.20	KNN	3	50	Point Cloud Network
MNISTSuperPixels	70,000	75.00	1,393.03	1	10	Super-pixel Network

where edges indicate task-based collaborations. Additionally, historical and social Q&A dynamics are included through Roman-empire and Questions [52], respectively, enhancing the diversity of our dataset collection. This thoughtfully assembled set ensures a broad and nuanced assessment of GU methods across multiple domains and graph characteristics.

Datasets for Graph-level Tasks

In the context of graph classification tasks, OpenGU incorporates a diverse collection of 18 datasets drawn from multiple domains, enabling a comprehensive evaluation of graph unlearning strategies. These datasets are carefully selected to reflect varied graph structures and application areas, ensuring robust testing across a wide array of real-world and theoretical scenarios. Specifically, the compound networks, including MUTAG, PTC-MR, BZR, COX2, DHFR, AIDS, NCI1, ogbg-molhiv, and ogbg-molpcba [20, 30, 61, 56, 66, 32], focus on molecular structures and their chemical properties, providing a robust and detailed basis for analyzing small molecules and their complex interactions. The protein networks, such as ENZYMES, DD, PROTEINS, and ogbg-ppa [8, 22, 31, 32], center on biological data, capturing protein interactions and enzymatic functions that are critical to bioinformatics research and drug discovery. In the realm of social and community structures, the movie networks, comprising IMDB-BINARY and IMDB-MULTI [80], explore film collaborations, while the collaboration network COLLAB [36] models scientific co-authorship patterns, offering valuable insights into social dynamics. Additionally, ShapeNet [83] and MNISTSuperPixels [47] address distinct tasks involving 3D shape recognition and image superpixel segmentation, respectively, broadening the scope to include spatial and visual graph representations. This thoughtfully curated set of datasets ensures that OpenGU effectively supports the assessment of graph classification methods across a wide range of practical and theoretical contexts, facilitating advancements in GU research.

B.2 Dataset Details

In OpenGU, the selected datasets play a pivotal role in benchmarking and evaluating the performance of GU methods under a range of realistic and challenging conditions. These datasets, chosen for their diversity in structure and application domain, enable a comprehensive analysis of the methods' effectiveness, efficiency, and robustness. By capturing various graph characteristics, they provide an essential foundation for assessing unlearning strategies and comparing their adaptability across different scenarios. The following section provides a detailed overview of each dataset:

Cora, **CiteSeer**, and **PubMed** [82] are among the most widely utilized citation network datasets in the GU field. In these datasets, nodes represent research papers, and edges indicate citation relationships between them. Each node is characterized by a bag-of-words feature and is uniquely

associated with a specific category. These datasets are frequently employed for tasks such as node classification, providing a reliable basis for evaluating model performance across citation networks.

DBLP [85], derived from the extensive DBLP Computer Science Bibliography, offers a unique view into academic collaboration by modeling a co-authorship network. In this dataset, nodes represent individual authors, and edges indicate co-authored publications, capturing dynamic, multi-disciplinary research networks. Each node is labeled according to research domains, enabling robust experiments in node classification, clustering, and link prediction.

ogbn-arxiv [32] is a comprehensive academic graph derived from the Microsoft Academic Graph (MAG) [68], designed to facilitate machine learning tasks on graph data. It represents a paper citation network of arXiv papers, capturing the scholarly communication and influence within the field of computer science. The graph structure of ogbn-arxiv is characterized by nodes representing scientific papers and edges representing citations between them, reflecting the academic referencing relationships. It offers a challenging and realistic testbed for the development and evaluation of GNNs and other machine learning models on graph data.

CS and **Physics** [57] are co-authorship graphs derived from the Microsoft Academic Graph, specifically designed for node classification tasks. These datasets represent academic collaborations where nodes correspond to authors and edges indicate co-authorship on papers. In both datasets, node features are represented by paper keywords associated with each author's publications and class labels denote the most active fields of study for each author, providing a rich semantic profile for classification purposes.

Flickr [86] encapsulates the structure and properties of online images, with each node representing an individual image. The dataset is characterized by its rich feature set, where nodes are described by a comprehensive set of features extracted from image metadata, such as comments, likes, and group affiliations. Edges in the Flickr dataset signify connections between images, which are based on shared attributes or interactions. The Flickr dataset stands as a significant resource for researchers and practitioners in the field of graph neural networks.

Photo and **Computers** [57] datasets are derived from the Amazon co-purchase graph, representing the relationships between products frequently bought together. Nodes in these datasets symbolize individual products, while edges indicate co-purchase frequency, reflecting the market dynamics and consumer behavior on Amazon's e-commerce platform. The Photo dataset focuses on photographic equipment, while the Computers dataset centers on computer-related products, providing a nuanced view into consumer purchasing patterns within these specific domains.

ogbn-products [32] dataset, part of the Open Graph Benchmark (OGB), is an extensive co-purchasing network that captures the intricate relationships between products. Nodes in this dataset symbolize products and edges indicate that two products are frequently bought together. Node features are extracted from product descriptions, transformed into a bag-of-words representation. This dataset is renowned for its large scale and complex structure, which makes it an exemplary testbed for large-scale graph learning applications. It serves as a critical benchmark for assessing the scalability and effectiveness of graph algorithms, providing a realistic challenge for models to handle vast amounts of data and intricate connections.

Chameleon and **Squirrel** [51] datasets, sourced from Wikipedia, represent webpage networks where nodes correspond to individual pages and edges indicate mutual links. Each node's features are derived from webpage content, capturing unique structural characteristics specific to each network. These datasets are particularly valuable for evaluating heterophilic graph learning methods, as nodes with similar labels are less densely connected, challenging traditional GNNs and encouraging the development of models that effectively handle low homophily settings.

Actor [51] dataset represents a actor network within the film industry, where each node corresponds to an actor and edges represent co-appearances in the same movie. Each node is characterized by features based on personal and professional attributes, providing a basis for relational insights. The dataset's structure, with nodes labeled based on actor roles or genres, is well-suited for evaluating algorithms in low-homophily settings, challenging GNNs to accurately classify nodes when similar labels are sparsely connected.

Minesweeper [52] dataset, sourced from an online gaming platform, models interactions within the Minesweeper game environment. In this graph, each node represents a player, and edges denote

collaborations or competitions between players during gameplay sessions. Node features are derived from player statistics and gameplay metrics, providing a comprehensive view of user behavior patterns. This dataset is valuable for analyzing patterns in social connectivity and behavior, making it useful for assessing unlearning strategies in dynamic, interaction-driven networks.

Tolokers [52] dataset, drawn from a crowdsourcing platform [42], represents worker interactions within collaborative tasks. Nodes correspond to individual workers, with edges indicating collaborations on shared tasks. The dataset captures complex relationships, as it aims to predict which workers may face restrictions or bans based on past activities.

Roman-empire [52] dataset is a dependency graph from the Roman Empire Wikipedia article [37], with nodes as words and edges as dependencies or word adjacencies. It is used for node classification based on syntactic roles, offering insights into language structure and word relationships within a historical text context, enabling advanced analysis of linguistic patterns.

Amazon-ratings [52] dataset captures user interactions with products on Amazon, forming a graph where nodes are items and edges represent user ratings. This dataset is instrumental for tasks like predicting user preferences and understanding product affinities, offering a real-world testbed for GNNs to operate at scale in dynamic e-commerce environments.

Questions [52] dataset represents interactions within a community Q&A platform, where nodes correspond to users and edges indicate interactions such as asking, answering, or commenting on questions. This dataset reflects user engagement and information exchange patterns, making it suitable for evaluating algorithms focused on social dynamics and collaborative learning. Its structure provides valuable insights into the spread of information and influence within online communities.

MUTAG [20] is a chemical compounds dataset focusing on the mutagenicity of molecules, where each graph represents a molecule and the nodes represent atoms. The task is to predict whether the molecule is mutagenic or not. The dataset consists of 188 graphs with 7 distinct classes, making it a benchmark for graph classification in the field of cheminformatics.

PTC-MR [30] is a dataset used for mutagenic toxicity prediction, where each graph represents a chemical compound, and the nodes correspond to atoms. The task is to predict whether a compound is mutagenic based on its chemical structure. With 344 graphs, it includes two distinct classes and is commonly used for evaluating GNNs in bioinformatics applications.

BZR [61] dataset contains chemical compounds, where each molecule is represented as a graph, with nodes representing atoms and edges representing bonds. The classification task is to determine whether a compound can act as a potent estrogen receptor binder. It includes 405 graphs, with active and inactive classes balanced for robust evaluation.

COX2 [61] is a dataset related to the inhibition of the COX-2 enzyme, an important target for anti-inflammatory drugs. It consists of chemical compounds where the nodes represent atoms and the edges represent bonds. The task is to predict whether a compound inhibits COX-2 or not. The dataset includes 467 graphs and is used for evaluating graph neural networks in drug discovery.

DHFR [61] is a dataset of 1,634 chemical compounds used to predict inhibition of the dihydrofolate reductase (DHFR) enzyme, a target for antimicrobial drugs. The dataset is significant for its application in drug design, where GNNs are used to model molecular structures and predict bioactivity. The task is binary classification (inhibitor or non-inhibitor), making it a valuable benchmark in computational biology for advancing therapeutic development..

AIDS [56] dataset, which contains 2,000 chemical compounds, is used to predict the activity of these compounds against the HIV virus. It is a key dataset in drug discovery, particularly for anti-HIV drug. Each molecule is represented as a graph where nodes are atoms and edges are bonds. The binary classification task helps evaluate GNNs in pharmaceutical research and HIV treatment.

NCI1 [66] dataset consists of 4,110 chemical compounds and is used for cancer cell line screening. Each graph represents a molecule, and the task is to classify whether a compound is active or inactive against cancer cells. It is one of the largest datasets in molecular graph classification, playing a significant role in drug discovery, particularly in identifying potential anticancer compounds.

ogbg-molhiv [32] dataset, part of the Open Graph Benchmark (OGB), consists of 41,127 graphs representing chemical compounds. The goal is to predict whether a compound inhibits HIV. This large-

scale dataset is important for advancing GNNs in drug discovery, providing a real-world application in the search for anti-HIV drugs. It is valuable due to its scale and complexity in bioinformatics.

ogbg-molpcba [32] dataset is another graph dataset from the Open Graph Benchmark (OGB) focused on predicting the biological activity of compounds across multiple targets. With over 437,000 compounds, this dataset presents a multi-label classification task, where each compound can have multiple activity labels. It is important for developing models capable of handling large-scale, multi-label classification tasks in cheminformatics.

ENZYMES [8] dataset is used for enzyme classification, where each graph represents a protein, and the task is to predict the enzyme class to which it belongs. The dataset contains 600 graphs and is significant for understanding protein structures and functions in computational biology. It provides a benchmark for GNNs in biological and biomedical applications, particularly for enzyme function prediction across diverse biological systems and complex proteomic studies.

DD [22] dataset consists of 1,178 protein-protein interaction graphs, where nodes represent proteins and edges denote interactions between them. The task is binary classification, determining whether two proteins interact. This dataset is important in the study of biological systems, particularly in understanding cellular processes and identifying potential therapeutic targets.

PROTEINS [31] dataset contains 1,113 graphs, each representing a protein, with nodes corresponding to amino acids and edges indicating spatial proximity. The classification task is to distinguish between enzymes and non-enzymes. This dataset is significant for studying protein structure and function, which is fundamental in drug discovery and molecular biology.

ogbg-ppa [32] dataset, from the Open Graph Benchmark (OGB), includes 158,100 graphs representing pairs of proteins and their potential interactions. The task is binary classification to predict whether a pair of proteins interacts. It is particularly valuable for exploring protein interaction prediction, which plays a critical role in understanding biological networks and disease mechanisms.

IMDB-BINARY [80] dataset consists of 1,000 graphs, each representing a movie and the relationships between users and movies. The task is to predict whether a movie belongs to a specific genre based on interactions. This dataset is relevant for applications in social network analysis and movie recommendation systems leveraging user behavior patterns.

IMDB-MULTI [80] dataset, derived from IMDB, includes 1,500 graphs, where each graph represents a movie and contains co-occurrence relationships between users and movies. The task is to predict multiple genres for each movie. It is important for multi-label classification tasks in social network analysis and content-based recommendation systems.

COLLAB [36] dataset contains 5,000 graphs, each representing a scientific collaboration network, where nodes represent researchers and edges represent co-authorships. The classification task involves predicting the research domain of a collaboration network. This dataset is valuable for studying academic collaboration patterns and community detection in social networks.

ShapeNet [83] dataset is a large-scale collection of 3D models representing various object categories, including chairs, tables, and airplanes. Each object is represented as a graph where nodes correspond to geometric features, and edges define spatial relationships. The task is to classify objects into categories, making it important for 3D shape recognition and computer vision applications.

MNISTSuperPixels [47] dataset is a graph-based version of the famous MNIST digit classification dataset, where each digit is represented as a graph of superpixels. The task is to classify the digits (0-9) based on the graph structure. It is useful for exploring how graph-based models can be applied to image classification tasks, particularly in the context of digit recognition.

B.3 Transductive and Inductive Settings

To enhance adaptability and evaluation rigor, OpenGU introduces standardized preprocessing protocols addressing critical gaps in dataset splitting and inference flexibility. Current GU methods lack unified splitting strategies, often relying on fixed ratios that constrain experimental design. OpenGU resolves this by enabling customizable data splits to accommodate diverse methodological needs. Additionally, we extend preprocessing to support both transductive and inductive inference scenarios, allowing datasets optimized for a single paradigm to function under dual frameworks. These enhance-

ments ensure broader performance assessment across varying experimental and inference contexts, solidifying OpenGU as a comprehensive benchmark for robust GU evaluation.

In the **transductive** setting, models leverage the global topological information of an entire graph—including both training and test nodes and edges—during the training phase. However, the labels of the test nodes or edges are withheld and predicted only during inference. This approach assumes that the graph's structure, such as node connectivity, is fully accessible during training, while the test labels remain masked. Transductive learning is frequently employed in node- or edge-level tasks within static graphs, such as node classification in citation networks. For instance, a model might exploit the structural relationships across all nodes in an academic citation graph during training to predict labels for a designated subset of nodes. By harnessing global structural patterns, the transductive approach enhances local predictions, though its applicability is confined to generalizing within the boundaries of the observed graph structure.

In the **inductive** setting, models are trained on specific subgraphs or training instances, requiring them to generalize to entirely unseen graphs or subgraphs at inference time. In this paradigm, models learn transferable patterns from local or partial graph structures without prior knowledge of the test graph's topology. This setting is particularly suited to dynamic graph scenarios, such as evolving social networks with newly added users, or cross-graph tasks, like molecular property prediction. For example, in molecular graph classification, a model trained on a subset of molecular structures must infer properties for novel, unseen molecular graphs. The inductive approach prioritizes generalization to new structures, but its performance can be more sensitive to graph heterogeneity, relying heavily on parameterized learning rather than predefined topological constraints.

These two settings cater to fundamentally different use cases: transductive learning optimizes performance on static graphs with known entities, while inductive learning addresses generalization in open-world contexts. OpenGU facilitates both paradigms through adaptable preprocessing capabilities, enabling robust evaluation across a wide range of inference scenarios.

C GNN Backbone

C.1 Backbone Overview

For the traditional GNNs, we implement widely-recognized models such as GCN [33], GAT [65], GIN [78] and others [12, 5, 29]. Sampling GNNs include GraphSAGE [27], GraphSAINT [86] and Cluster-gcn [14]. Additionally, to accommodate GU methods which rely on linear-GNN, we further incorporate scalable, decoupled GNN models into the benchmark, specifically SGC [72], SSGC [92], SIGN [25] and APPNP [34]. These models offer scalability advantages and efficient decoupling for handling larger datasets and supporting diverse GU methods.

C.2 Backbone Details

In this section, we provide an in-depth overview of the GNNs utilized within OpenGU. These foundational architectures play a critical role in implementing and benchmarking GU strategies, as they offer diverse structures and mechanisms that impact the model's effectiveness, efficiency, and adaptability. By detailing each backbone, we aim to give readers a clearer understanding of the underlying model choices and their relevance to unlearning process.

GCN [33] is a foundational GNN model that effectively captures graph structure by recursively aggregating feature information from neighboring nodes through a first-order approximation of spectral convolutions. GCN achieves strong performance in tasks such as semi-supervised node classification, link prediction, and community detection.

GCNII [12] is an extension of GCN that improves expressiveness by incorporating higher-order graph convolutions while avoiding over-smoothing. It introduces a novel initialization and an implicit regularization mechanism, making it robust for deep graph networks. GCNII excels in tasks involving large and sparse graphs, such as semi-supervised node classification and graph regression, while maintaining remarkable stability in deep models with consistently reliable performance.

LightGCN [29] is a light-weight GNN model optimized for collaborative filtering tasks, specifically in recommender systems. Unlike traditional GCNs, it simplifies the convolution operation by removing unnecessary transformations and focusing solely on neighborhood aggregation. LightGCN

delivers state-of-the-art performance in tasks such as link prediction and recommendation while maintaining computational efficiency.

GAT [65] significantly enhances traditional GNNs by introducing attention mechanisms to carefully weigh the relative importance of neighboring nodes when aggregating features. This inherently attention-driven approach makes GAT especially effective in graphs with complex node relationships, offering a truly robust solution for node classification and link prediction.

GATv2 [5] is an enhanced version of the Graph Attention Network (GAT) that improves the flexibility and expressiveness of attention mechanisms. GATv2 introduces a more generalized attention mechanism, allowing for better handling of noisy graph data and fine-grained aggregation of neighborhood information. It has shown superior performance in node classification and graph classification tasks, especially for graphs with heterogeneous features.

GIN [78] is designed to capture structural distinctions in graphs with maximal expressive power. By applying a simple sum aggregator followed by a powerful multi-layer perceptron, GIN achieves provably injective mapping of neighborhood features, allowing it to differentiate between complex graph structures considerably more effectively than other models.

GraphSAGE [27] is a powerful framework that generalizes GNNs by learning node embeddings through sampling and aggregating features from a fixed-size neighborhood. Unlike traditional GNNs, which require access to the entire graph, GraphSAGE supports efficient inductive learning by training on sampled node neighborhoods, enabling it to scale effectively to large graphs and adapt to unseen nodes in dynamic environments.

GraphSAINT [86] is a scalable GNN model designed to handle large-scale graphs by employing efficient sampling methods. It combines graph sampling with mini-batch training, allowing the model to operate on subgraphs rather than the entire graph, which greatly improves training efficiency. GraphSAINT achieves competitive performance in node classification and graph classification tasks, particularly for large graphs with millions of nodes.

Cluster-gcn [14] is a innovative GNN model designed to handle intricate graph clustering tasks by exploiting node-level similarity structures. It divides the graph into multiple subclusters and performs learning on these subgraph clusters to improve performance and scalability. Cluster-gcn has been demonstrated to achieve remarkably high accuracy in clustering-based tasks while maintaining excellent computational efficiency in large-scale graphs.

SGC [72] is a streamlined variant of GCN that removes non-linearity between layers, significantly reducing computational complexity. By collapsing multiple layers into a single linear transformation, SGC preserves essential neighborhood information while enhancing scalability, making it particularly suitable for large-scale node classification tasks.

SSGC [92] presents an extension of the SGC model by incorporating self-supervised learning through contrastive loss, enabling more robust representation learning. This method preserves the efficient linear architecture of SGC while simultaneously enhancing the learned embeddings by contrasting positive and negative samples, thereby capturing meaningful graph structures.

SIGN [25] is designed for large-scale graphs, leveraging a pre-computation strategy to improve efficiency. It processes multiple hops of neighborhood information independently, storing them as separate feature channels. By avoiding recursive message passing during training, SIGN achieves scalability and faster computation, making it well-suited for tasks on massive graph datasets.

APPNP [34] introduces a novel approximation of personalized PageRank, combining random walk-based methods with graph neural networks. APPNP performs graph-based label propagation with high efficiency, significantly improving node classification performance, particularly in semi-supervised learning settings. The model's robustness to noisy graphs and its ability to propagate information over distant nodes makes it a powerful tool for graph-based recommendation systems.

D GU Method Details

D.1 GU Method Details

GraphEraser [11] is a GU method designed to efficiently remove specific nodes or edges from a trained GNN model. By utilizing two novel graph partition algorithms and a learning-based

aggregation method, GraphEraser identifies and updates only the relevant substructures, enabling precise removal while minimizing computational overhead.

GUIDE [67] is an inductive GU framework designed to address the limitations of traditional transductive unlearning approaches like GraphEraser. GUIDE incorporates a balanced graph partitioning mechanism, efficient subgraph repair, and similarity-based aggregation to ensure effective unlearning while maintaining computational efficiency. This approach enables unlearning with low partition costs, preserving model adaptability in inductive graph learning tasks.

GraphRevoker [87] addresses the challenge of GU by balancing efficient unlearning with high model utility. Unlike traditional methods that may fragment essential information during partitioning, GraphRevoker employs property-aware sharding, preserving key structural and attribute information. For final predictions, it integrates sub-GNN models through a contrastive aggregation technique, ensuring coherent model utility while unlearning data.

GIF [73] innovatively tackles GU by leveraging a tailored influence function, enhancing both efficiency and precision in unlearning processes. GIF redefines influence to address dependencies among neighboring nodes by introducing a specialized loss term that accounts for structural interactions, which traditional influence functions overlook. This model-agnostic approach enables GIF to estimate parameter adjustments in response to small data perturbations, providing a closed-form solution that facilitates understanding of the unlearning mechanics.

ScaleGUN [84] is a scalable framework for certified GU, addressing the inefficiency of traditional methods by integrating approximate graph propagation techniques. It ensures certified guarantees for node feature, edge, and node unlearning scenarios while maintaining bounded model error on embeddings. By balancing efficiency and accuracy, ScaleGUN extends certified unlearning to large-scale graphs without compromising privacy guarantees.

CGU [15] offers the first approximate GU approach with theoretical guarantees, designed to manage varied unlearning requests. CGU emphasizes precise handling of feature mixing during propagation, particularly within SGC. This enables CGU to efficiently balance privacy, complexity, and performance, achieving rapid unlearning with minimal accuracy loss.

CEU [74] introduces an efficient solution for edge unlearning in GNNs by bypassing the high costs of full retraining. CEU uniquely leverages a single-step parameter update on the pre-trained model, effectively erasing edge influence while preserving model integrity. The CEU framework also offers theoretical guarantees under convex loss conditions, achieving notable speedup with model accuracy nearly on par with complete retraining.

GST [49] offers an innovative approach to GU by leveraging its efficient, stable, and provably resilient framework under feature or topology perturbations. Unlike traditional GNN retraining, GST-based unlearning introduces a nonlinear approximation mechanism that achieves competitive graph classification performance, making GST an advantageous method for privacy-sensitive applications demanding efficient unlearning without sacrificing performance.

IDEA [23] represents an outstanding framework in the realm of GU, addressing the critical need for flexible and certified unlearning. It pioneers an approach that accommodates a variety of unlearning requests, transcending the limitations of specialized GNN designs or objectives. IDEA's flexibility is underscored by its ability to provide theoretical guarantees for information removal across diverse GNN architectures, setting a new standard for privacy protection.

GNNDelete [13] addresses key challenges in GU by introducing a model-agnostic, layer-wise operator that effectively removes graph elements. Its core mechanisms, Deleted Edge Consistency and Neighborhood Influence, ensure that deleted edges and nodes are fully excluded from model representations without compromising the influence of neighboring nodes.

MEGU [41] introduces a pioneering mutual evolution approach for GU, where prediction accuracy and unlearning effectiveness evolve synergistically within a single training framework. MEGU's adaptability enables precise unlearning at the feature, node, and edge levels, showing strong superiority in the field of GU.

SGU [40] is a pioneering approach that addresses the challenges of gradient-driven node entanglement and scalability in GU. By integrating Node Influence Maximization, SGU identifies the highly

influenced nodes through a decoupled influence propagation model, offering a scalable and plug-and-play solution.

D2DGN [58] revolutionizes GU through knowledge distillation. It adeptly addresses the complexities of removing specific elements from GNNs by dividing and marking graph knowledge for retention and deletion. D2DGN stands out for its efficiency, effectively eliminating the influence of deleted elements while preserving retained knowledge, and its superior performance in both edge and node unlearning tasks across real-world datasets.

GUKD [90] harnesses the power of knowledge distillation, emerging as an innovative solution for class unlearning in GNN. It distinctively pairs a deep GNN model with a shallow student network, facilitating the transfer of retained knowledge and enabling targeted forgetting. GUKD stands out with its exceptional performance, thereby setting a new benchmark for efficient and effective GU.

UtU [62] represents a paradigm shift in edge unlearning for GNNs, offering a streamlined solution that eschews the pitfalls of over-forgetting associated with traditional methods. This innovative technique simplifies the unlearning process by directly unlinking specified edges from the graph. UtU's elegance lies in its ability to safeguard privacy with minimal computational overhead, aligning closely with the performance of a freshly retrained model while ensuring the integrity of the remaining graph structure.

Projector [18] stands out by projecting the pre-trained model's weight parameters onto a subspace unrelated to the features of unlearning nodes, effectively bypassing node dependency issues. This method ensures a perfect data removal, where the unlearned model parameters are devoid of any information concerning the unlearned nodes, a guarantee inherent in its algorithmic design.

E Implementation of OpenGU Codebase

E.1 Codebase Example

In this section, we introduce the extensibility and integration capabilities of OpenGU from three key perspectives: modular architecture, method-specific pipelines, and unified foundational components. Through representative code examples, we demonstrate how OpenGU facilitates seamless integration and development of diverse GU methods, ensuring flexibility for GU research.

Algorithm 1 OpenGU-*GIF-Example*.*Pytorch style*.

```
import torch
from OpenGU.unlearning_manager import GU_Manager

# Device configuration (CUDA acceleration)
device = "cuda" if torch.cuda.is_available() else "cpu"

# Initialize Graph Unlearning Manager with GIF method
GIF = GU_Manager(unlearn_method="gif",           # Unlearning Algorithms
                 dataset_name="cora",            # Graph Dataset
                 num_runs=10,                    # Experiment Repetitions
                 unlearn_ratio=0.1,              # Data Removal Ratio
                 unlearn_task="node",            # Unlearning Requests
                 downstream_task="node",         # Downstream Tasks
                 base_model="GCN",               # Backbone GNN Architecture
                 device=device)                  # Computation Device

# Execute graph unlearning experiment
GIF.run_exp()
```

Modular Framework in OpenGU.

The OpenGU framework is designed with a unified execution pipeline to ensure seamless integration of diverse GU methods while maintaining extensibility. Through object-oriented programming, we encapsulate algorithmic workflows into the GU_Manager class, enabling users to instantiate any supported GU method (e.g., GIF, GraphEraser) by specifying core parameters: unlearn_method (algorithm selection), dataset_name (graph data), unlearn_task (node/edge/graph-level unlearning), and downstream_task (evaluation objective). This abstraction layer isolates method-specific implementations from experimental protocols, allowing researchers to focus on algorithmic comparisons without redundant code modifications. We further modularize common components across GU methods—including training initialization, unlearning operators, and verification steps—into reusable

functions, while preserving method-specific configurations as tunable hyperparameters. Such design ensures that users can reproduce baseline methods with minimal code (as shown in Algorithm 1) or explore advanced settings via parameter adjustments. The framework also enforces standardized input or output formats across different GU algorithms, ensuring consistent evaluation metrics and reproducibility. By balancing accessibility for rapid experimentation with flexibility for methodological customization, OpenGU significantly lowers the barrier to conducting rigorous GU research.

Algorithm 2 OpenGU-*Pipeline for learning-based GU methods.* *Pytorch style*.

```
class Learning_based_pipeline:
    """
    Base class for implementing a learning-based pipeline.
    """
    def __init__(self, args, logger, model_zoo):
         # Implementation omitted for brevity

    def run_exp(self):
        """Executes the experimental pipeline, encompassing training, unlearning, and evaluation."""
        for self.run in range(self.args["num_runs"]):
            self.exp_train_model()
            self.exp_unlearn()
            self.exp_evaluate()

    def determine_target_model(self):
        """Determines the specific model architecture to be used in the current experiment."""
        pass

    def train_original_model(self):
        """Defines the training process for the initial model on the complete graph data."""
        pass

    def unlearning_request(self):
        """Generates or defines the data removal request and performs any necessary preprocessing related
     to the unlearning method."""
        pass

    def learning_for_unlearn(self):
        """Implements the core learning mechanism for unlearning, such as training an unlearning model or
     fine-tuning the original model."""
        pass

    def exp_train_model(self):
        """Executes the steps to determine the target model and train the initial model."""
        self.determine_target_model()
        self.train_original_model(self.run)

    def exp_unlearn(self):
        """Executes the unlearning process, including handling the unlearning request and performing the
     learning for unlearning."""
        self.unlearning_request()
        self.learning_for_unlearn()

    def exp_evaluate(self):
        """Evaluates the performance of the unlearned model."""
        pass
```

Extensible Pipeline in OpenGU.

To illustrate the extensibility of the OpenGU framework for GU methods, we have integrated a variety of pipelines tailored to different approaches. Here, we use the core abstract class **Learning_based_pipeline** as an example to demonstrate the framework's capabilities (See Algorithm 2). This pipeline encapsulates the general workflow of learning-based GU methods and provides reserved interfaces for key steps, significantly simplifying the integration and development of new approaches. While most methods are defined as abstract, requiring implementation in subclasses, the base class delineates the primary steps necessary for the unlearning process. Specifically, the `run _exp()` method orchestrates the core workflow, comprising three critical stages: model training (`exp_train_model()`), unlearning (`exp_unlearn()`), and evaluation (`exp_evaluate()`). This modular design ensures that most learning-based GU methods can be seamlessly integrated by implementing these steps. Developers can implement custom logic for each step while adhering to the standardized pipeline, significantly reducing the complexity of integrating novel GU methods.

Unified Foundation in OpenGU.

To streamline the integration of various methods into OpenGU, we designed the BaseTrainer class as a foundation for completing essential GNN tasks, as shown in Algorithm 3. This class provides a unified framework for node, edge, and graph-level tasks, facilitating seamless integration of diverse

unlearning requests and downstream tasks. Key methods, such as train and evaluate, dynamically adapt to the specified task, ensuring modularity and ease of use. By inheriting BaseTrainer, developers can rapidly implement custom GU methods, focusing on innovations like new loss functions or data manipulations while leveraging a robust training pipeline. The code snippet highlights this flexible structure, omitting implementation details to emphasize its reusable design.

Algorithm 3 OpenGU-*BaseTrainer for GU methods.* *Pytorch style*.

```
class BaseTrainer:
    """ A base trainer class for training models on various tasks (node, edge, or graph level). """

    def __init__(self,args,logger,model, data):
        """ Initializes the BaseTrainer with the provided configuration, logger, model, and data. """
        # Implementation omitted for brevity

    def train(self,save=False,model_path=None):
        """ Trains the model based on the specified downstream task (node, edge, or graph). """
        if self.args["downstream_task"] == 'node':
            return self.train_node(save,model_path)
        elif self.args["downstream_task"] == 'edge':
            return self.train_edge(save,model_path)
        elif self.args["downstream_task"]=="graph":
            self.train_loader = DataLoader(self.data[0], batch_size=64, shuffle=False)
            self.test_loader = DataLoader(self.data[1], batch_size=64, shuffle=False)
            return self.train_graph(save,model_path)

    def evaluate(self):
        """
        Evaluates the model based on the specified downstream task (node, edge, or graph).
        """
        if self.args["downstream_task"] == 'node':
            return self.evaluate_node_model()
        elif self.args["downstream_task"] == 'edge':
            return self.evaluate_edge_model()
        elif self.args["downstream_task"]=="graph":
            return self.evaluate_graph_model()
    def train_node(self,save=False,model_path=None):
        """
        Trains the model for node-level tasks.
        Returns:
            tuple: A tuple containing the best F1 score achieved during training and
                   the average training time per epoch.
        """
        if self.args["use_batch"]:
            return self.train_node_minibatch(save,model_path)
        else:
            return self.train_node_fullbatch(save,model_path)

    def train_graph(self, save=False, model_path=None):
        """
        Trains the model for graph-level tasks.
        """
        # Implementation omitted for brevity

    def train_edge(self, save=False, model_path=None):
        """
        Trains the model for edge-level tasks.
        """
        # Implementation omitted for brevity
```

E.2 Completeness and Modularity

A primary contribution of OpenGU lies in its enhancement of graph unlearning codebases, achieving both completeness and modularity to address the diverse needs of GU applications. In terms of completeness, OpenGU provides a comprehensive framework by integrating 16 SOTA GU algorithms and systematically supporting three downstream tasks—node classification, link prediction, and graph classification—alongside three unlearning request types: node unlearning, edge unlearning, and feature unlearning. For concise notation, we denote these unlearning request types as N-U, E-U, and F-U, respectively.This exhaustive coverage ensures that OpenGU captures a wide range of GU scenarios, extending beyond the limited task-specific implementations of original source codes. Complementing this, its modularity allows seamless recombination of these tasks and requests, enabling flexible adaptation to varied unlearning demands without compromising algorithmic integrity. We detail these advancements in Table 6, which evaluates the implementation status across all methods. Checkmarks (✓) indicate successful realization of a task or request, while crosses (✗) denote absence, with the left column representing the original source code and the right column reflecting OpenGU's integrated

environment. Short line (–) specifically highlight cases where inherent technical constraints of a method prevent extension, offering clarity on the boundaries of current GU approaches. This dual emphasis on completeness and modularity not only standardizes GU implementations but also enhances their adaptability, positioning OpenGU as a robust and versatile platform for advancing research and practical deployment in graph unlearning.

Table 6: Code Completion Granularity of OpenGU Across GU Algorithms

Combinations	Node Classification			Link Prediction			Graph Classification		
	N-U	E-U	F-U	N-U	E-U	F-U	N-U	E-U	F-U
GraphEraser	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	– | ✓	– | ✓	– | ✓
GraphRevoker	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	– | –	– | –	– | –
GUIDE	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	– | –	– | –	– | –
GIF	✓ | ✓	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓
CGU	✓ | ✓	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✗	✗ | ✗	✗ | ✗
ScaleGUN	✓ | ✓	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✗	✗ | ✗	✗ | ✗
IDEA	✓ | ✓	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓
CEU	✗ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✗	✗ | ✗	✗ | ✗
GST	– | –	– | –	– | –	– | –	– | –	– | –	✓ | ✓	✓ | ✓	✓ | ✓
GNNDelete	✗ | ✓	✗ | ✓	✗ | ✓	✓ | ✓	✓ | ✓	✓ | ✓	– | –	– | –	– | –
MEGU	✓ | ✓	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	– | –	– | –	– | –
SGU	✓ | ✓	✓ | ✓	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	– | –	– | –	– | –
D2DGN	✗ | ✓	✗ | ✓	✗ | ✓	✓ | ✓	✓ | ✓	✗ | ✓	– | –	– | –	– | –
GUKD	✓ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	✗ | ✓	– | –	– | –	– | –
Projector	✓ | ✓	✗ | ✓	– | –	✗ | ✓	✗ | ✓	– | –	– | –	– | –	– | –
UtU	✗ | ✓	✗ | ✓	– | –	✗ | ✓	✓ | ✓	– | –	– | –	– | –	– | –

F Evaluation Metric Details

F.1 Metrics For Predictive Performance

Our OpenGU includes three downstream tasks in total: node classification, link prediction, and graph classification. These downstream tasks are evaluated using F1-score, AUC-ROC, and accuracy, respectively. The choice of evaluation metrics for each task is carefully made based on the unique characteristics and performance evaluation requirements of the corresponding downstream task.

F1-score, a widely-used metric for classification tasks, computes the harmonic mean between precision and recall. This metric is commonly applied to node classification tasks, where class imbalance is frequently observed in the form of considerable differences in instance counts among categories. In such cases, accuracy alone might not fully reflect the effectiveness of the model. F1-score addresses this limitation by integrating both precision, which quantifies the proportion of positive predictions that are correct, and recall, which measures the proportion of actual positive instances that are detected, thereby providing an effective evaluation of performance in scenarios characterized by imbalanced node classes. Formally, F1-score is computed as follows:

$$\text{F1-score} = 2\,\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where precision is defined as the ratio of correctly predicted positive instances, denoted as TP, to the total number of predicted positive instances, which equals the sum of true positives and false positives, denoted as FN:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

and recall is defined as the ratio of correctly predicted positive instances to all actual positive instances, that is, the sum of true positives and false negatives, denoted as FN:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

AUC-ROC, referring to the Area Under the Receiver Operating Characteristic Curve, evaluates a model's ability to distinguish positive instances from negative ones effectively, independent of classification thresholds. Specifically, in the link prediction task, the main objective is to assess the ranking capability of the model, predicting whether node pairs form links accurately. A higher AUC-ROC value reflects stronger discriminative ability between connected and unconnected node pairs, which makes it a widely adopted metric for binary relational data, especially in scenarios involving class imbalance. Formally, AUC-ROC can be interpreted as the probability that a randomly chosen positive instance receives a higher predicted score than a randomly chosen negative instance:

$$\text{AUC} = \Pr\left(s(x^+) > s(x^-)\right)$$

where $s(\cdot)$ denotes the predicted score produced by the model, x^+ represents a connected node pair, and x^- denotes a node pair without an observed link. This formulation reflects the core objective of link prediction, which is to assign higher scores to truly connected node pairs than to unconnected ones across all possible candidates.

Accuracy, a straightforward yet informative evaluation metric, quantifies the proportion of correct predictions over the total number of predictions. It is commonly adopted as the primary performance indicator in graph classification tasks due to its conceptual simplicity and interpretability. Unlike node-level or link-level prediction, graph classification involves assigning a categorical label to an entire graph instance. Therefore, accuracy offers a holistic measure of a model's overall classification capability at the graph level. Formally, accuracy is defined as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

where TP and TN represent the number of correctly predicted positive and negative instances, respectively, while FP and FN denote the number of incorrectly predicted positive and negative instances. This metric captures the overall correctness of the model's predictions across all classes and is considered reliable under conditions of approximately balanced class distributions.

Precision, as previously defined in the context of the F1-score, is a standard metric used to assess the correctness of positive predictions, indicating the extent to which predicted positive instances are truly relevant. It reflects the model's capacity to reduce false alarms and is particularly relevant in contexts where false positives carry substantial cost, such as in fraud detection or medical screening. By focusing on the proportion of predicted positives that are actually correct, precision serves as an important complement to recall in evaluating model behavior.

F.2 Metrics For Unlearning Performance

To assess whether the target information has been effectively forgotten, we adopt two representative evaluation strategies tailored to the characteristics of node-level and edge-level tasks. Specifically, we implement Membership Inference Attack and Poisoning Attack to evaluate unlearning performance from both instance-level and structural perspectives.

Membership Inference Attack is a privacy attack technique targeting machine learning models, in which an adversary attempts to infer whether a particular data instance was part of the model's original training dataset [19, 48, 89]. Given simultaneous access to both the original and the unlearned models, the attacker can further determine whether the target instance has been effectively removed from the training set underlying the unlearned model. Due to its capability of quantifying the presence

or absence of specific training instances, MIA serves as a rigorous evaluation protocol for measuring the effectiveness of unlearning methods.

In our experiments, we directly apply MIA to the graph neural network following the forgetting procedure, adhering to the evaluation framework introduced by GraphEraser [11]. Specifically, MIA is employed in node-node experiments to quantitatively assess the extent of information removal by evaluating the adversary's ability to distinguish forgotten nodes from nodes never included in the training process. The performance of the membership inference attack is quantified by the AUC-ROC metric. An AUC-ROC score approaching 0.5 indicates that the attacker cannot reliably differentiate forgotten nodes from nodes not observed during training, signifying effective elimination of node-specific memorization from the unlearned model.

Poisoning Attack is a class of adversarial manipulation designed to degrade model performance by injecting deliberately constructed edges into the training graph [94, 95]. These adversarial edges interfere with the message-passing mechanism of graph neural networks (GNNs), introducing corrupted relational dependencies that negatively affect downstream tasks such as link prediction. In our edge-edge experiments, such poisoned edges are injected during training and subsequently designated as forgetting targets.

During the unlearning phase, the model is retrained to remove the influence of these injected edges. The effectiveness of unlearning is evaluated by monitoring the change in link prediction performance on the poisoned subset before and after unlearning. An increase in the AUC-ROC score following unlearning indicates that the negative influence of the poisoned edges has been successfully eliminated. This improvement serves as an indirect but informative measure of the model's capability to forget targeted edge-level information while preserving performance on non-poisoned regions of the graph.

G Experimental Details

G.1 Experimental Environment Details

All experiments are conducted on a system equipped with an NVIDIA A100 80GB PCIe GPU and an Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz, with CUDA Version 12.4 enabled. The software environment is set up with Python 3.8.0 and PyTorch 2.2.0 to ensure optimal compatibility and performance for all GU algorithms. Additionally, hyperparameters for each GU algorithm are configured based on conclusions drawn from prior research to provide consistent and reliable results.

G.2 Standardized Experimental Protocol for Graph Unlearning

Prior studies have adopted different experimental settings, including inconsistent dataset splits, diverse GNN backbones, and varying formulations of unlearning requests, as shown in Table 7. These disparities have impeded direct and intuitive comparisons across methods, limiting the ability to assess their relative strengths and weaknesses systematically. To address this challenge, we propose a unified evaluation framework that standardizes these key components, including dataset partitioning, GNN backbone selection, and unlearning request design, and provide detailed justifications for our choices. This framework enables a fair and comprehensive comparison of existing GU approaches while accommodating a broader range of tasks and scenarios.

Standardized Dataset Partitioning
Dataset partitioning is a foundational aspect of model training and evaluation, yet prior GU methods have employed inconsistent split ratios, complicating direct comparisons. For instance, GraphEraser, a seminal work in graph unlearning, adopts an 80%/20% train-test split, whereas GIF, a representative IF-based method, opts for a 90%/10% division. Such variations in data allocation fundamentally affect model performance and unlearning outcomes, as the size of the training set influences both the initial learning capacity and the subsequent unlearning burden. ***To establish a consistent baseline, we adopt an 80%/20% split as the standard for our experiments***. This choice aligns with the majority of foundational GU studies, such as GraphEraser, and strikes a balance between providing sufficient training data and retaining a meaningful test set for evaluation, thereby facilitating reproducible and comparable results across methods.

Standardized GNN Backbones
The choice of GNN backbone is another critical factor influencing GU performance, as different

Table 7: A Summary of Previous GU Settings

Algorithm	Dataset Partition	Downstream Task	Unlearning Request	Task Description
GraphEraser	80% train, 20% test	Node classification	Node/Edge unlearning	Unlearn 100 nodes/edges, Unlearn at increasing proportion
GraphRevoker	70% train, 20% val, 10% test	Node classification	Node/Edge unlearning	Unlearn 0.5% nodes
GUIDE	80% train, 20% test	Node classification / Bitcoin illegal transaction detection	Node/Edge unlearning	Unlearn 100 nodes/edges, Unlearn by increasing number
GIF	90% train, 10% test	Node classification	Node/Edge/Feature unlearning	Unlearn 5% edges
CGU	Variable Proportion Split	Node classification	Node/Edge/Feature unlearning	Unlearn by increasing number
ScaleGUN	Variable Proportion Split	Node classification	Node/Edge/Feature unlearning	Unlearn by increasing number
IDEA	90% train, 10% test	Node classification	Node/Edge/Feature unlearning	Unlearn 5% nodes/edgess
CEU	70% train, 20% val, 10% test	Node classification	Edge unlearning	Unlearn by increasing number
GST	80% train, 10% val, 10% test	Graph classification	Node unlearning	Unlearn one node from 10% training graph
GNNDelete	90% train, 5% val, 5% test	Node classification	Node/Edge/Feature unlearning	Unlearn 2.5% edges, Unlearn 100 nodes, Unlearn by increasing proportion
MEGU	80% train, 20% test	Node classification	Node/Edge/Feature unlearning	Unlearn 10% nodes
SGU	80% train, 20% test	Node classification / Link prediction	Node/Edge/Feature unlearning	Unlearn 10% nodes
D2DGN	No description	Link prediction	Edge unlearning	Unlearn 2.5% edges
GUKD	No description	Node classification	Node unlearning	Unlearning single-class
Projector	No description	Node classification	Node unlearning	Unlearn 5% or 10% nodes
UtU	90% train, 5% val, 5% test	Link prediction	Edge unlearning	Unlearn 5% edges, Unlearn by increasing proportion

methods are often tailored to specific architectures. For example, CGU is exclusively designed around the SGC model, while learning-based approaches typically exhibit flexibility across a wider range of GNNs. To ensure broad applicability, we select three representative GNN backbones tailored to distinct task types. For node-node tasks, we employ ***SGC***, a decoupled GNN known for its simplicity and efficiency, as the backbone due to its prominence in prior GU studies and its suitability for node-level predictions. For edge-edge tasks, we adopt ***GraphSAGE***, a classic sampling-based GNN, which aligns well with link prediction task. Finally, for graph-feature tasks, we use the ***GCN***, a widely adopted and traditional architecture, to leverage its robustness in graph-level classification. This strategic selection enables a comprehensive evaluation of GU algorithms across diverse GNN paradigms, ensuring both comparability and insight into their adaptability.

Standardized Unlearning Requests

The formulation of unlearning requests varies significantly across existing studies, further complicating comparisons. For instance, GraphEraser removes a fixed number of 100 nodes per unlearning request, whereas MEGU targets 10% of the nodes in the graph. Fixed-quantity requests fail to account for differences in dataset scale, potentially skewing performance metrics and limiting generalizability. To address this, we standardize unlearning requests by adopting a proportional approach. Specifically, for node-level and edge-level unlearning, we designate ***10% of the respective elements*** (nodes or edges) as the unlearning target, reflecting a scalable and dataset-agnostic strategy that adapts to varying graph sizes. For graph-feature tasks, where prior work offers limited guidance, we propose a novel setting: we select ***half of the graphs*** in the dataset and remove ***10% of their features*** by setting

them to zero vectors. This design simulates feature-level unlearning in a simplified yet practical manner, ensuring consistency while accommodating the constraints of existing algorithms, many of which lack support for partial feature unlearning. These standardized unlearning requests provide a coherent basis for evaluating GU performance across diverse scenarios.

Standardized Experimental Tasks and Evaluation Scope
Previous GU research has predominantly focused on node unlearning for node classification tasks, leaving other downstream tasks—such as link prediction and graph classification—and their corresponding unlearning requests underexplored. This narrow focus has hindered a holistic understanding of GU capabilities across varied applications. To bridge this gap, we define three core experimental tasks that integrate distinct downstream objectives with corresponding unlearning requests.
(1) *Node-Node Task*: As a benchmark in the GU domain, the node-node task pairs node classification with node unlearning requests. Given its foundational role in prior studies, we establish this task as a cornerstone of our framework, applying our standardized dataset split (80%/20%), SGC backbone, and 10% node removal. This unified setup facilitates a rigorous comparison of existing methods and provides a reference point for further advancements.
(2) *Edge-Edge Task*: The edge-edge task combines link prediction with edge-level unlearning, addressing a critical yet underexamined scenario. We enhance prior work by supplementing code implementations and results, utilizing GraphSAGE as the backbone and removing 10% of edges as the unlearning request. This task highlights the adaptability of GU algorithms to real-world settings, such as social network link management, and enriches the evaluation landscape.
(3) *Graph-Feature Task*: The graph-feature task integrates graph classification with feature unlearning, a novel combination designed to probe GU performance in feature-centric applications. Using GCN as the backbone, we remove 10% of features from half of the graphs, simplifying the unlearning request to full feature deletion (zero vectors) to accommodate the limitations of existing methods. This task underscores the practical relevance of feature unlearning, such as in privacy-sensitive graph data, and establishes a baseline for future exploration.

While we emphasize these three task combinations for detailed analysis, our framework is designed with extensibility in mind. We have implemented code for all possible combinations of downstream tasks (node classification, link prediction, graph classification) and unlearning requests (node, edge, feature), ensuring that our methodology can support broader investigations. This comprehensive approach not only standardizes current evaluations but also lays the groundwork for future GU research across diverse contexts.

H Reasoning Performance Experiment

H.1 Reasoning Performance Experiment Details for Node-Node

Experiment Settings

For the prominent Node-Node task in graph unlearning, we evaluate 14 GU algorithms using 9 datasets drawn from diverse domains, with data scales varying from small to large. Each dataset is partitioned into an 80% training set and a 20% test set in transductive setting, with 10% of the training nodes designated for removal to establish a standardized Node-Node unlearning request. We select the decoupled **SGC** as the GNN backbone, given its broad compatibility with most GU algorithms, ensuring consistent application across experiments. Performance is measured using the F1-score, a widely adopted metric in graph unlearning, with results derived from 10 independent runs per dataset and expressed as the mean ± standard deviation to provide a reliable statistical summary. UtU is absent from the table because its original methodology relies entirely on edge information. Although node unlearning can be reformulated as edge unlearning, this transformation fundamentally undermines the method's intended purpose. Similarly, while we adapt GST's source code—originally designed for graph classification—to address node classification, its effectiveness for node unlearning tasks was markedly inferior. This deficiency become especially pronounced with larger datasets, ultimately rendering GST's overall performance non-competitive when compared with other established methods and thus justifying its exclusion from the final comparative table.

General hyperparameter Settings

For the SGC backbone, which serves as a universal component across all GU algorithms, we configure a 2-layer architecture with a hidden layer dimension of 64. During the training process, we set the

Table 8: Hyperparameter Settings and Search Space for GU Algorithms

Algorithm	Hyperparameter	Search Space
GraphEraser	Shard number k	{20, 30, 50, 100}
	Maximum node number in a shard δ	$\lceil n/k \rceil$
	Node number for aggregation	{10%, 1000}
	Maximum iterations T	{30}
GraphRevoker	Shard number k	{20}
	Learning rate for partition function Ψ	$\{1e-3\}$
	Epoch for partition function Ψ	[10, 30]
	Learning rate for aggregation	$\{1e-4\}$
	Node number for aggregation	{10%, 1000}
GUIDE	Shard number k	{20}
GIF	Scaling coefficient λ	$\{10^1, 10^2, 10^3, 10^4\}$
CGU	Differential privacy constraint (ϵ, δ)	{(1, 1e-4)}
	Noise standard deviation α	{0.1}
	Regularization coefficient λ	{0.01}
ScaleGUN	Differential privacy constraint (ϵ, δ)	{(1, 1e-4)}
	Threshold r_{max} fo approximation error	{1e-7, 1e-8, 1e-9, 5e-10, 1e-10, 1e-15}
IDEA	Standard deviation of Gaussian noise σ	[0.001, 1]
	Strong convexity parameter λ	{0.05}
	Lipschitz constant of loss function L	{0.25}
	Numerical bound of training loss per node C	{3.0}
	Lipschitz constant of Hessian matrix L_H	{0.25}
	Upper bound of loss derivative G	{1.0}
	Lipschitz constant of the first-order derivative L_g	{1.0}
CEU	Differential privacy constraint (ϵ, δ)	$\epsilon = [0.1, 10]$, $\delta = 1e-4$
	Regularization coefficient λ	{0.01}
	Gaussian distribution $(0, \sigma^2)$	$\sigma = 0.1$
GST	Differential privacy constraint (ϵ, δ)	{(1, 1e-4)}
	Noise standard deviation α	{0.1}
	Number of scales for graph wavelets J	{3, 4, 5}
	Number of moments computed Q	{3, 4}
	Number of layers in the scattering tree L	{2, 3}
GNNDelete	Regularization parameter in loss function λ	[0, 1]
MEGU	Loss function coefficient k	[0.01, 0.1]
	Personalized pageRank coefficient α_1, α_2	[0, 1], [0.05, 0.5]
SGU	Unlearning budget	[0.1, 0.3]
	Activation threshold θ	[0.5, 1]
	Loss function coefficient λ	[0, 1]
D2DGN	Regularization parameter in loss function α	{0.5}
GUKD	Epoch for student model	{25}
	Learning rate for student model	{0.01}
Projector	Learning rate for SGD model	{0.1, 1}
	Momentum	{0.9}
	Adaptive aggregation step size γ	{1}
	Regularization coefficient	{1e-6}
UtU	No specific hyperparameter	

learning rate to 0.02 and the weight decay to $1e-6$, conducting training over 100 epochs. The model yielding the optimal performance was selected as the backbone for subsequent GU methods.

Specific hyperparameter

For the hyperparameters in each Node-Node task, we carefully examine the settings detailed in the original research papers. Where hyperparameters are explicitly fixed by the authors, we adopt the optimal configurations from these prior methods, ensuring consistency with established benchmarks. For those amenable to variation, we establish comprehensive search spaces to systematically determine the most effective values. The resulting configurations are detailed in Table 8.

Table 9: AUC-ROC ± STD comparison(%) for inductive link prediction task with edge unlearning.

Edge-Level	Cora	CiteSeer	PubMed	DBLP	Physics	Questions
Retrain	90.48±0.56	89.48±0.67	93.59±0.59	94.14±0.28	92.82±0.31	91.87±0.09
GraphEraser	67.19±1.59	67.89±2.60	78.53±0.28	74.54±0.32	78.54±0.69	70.16±0.73
GUIDE	64.11±1.19	65.31±1.99	64.67±0.28	65.28±0.36	50.00±0.08	39.32±0.43
GraphRevoker	76.39±1.18	76.00±1.60	87.73±0.39	81.27±0.61	OOM	OOM
GIF	**84.58±1.36**	**85.91±1.21**	88.48±0.95	89.91±1.49	88.43±2.28	85.71±0.21
ScaleGUN	78.78±0.12	74.62±0.31	77.89±0.02	73.21±0.04	**91.27±0.04**	71.58±0.09
IDEA	84.34±1.41	85.41±0.45	**88.63±1.07**	**92.64±0.10**	83.96±0.15	85.75±0.20
GNNDelete	84.28±1.38	83.47±2.20	86.33±0.79	90.62±0.95	83.47±0.44	83.52±1.09
MEGU	62.42±0.86	69.83±0.35	57.79±0.77	51.60±0.23	51.21±0.14	**85.79±0.25**
SGU	78.85±1.08	78.85±0.96	77.02±0.23	74.55±0.38	72.66±0.13	66.68±0.11
D2DGN	80.65±0.82	82.64±0.92	78.90±2.57	83.92±0.83	82.82±0.43	75.53±0.34
GUKD	56.43±1.21	62.21±7.09	62.21±7.09	78.49±1.91	76.52±4.79	78.83±0.98
UtU	80.76±1.72	82.60±1.29	86.99±0.88	85.95±0.18	72.62±0.13	79.50±0.67

H.2 Reasoning Performance Experiment Details for Edge-Edge

Review

Upon reviewing previous work, it is found that apart from methods specifically designed for edge unlearning requests, such as GNNDelete and UtU approaches, few existing GU methods have been evaluated on link prediction as a downstream task. This evident limitation in prior research prevents a fair assessment of whether these methods are suitable for link prediction tasks and edge unlearning requests. To address this gap, OpenGU significantly extends the existing collection of GU methods by thoughtfully adapting them for edge-edge unlearning requests, thereby offering a unified and user-friendly interface that facilitates the seamless implementation and evaluation of comprehensive edge-edge unlearning experiments across various graph datasets.

Experiment Settings

Through seamless implementation provided by OpenGU, we conduct experiments on 13 GU methods across 6 datasets on Edge-Edge Task. These datasets are partitioned with a fixed ratio, allocating 80% for training and 20% for testing in inductive setting, with 10% of the edges from the training set designated as the unlearning request. We select **GraphSAGE**, a representative GNN of the sampling paradigm, as the backbone for these experiments due to its effectiveness in edge-centric tasks. Performance is assessed using the AUC-ROC metric, the standard measure in link prediction, with final results obtained from 10 independent runs to ensure statistical reliability. CEU and CGU are not compared with other algorithms because they are not suitable for GraphSAGE. Projector is excluded from the table data due to its limited adaptability to the Edge-Edge task, which precludes effective integration into this experimental setup. Similarly, GST's exclusion stems from problems analogous to those encountered in the Node-Node task.

General hyperparameter Settings

For the GraphSAGE backbone, which serves as a foundational component for subsequent experiments, we configure a 2-layer GNN architecture with a hidden layer dimension of 64. During training phase, we set the initial learning rate to a value of 0.005 and the weight decay to 1×10^{-6}, performing training over 100 epochs to ensure sufficient convergence. The model parameters yielding the best performance are fixed as the baseline for all subsequent experimental evaluations.

Specific hyperparameter

For specific hyperparameters employed in our experimental setup, our configurations closely align with those used in the Node-Node task, as detailed in Table 8.

Table 10: ACC ± STD comparison(%) for graph classification task with feature unlearning.

Method	MUTAG	PTC-MR	BZR	COX2	DHRF	AIDS	NCI1
Retrain	78.42±3.07	66.67±1.83	78.52±1.67	92.13±0.52	90.53±0.89	98.95±0.19	59.88±0.74
GraphEraser	65.79±0.05	53.04±0.34	73.33±0.19	**90.21±0.76**	85.26±2.68	93.30±0.10	OOM
GST	68.42±0.13	55.07±0.76	74.07±0.51	88.30±0.46	**86.18±3.14**	90.90±0.20	**59.08±0.20**
GIF	73.68±1.03	60.29±2.17	**75.31±3.02**	88.51±1.24	74.47±5.32	98.40±0.12	52.90±1.40
IDEA	**73.68±0.03**	**60.58±2.13**	73.33±4.18	87.87±1.59	67.24±8.28	**98.50±0.32**	53.63±1.11

H.3 Reasoning Performance Experiment Details for Graph-Feature

Review

In the current landscape of GU research and OpenGU's integration, GST is the only method proposed to address the GU challenge in graph classification task, highlighting a significant area for further exploration. Given the limited availability of methods tailored to such problems, OpenGU extends existing frameworks and GU strategies to enable graph unlearning in graph classification task, implementing four GU methods. Among these, Partition-based approaches are adapted by extending GraphEraser to the graph classification context, where K-means clustering is used to partition graphs for unlearning. GIF and IDEA, on the other hand, focus on leveraging influence functions to update parameters at the graph level. However, learning-based methods are not adapted as their unlearning strategies are not suitable for graph classification tasks (e.g. lack of node class label).

Experiment Settings

By completing existing and suitable algorithms, we evaluate the performance of these 4 prominent GU methods on 7 graph datasets. For the graph classification task, we consistently partitioned each dataset into an 80% training set and a 20% test set to ensure robust evaluation. Regarding the unlearning request, we randomly selected 50% of the training set and set the features of 10% of its nodes to zero vectors. We adopted the widely utilized **GCN** as the GNN backbone due to its established efficacy in graph-based tasks. Prediction accuracy is evaluated using the accuracy metric (Acc), with all reported results averaged over 10 independent runs to enhance statistical reliability. We utilized only seven datasets because current GU algorithms predominantly lack support for large-scale graph classification datasets, encountering issues such as out-of-memory and out-of-time constraints.

General hyperparameter Settings

For the GCN backbone, employed as the core framework for subsequent experiments, we established a standard 2-layer GNN structure with a hidden layer size of 64. Throughout the entire training process, we assign an initial learning rate of 0.005 and a weight decay of 1×10^{-6}, conducting the training procedure across 100 epochs. The configuration delivering optimal performance is retained as the standard reference for all ensuing experimental assessments.

Specific hyperparameter

For specific hyperparameters employed in our experimental setup, our configurations closely align with those used in the Node-Node task, as detailed in Table 8.

I Forgetting Performance Experiment

I.1 Forgetting Performance Experiment for Node-Node

In the Node-Node experiments, the comprehensive evaluation of predictive performance on retained data and the unlearning capability for removed nodes are concurrent and inherently interrelated. Consequently, the experimental setup in this section is identical to that detailed in Appendix H.1. For the Node-Node task, we adopted the MIA as the primary evaluation metric, with a thorough and detailed explanation of MIA provided in Appendix F.2.

I.2 Forgetting Performance Experiment for Edge-Edge

In the Edge-Edge unlearning capability experiments, our setup differs only in one aspect from that described in Appendix H.2: to assess the unlearning effectiveness of GU algorithms, we employ the highly suitable Poisoning Attack. Specifically, we introduce heterophilous edges between nodes of different classes, adding a quantity equivalent to 10% of the total edge count. Intuitively, GNN training on poisoned data yields degraded performance compared to clean data, and we designate these added edges as the edge-level unlearning request. Following the application of GU algorithms, a greater improvement in predictive performance after unlearning indicates more thorough removal of the poisoned data, reflecting stronger unlearning capability.

J Trade-off Experiment

In this experiment, we concentrate on the node classification task, a domain where most graph unlearning methods exhibit robust performance, and employ the **SGC** as the backbone to ensure uniformity across all comparative analyses. The experimental setup adheres to the configurations outlined in Appendix H.1, providing a consistent foundation for our evaluations. To offer a thorough and precise assessment of the advancements achieved by these GU methods, we report their performance on three widely recognized datasets (Cora, CiteSeer, and PubMed), each selected for its distinct characteristics in node classification benchmarks. Furthermore, to capture a broader perspective, we compute the average performance of each algorithm across 9 datasets, aligning with the scope defined in Q1. Methods absent from the figure denote instances where their AUC scores surpass the upper boundary of the visualization, indicating exceptional unlearning efficacy beyond the plotted range.

K Algorithm Complexity And Practical Efficiency Analyses

K.1 Algorithm Complexity Analyses

For clarity in the complexity analysis, we introduce a set of key notations that will be used consistently throughout the discussion. To ensure a uniform evaluation of complexity, we consider GCN as the backbone model, with the number of layers denoted by L. In terms of the dataset, n represents the total number of nodes, m refers to the number of edges, f is the feature dimension of the nodes, and d denotes the average degree of the nodes. The number of classes is denoted by c, which corresponds to the label dimensionality, while u signifies the number of unlearning requests, representing the nodes, edges or features to be deleted during the unlearning process. Since part of the methods involve sampling, we define the number of samples as n_s.

Partition-based methods, distinguished by their unique design, integrate partitioning into the preprocessing phase and aggregation into the inference phase. To avoid redundancy, we consistently categorize the partitioning component of these algorithms within preprocessing and the aggregation component within inference. Additionally, we provide a comprehensive summary of space complexity for all methods, accounting for memory requirements across model parameters, data storage, and auxiliary structures, thereby ensuring a thorough evaluation. Unless restricted to edge-level operations, we adopt node unlearning as the analytical perspective for all methods. These findings are detailed in Table 3.

For **Partition-based** methods, the parameter k represents the number of shards, and t denotes the number of iterations required by the algorithm. For the three methods considered, the fundamental approach is similar, leading to comparable time and space complexities in both the training and unlearning phases. The principal differences between these GU methods are manifested in the partitioning and aggregation. In the partitioning phase, GUIDE and GraphEraser depend on non-training algorithms, whereas GraphRevoker relies on constructing loss functions. During the aggregation phase, GUIDE utilizes the pyramid match graph kernel, with complexity $O(L_k fn)$, where L_k is the number of layers in the kernel.

For most **IF-based** methods, training time is predominantly bounded by a complexity of $O(Lfm + Lf^2n)$, reflecting the computational load of feature propagation across layers. Specifically, in the case of GST, which is designed for graph-level classification tasks, the initialization time complexity is expressed as $O(\sum_{i=0}^{N} pg_i^2)$. Here, N is the number of graphs, g_i denotes the number of nodes in

graph G_i, and p corresponds to $O(\sum_{l=0}^{L-1} J^l)$ determined by the J-ary scattering tree. Regarding the unlearning process, since these methods utilize influence function for analysis, the unlearning complexity and memory overhead are predominantly dependent on the model parameter θ. For ScaleGUN, the paper indicates that the time complexity for removing a single edge is approximately $O(L^2d)$. When considering node unlearning, the time complexity is constrained by $O(L^2d^2u)$.

For **Learning-based** methods, the training phase typically involves training the GNNs with the time complexity bounded by $O(Lfm + Lf^2n)$. In the case of GUKD, an additional step is required to train a teacher model, which may differ in layer depth from the target model. To account for this difference, we define the layer count of the teacher model as L_t. Moreover, since most Learning-based GU methods rely on sampling or similar operations to enhance efficiency, the unlearning process introduces complexity terms influenced by the number of unlearning requests u, the sampling size n_s, or hyperparameters such as B in SGU. The memory complexity of Learning-based GU methods varies depending on their specific design and operations. For example, SGU leverages node influence maximization strategies, resulting in a memory complexity of $O(Bfn_s+f^2)$, where B stands for the budget hyperparameter. GNNDelete's memory usage is primarily determined by the neighborhood of the edges involved in the unlearning process, leading to a complexity of $O(fu+d^2fu)$.

For **Projector**, the authors employed a specialized algorithm to simplify the matrix inversion process, ultimately limiting the time complexity to $O(f^2n+\max\{u^3,f^2u\})$, with the space complexity controlled at $O(f^2+fn)$. Notably, for **UtU**, since it only alters the graph's information without requiring additional training, the unlearning process incurs negligible time overhead $O(u)$, being primarily dependent on the number of deletion requests.

K.2 Practical Efficiency Analyses

Time Efficiency Experiment

In experiments assessing the practical time efficiency of GU algorithms, we adopted the SGC as the GNN backbone due to its streamlined mechanism, enabling a focused evaluation of the time overhead introduced by the GU algorithms themselves. For time efficiency experiments, we utilized the baseline Node-Node setup, with 10% of nodes designated as the unlearning request, and adhered to the detailed configurations outlined in Appendix H.1. Our analysis addresses two primary objectives: first, identifying performance variations among different GU algorithms on a single dataset; second, evaluating the scalability of individual GU algorithms across datasets of increasing size. To this end, we selected a spectrum of datasets ranging from small to large, namely Cora, PubMed, ogbn-arxiv, and ogbn-products, where ogbn-products, featuring a million-scale node count, serves as a robust benchmark for assessing time overhead under near real-world conditions. It is noteworthy that the unlearning time presented in Figure 5 refers specifically to the duration from receiving unlearning requests to obtaining the unlearned model through the application of a GU algorithm.

Memory Efficiency Experiment

In experiments assessing the practical memory efficiency of GU algorithms, we consistently utilized the SGC as the GNN backbone. For the evaluation of space overhead, the experimental setup aligns fully with that employed in the time efficiency experiments. Regarding dataset selection, we incorporated four datasets not previously used in Node-Node experiments, namely Actor, Photo, Squirrel, and CS, to broaden the scope of our analysis. Owing to visualization space constraints or method-specific limitations, such as GraphEraser's substantial memory demand exceeding 4000 MB even on the minimal-scale dataset Actor, we chose to present data from methods that exhibit strong performance and representativeness. Memory overhead is quantified as the peak memory consumption recorded throughout the experiment, determined by the maximum space required for the execution of each GU algorithm.

L Robustness Analyses Details

L.1 Unlearning Intensity Experiment

To evaluate the robustness of GU algorithms with respect to unlearning intensity, we employ multiple GNN backbones to demonstrate OpenGU's flexibility in backbone selection while elucidating the impact of different GNN backbones on GU algorithm performance. In Figure 7, we utilize four

distinct backbones: GraphSAINT, Cluster-GCN, GAT, and GCN. For each backbone, we configure a two-layer propagation architecture, setting the learning rate to 0.005, the weight decay to 1e-6, and the hidden layer dimension to 64 during training. To comprehensively assess the influence of unlearning intensity on GU algorithms, we conduct experiments at both node and edge levels, incrementally increasing the unlearning ratio from 0 to 0.5, where a ratio of 0 corresponds to the initial, unperturbed model. Given that current algorithms exhibit limited exploration of feature-level unlearning and many lack support for partial feature removal, we refrain from conducting experiments with varying feature deletion intensities. By performing experiments on Cora, ogbn-arxiv, CiteSeer, and Chameleon, we present a thorough showcase of representative results from mainstream algorithms.

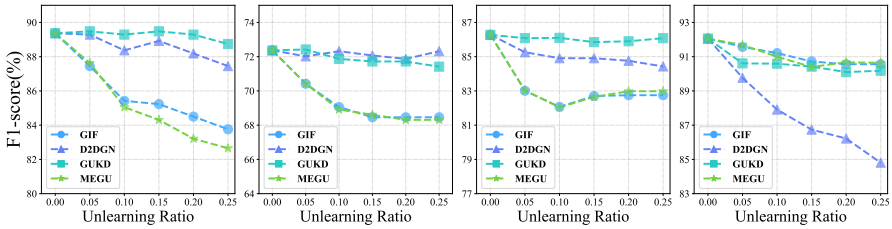

Figure 10: Performance under Incremental Unlearning Requests.

L.2 Incremental Experiment

In practical scenarios, unlearning requests typically occur repeatedly in multiple small batches rather than being executed all at once. Therefore, evaluating the robustness of unlearning algorithms under incremental forgetting is crucial. To this end, we conduct incremental experiments on several representative GU algorithms, where node-level unlearning requests are performed sequentially in batches, each removing 5% of the target nodes. We employ the GCN as the backbone and carry out extensive experiments on the Cora, CiteSeer, PubMed, and CS datasets. The corresponding results are presented in Figure 10, offering a clear depiction of model behavior under repeated forgetting operations. From the results, all methods demonstrate a consistent downward trend in accuracy as the number of unlearning iterations increases, indicating the accumulated impact of sequential unlearning. Notably, the first unlearning request causes the most significant performance degradation, suggesting that the initial perturbation exerts a stronger influence on the learned representations. As the unlearning process continues, the decline tends to stabilize, reflecting the models' adaptive capacity to incremental updates. These observations collectively underscore the importance of designing GU algorithms capable of mitigating performance decay under realistic, multi-stage unlearning scenarios.

L.3 Noise and Sparsity Experiment

To rigorously assess the robustness of contemporary GU algorithms under realistic scenarios, we devise experiments targeting two distinct conditions: noise and sparsity. For the noise condition, we opt for the Cora dataset due to its suitability for evaluating robustness against perturbations, while for the sparsity condition, we choose the CiteSeer dataset to reflect sparser graph structures. In these experiments, we treat the perturbation levels of noise and sparsity as independent variables, systematically increasing them from 0.1 to 0.8 in incremental steps to capture a broad range of effects. As the GNN backbone, we select the SSGC, configuring it with a learning rate of 0.005, a weight decay of 1×10^{-6}, and a hidden layer dimension of 64 to ensure consistency across trials. The resulting changes in prediction accuracy for representative GU algorithms under these noise and sparsity conditions appear in Figure 8, providing a clear visualization of their adaptive capabilities. Due to spatial limitations in the main text, we relegate the detailed graphical analysis of GU unlearning capabilities to the appendix, where it appears in Figure 9, offering a comprehensive view of the unlearning performance trends.

M Future Direction

GU stands as a dynamic and rapidly growing field, attracting significant interest within the research community. Numerous innovative GU methods are emerging, demonstrating a collective drive to

address the complexities of unlearning in graph scenario. Our work closely tracks these advancements, striving to contribute to the field's progress by exploring and evaluating a diverse range of GU approaches. However, due to the constraints of time and the availability of code resources, we are unable to incorporate every relevant GU method into our current study. Nevertheless, we remain committed to advancing the accessibility and inclusivity of GU research. In the future, we plan to advance the OpenGU by incorporating a broader selection of high-quality GU methods. Through this collaborative framework, we can stimulate continuous innovation and facilitate more comprehensive evaluations of graph unlearning techniques.