# Corrective Unlearning For Graph Neural Networks

A thesis submitted in partial fulfillment
of the requirements for the degree of

*(Master of Science in **Computational Natural Sciences** by Research)*

by

Varshita Kolipaka
2020113007
varshita.k@research.iiit.ac.in

Advisor: Prof. Ponnurangam Kumaraguru


International Institute of Information Technology Hyderabad
500 032, India

Jun 2025

International Institute of Information Technology Hyderabad
Hyderabad, India

# CERTIFICATE

This is to certify that work presented in this thesis proposal titled ***Corrective Unlearning For Graph Neural Networks*** by *Varshita Kolipaka* has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Advisor: Prof. Ponnurangam Kumaraguru

# Acknowledgement

I am grateful to have been part of IIIT Hyderabad and its various wonderful communities. I sincerely thank Prof. Ponnurangam Kumaraguru for welcoming me into his lab as a dual degree student in my final year of university, for believing in me, and for providing the resources and freedom to pursue meaningful research. I'm also grateful for being given opportunities to be involved in impactful initiatives and for the numerous opportunities to connect with the broader research community.

To my parents, for their incomparable and unconditional love and warmth, for being my biggest champions through times of great uncertainty and hardship, and for celebrating even the smallest of my successes. I thank my sister for her wonderfully over-the-top hypemanship.

To my friends, who have been the cornerstone of my college experience: thank you for pulling me out of my loops, for showing me a bigger world, and for being as hilarious as you are caring.

# Contents

# List of Figures

# List of Tables

# List of Related Publications

[P1] Kolipaka, Varshita, Akshit Sinha, Debangan Mishra, Sumit Kumar, Arvindh Arun, Shashwat Goel, and Ponnurangam Kumaraguru. **"A Cognac shot to forget bad memories: Corrective Unlearning in GNNs."**, to appear in International Conference on Machine Learning (ICML), 2025

[P2] Varshita Kolipaka, Akshit Sinha, Debangan Mishra, Sumit Kumar, Arvindh Arun, Shashwat Goel, Ponnurangam Kumaraguru. **"Sanity Checks For Evaluating Graph Unlearning"**, CoLLAs Workshop Track, 2024

# Abstract

Graph Neural Networks (GNNs) are increasingly being used for a variety of ML applications on graph data. Because graph data does not follow the independently and identically distributed (*i.i.d.*) assumption, adversarial manipulations or incorrect data can propagate to other data points through message passing, which deteriorates the model's performance. To allow model developers to remove the adverse effects of manipulated entities from a trained GNN, we study the recently formulated problem of *Corrective Unlearning*. We find that current graph unlearning methods fail to unlearn the effect of manipulations even when the whole manipulated set is known. We introduce a new graph unlearning method, ***Cognac***, which can unlearn the effect of the manipulation set even when only 5% of it is identified. It recovers most of the performance of a strong oracle with fully corrected training data, even beating retraining from scratch without the deletion set, and is 8x more efficient while also scaling to large datasets. We hope our work assists GNN developers in mitigating harmful effects caused by issues in real-world data, post-training.

Beyond introducing a novel method, this work advances scientific methodology in GNN unlearning. We first use adversarial evaluation for graph unlearning methods beyond privacy applications, showing that metrics must genuinely reflect unlearning efficacy in corrective settings. Our extensive baselining includes methods from other domains like image unlearning for the first time in GNN unlearning, revealing that non-graph-specific approaches can surprisingly outperform graph-specific SOTA. Furthermore, our rigorous ablations challenge prevalent assumptions in GNN unlearning literature; for example, we show that common practices like node unlinking are not universally beneficial.

*Chapter 1*

# Background

## 1.1 Graph Neural Networks

Graph Neural Networks (GNNs) represent a fundamental advance in machine learning and in broader scientific fields as they provide a principled way to learn directly from data structured as graphs [23]. Such data, formally $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with entities $\mathcal{V}$ and their relationships $\mathcal{E}$, is ubiquitous in scientific domains and complex systems. GNNs distinguish themselves by their capacity to process these arbitrary, non-Euclidean structures, moving beyond the independent and identically distributed (i.i.d.) assumptions of many traditional models. This capability has been instrumental in addressing previously intractable problems, particularly in scientific research.

GNNs rely on the *message-passing* mechanism. Through an iterative scheme, each node refines its representation by aggregating information from its local neighborhood to encode complex relational dependencies into learned vector embeddings. While this may be reminiscent of Convolutional Neural Networks (CNNs), a key challenge overcome by GNNs is learning from graph data without dependence on a fixed node ordering or an implicit spatial grid constraint that limits the direct applicability of architectures like CNNs to, for example, raw adjacency matrices. GNNs are designed to be permutation equivariant for node-level tasks (or invariant for graph-level tasks), thereby focusing on intrinsic topology and features. The general formulation is as follows:

At each layer $l$ of a GNN, the update from $\mathbf{h}_i^{(l)}$ to $\mathbf{h}_i^{(l+1)}$ for a node $i$ typically involves two phases:

1. **Message Computation:** For each neighbor $j \in \mathcal{N}(i)$, a message $\mathbf{m}_{ji}^{(l)}$ is computed by a 'Message' function. This function typically takes as input the representation of the source node $\mathbf{h}_j^{(l)}$, the representation of the target node $\mathbf{h}_i^{(l)}$ (though some variants might omit this), and the features of the edge $\mathbf{e}_{ji}$ connecting them:

$$\mathbf{m}_{ji}^{(l)} = \text{Message}(\mathbf{h}_j^{(l)}, \mathbf{h}_i^{(l)}, \mathbf{e}_{ji}) \tag{1.1}$$

2. **Aggregation and Update:** The messages $\mathbf{m}_{ji}^{(l)}$ from all neighbors $j \in \mathcal{N}(i)$ are aggregated into a single vector using a permutation-invariant 'Aggregate' function (e.g., sum, mean, max). This aggregated neighborhood information is then combined with the current representation of node $i$, $\mathbf{h}_i^{(l)}$,

by an 'Update' function (often a neural network layer or a simpler combination like concatenation followed by a linear transformation) to produce the new representation $\mathbf{h}_i^{(l+1)}$:

$$\mathbf{h}_i^{(l+1)} = \text{Update}\left(\mathbf{h}_i^{(l)}, \text{Aggregate}\left(\{\mathbf{m}_{ji}^{(l)} : j \in \mathcal{N}(i)\}\right)\right) \tag{1.2}$$

This layer-wise update process, stacking multiple such layers, enables GNNs to capture information from increasingly larger receptive fields, effectively integrating local node attributes with broader structural context [76]. Here, $\mathbf{h}_i^{(l)}$ is the feature vector of node $i$ at layer $l$. Messages $\mathbf{m}_{ji}^{(l)}$ from neighboring nodes $j \in \mathcal{N}(i)$ are aggregated (e.g., via sum, mean, or max pooling) and combined with $\mathbf{h}_i^{(l)}$ by an 'Update' function to produce the new representation $\mathbf{h}_i^{(l+1)}$. This iterative process enables GNNs to capture hierarchical patterns, integrating local node attributes with progressively broader structural context [76]. However, this reliance on neighborhood aggregation, while fundamental to their expressiveness, also implies that the fidelity of a node's learned representation is intrinsically tied to the integrity of its connected components.

### 1.1.1 Architectural Developments

The general message-passing framework has served as a blueprint for diverse GNN architectures. Graph Convolutional Networks (GCNs) [35], for instance, derive principles from spectral graph theory to define efficient, localized aggregation operations. GraphSAGE [29] introduced learnable aggregation functions and neighborhood sampling techniques, significantly improving scalability to large-scale graphs and enabling inductive learning. Graph Attention Networks (GATs) [66] integrated attention mechanisms, allowing nodes to assign parametrized relative importance to their neighbors during message aggregation. The field continues to advance with architectures such as Graph Transformers [20] for improved long-range dependency modeling, and specialized models for heterogeneous [68] and dynamic [58] graph data.

Concurrently, the theoretical understanding of GNNs has matured. The work by Xu et al. [76], linked the discriminative power of message-passing GNNs to the Weisfeiler-Lehman (WL) graph isomorphism test, and established that most GNNs are, at most, as powerful as the 1-WL test in distinguishing non-isomorphic graphs.

## 1.2 Graph Neural Network Applications in Scientific Domains

### 1.2.1 Material Science

In materials science, GNNs predict material properties, and aid prediction of viable materials. The Crystal Graph Convolutional Neural Network (CGCNN) transforms crystal structures into graph grids, where nodes are atoms and edges are bond lengths [81]. GNNs can take full molecular structures or geometries as input and learn representations for relevant entities. Geometric information, such as

interatomic distances, can be incorporated as edge features [57]. For crystals, where bonds may not be uniquely defined and 3D arrangement is critical, this is relevant [57]. The Representation Learning from Stoichiometry (Roost) model uses dense weighted maps for material components and stoichiometric ratios [57]. The SchNet model utilizes atomic numbers and atomic coordinates as inputs [57]. Such machine learning methods achieve predictive levels comparable to Density Functional Theory (DFT) [81]. The GNoME project uses GNNs for materials exploration. GNoME inputs graph data representing atomic connections and employs two pipelines: one creating candidates similar to known crystals, and another based on chemical formulas. Outputs are evaluated using DFT calculations, and results inform an active learning process for discovering stable materials (DeepMind GNoME) [49].

### 1.2.2 Biomedical Research

In biomedical research, GNNs analyze biological data from diverse data sources including genomic data and Knowledge Graphs (KGs). KGs like PrimeKG integrate multiple data sources, describing diseases, proteins, biological processes, pathways, anatomical features, phenotypes, and drugs with their therapeutic actions through millions of relationships [10]. GNNs operate on such KGs, for example, in medicine recommendation systems that process longitudinal medical records [51].

For protein structure prediction, as exemplified by AlphaFold [33] a landmark achievement of GNN-like architectures, the key idea involves representing amino acids as nodes and the proximity between amino acids as edges. This method represents the entire structure as a spatial graph. The GNN then utilizes this spatial graph representation as input for predicting the protein's structure.

DRAGONFLY's neural architecture combines a graph transformer neural network (GTNN) with an LSTM-based chemical language model (CLM). It uniquely uses interactome-based deep learning, incorporating information from multiple targets and ligands, unlike conventional CLMs reliant on transfer learning [3].

### 1.2.3 Particle Physics

In particle physics, GNNs perform node-level classification (NC) and regression for event reconstruction [63]. Our work focuses on unlearning manipulated information in such node classification settings.

***Pileup Mitigation.*** One approach [48] uses a gated graph network architecture. Each particle is a node, with connectivity based on proximity ( $\Delta$ R $<$ 0.3). Node features are updated by gating an R-dependent message with prior node states using a Gated Recurrent Unit (GRU). After multiple graph layers and a skip connection, a dense model predicts per-particle pileup probability. Another method [50] employs a graph attention network. It generates a global graph latent representation, which is then used to compute an output mapped back to each node based on a given node order. This improves jet and di-jet mass resolution and maintains stability across a range of pileup densities.

*Chapter 2*

# Introduction

## 2.1 Scaling GNNs: Opportunities and Risks

Graph Neural Networks (GNNs) are seeing widespread adoption across diverse domains, from recommender systems to drug discovery [74, 84]. This surge in adoption reflects a fundamental shift in how we approach machine learning for structured data, moving beyond traditional assumptions of independent and identically distributed samples to embrace the relational nature inherent in many real-world systems.

Recently, GNNs are being scaled to large training sets for graph foundation models [47]. However, in these large-scale settings, it is prohibitively expensive to verify the integrity of every sample in the training data that can potentially affect desiderata like fairness [36], robustness [27, 55], and accuracy [60]. Making the training process robust to minority populations [27, 32] is challenging and can adversely affect fairness and accuracy [61]. Consequently, model developers may want post-hoc ways to remove the adverse impact of manipulated training data if they observe problematic model behavior on specific distributions of test-time inputs. Such an approach follows the recent trend of using post-training interventions to ensure models behave in intended ways [54].

In scientific applications, data integrity concerns extend beyond traditional adversarial scenarios to include experimental errors, measurement artifacts, annotation mistakes, and systematic biases [52]. These issues pose particular challenges because they often exhibit systematic patterns that can mislead learning algorithms in potentially difficult-to-diagnose but critical ways. Consider molecular property prediction, where experimental measurements may contain systematic errors due to assay conditions, equipment calibration issues, or batch effects [79]. When such corrupted measurements become part of GNN training data, the message-passing mechanism can propagate these errors through chemical similarity networks, potentially affecting predictions for entire molecular families. Similar propagation effects occur in ecological networks, where misclassified species interactions can affect ecosystem stability assessments, and in brain connectivity analysis, where imaging artifacts could compromise disease classification models [69]

## 2.2 Removing adverse effects of parts of the training data

Recently, [25] formulated corrective unlearning as the challenge of removing adverse effects of manipulated data with access to only a representative subset for unlearning while being agnostic to the type of manipulations. We study this problem in the context of GNNs, which face unique challenges due to the graph structure. The traditional assumption of independent and identically distributed (*i.i.d.*) samples does not hold for GNNs, as they use a message-passing mechanism that aggregates information from neighbors. This process makes GNNs vulnerable to adversarial perturbations, where modifying even a few nodes can propagate changes across large portions of the graph and result in widespread changes in model predictions [7, 86]. Consequently, for GNNs to effectively unlearn, they must remove the influence of manipulated entities on their neighbors. *Corrective Unlearning* is an emerging paradigm that focuses on removing the influence of arbitrary training data manipulations on a trained model, using only a representative subset of the manipulated data [25].

In this work, we focus on the use of GNNs in node classification tasks, studying unlearning for targeted binary class confusion attacks [44] on both edges and nodes. For edge unlearning, we evaluate the unlearning of spurious edges that change the graph topology in a way that violates the homophily assumption that most GNNs rely on. For node unlearning, we utilize a label flip attack [44] which is used as a classical graph adversarial attack, similar to the Interclass Confusion attack [24].

## 2.3 Contributions

First, we **evaluate whether existing GNN unlearning methods are effective** in removing the impact of manipulated entities. Our findings reveal that these methods consistently fail, even when provided with a complete set of manipulated entities. **We then propose our method**, ***Cognac***, which unlearns by alternating between two components, as illustrated in Figure 2.1. The first component *Contrastive unlearning on Graph Neighborhoods* (***CoGN***), finds affected neighbors of the known deletion set, updating the GNN weights using a contrastive loss that pushes representations of the affected neighbors away from the deletion entities while staying close to other neighbors. The second component, *AsCent DesCent de↯coupled* (***AC↯DC***) applies the classic *i.i.d.* unlearning method of gradient ascent on the *deletion set* and gradient descent on the *retain set*. Our proposed method shows promise for corrective unlearning: it only beats retraining-from-scratch but also recovers most of the performance of an oracle (a model trained on the complete and correct data) while discovering as little as 5% of the manipulated entities.

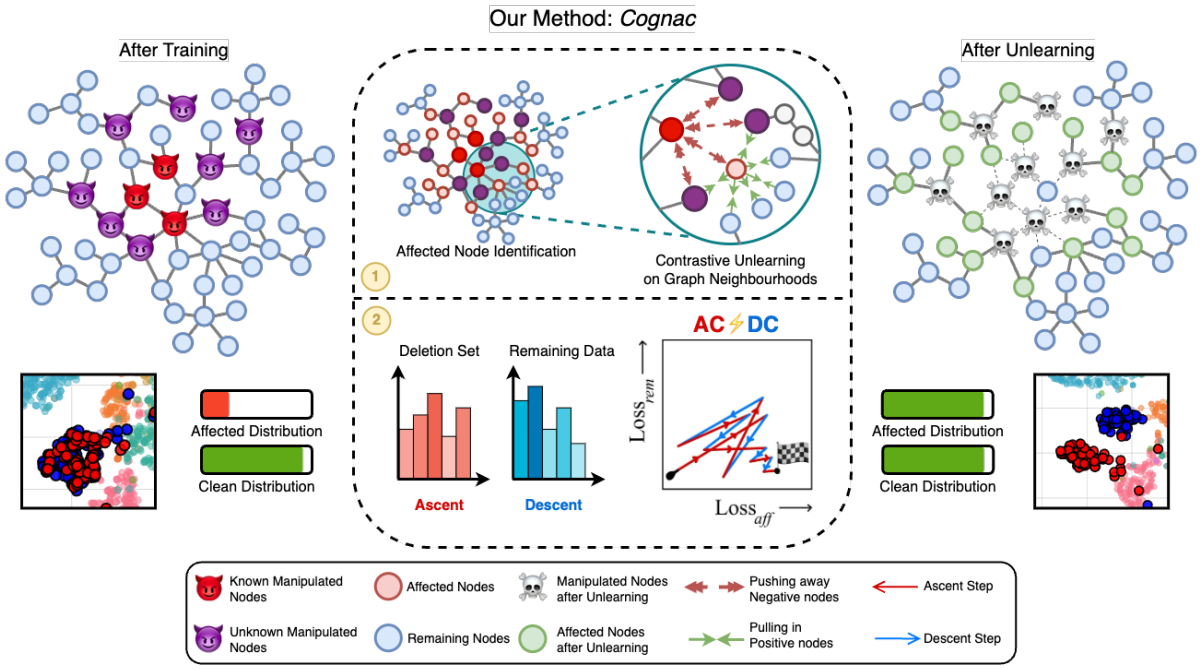


**Figure 2.1 Illustration of our method *Cognac*.** Initially (Left), the model is trained on manipulated data (Devils), out of which only a subset is identified for deletion (Dark-red-devils). Our method alternates between two steps. (1) Identifying neighbors by the deletion set, which can include both nodes from the remaining data (Light-red) and unidentified manipulated nodes (Purple), and pushes their representation away from the deletion set and toward other nodes in the neighborhood. (2) We then perform ascent on the deletion set labels, and descent on the remaining data with separate optimizer instances. This cleanly separates the embeddings of the affected classes (Right), recovering the accuracy on the affected distribution, and maintaining it on the remaining distribution.

*Chapter 3*

# Related Work

## 3.1 Attacks

Graph-based attacks, such as Sybil [19] and link spam farms [71], have long affected the integrity of social networks and search engines. Recent works reveal that even state-of-the-art GNN architectures are vulnerable to simple attacks on the trained models which either manipulate existing edges and nodes or inject new adversarial nodes [17, 22, 64, 87]. Parallelly, works have characterized the influence of specific nodes and edges that can guide such attacks [13].

## 3.2 Defenses

One strategy to mitigate the influence of such attacks is robust pretraining, such as using adversarial training [80, 82]. Post-hoc interventions like unlearning act as a complementary layer of defense, helping model developers when attacks slip through and affect a trained model. Removing the impact of manipulated entities begins with their identification [9], for which multiple strategies exist like data attribution [31], adversarial detection, and automated or human-in-the-loop anomaly detection [52].

While approaches like model debiasing and concept erasure [4, 21] can remove effects of identified manipulations post-training, they require knowledge of the manipulation. In contrast, unlearning methods are particularly valuable in adversarial settings where corruption effects may be deliberately obfuscated and impact multiple model behaviors simultaneously [55]. An important application of unlearning is for biosecurity, and chemical security: Weapons of Mass Destruction Proxy (WMDP) a benchmark consisting of multiple-choice questions (MCQ) for the evaluation of presence of hazardous knowledge in large language models (LLMs). It also serves as a benchmark for unlearning methods to remove such knowledge [40].

## 3.3 Machine Unlearning: Beyond Privacy

Recently, machine unlearning has received newfound attention beyond privacy applications [40, 41, 56, 62]. [25] demonstrated the distinction between the *Corrective* and *Privacy-oriented* unlearning settings for *i.i.d.* classification tasks, emphasizing challenges when not all manipulated data is identified for unlearning.

***Exact Unlearning*** arose in privacy applications, offering guaranteed removal of data influence through selective retraining [8, 11, 12].While perfect guarantees are valuable for privacy, the exponential cost of sequential deletions [70] becomes impractical as unlearning expands to broader challenges like model correction and debiasing [40, 56, 62].

This has driven the development of ***Inexact Unlearning*** methods that balance effectiveness with scalability, using either theoretical bounds for simple models [15, 73] or empirical validation for deep networks [14, 42, 72]. The recent formalization of *Corrective Unlearning* [25] opened a crucial new direction: removing corruption effects with only partial identification of manipulated data. We tackle this challenge in graphs, where the non-*i.i.d.* nature of data [59] results in the graph elements exerting a strong influence on other elements in their neighborhood.

*Chapter 4*

# Corrective Unlearning for Graph Neural Networks

We now formulate the *Corrective Unlearning* problem for graph-structured, non-*i.i.d.* data. We consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ represent the constituent set of nodes and edges respectively. For each node $\mathcal{V}_i \in \mathcal{V}$, there is a corresponding feature vector $\mathcal{X}_i$ and label $\mathcal{Y}_i$, with $\mathcal{V} = (\mathcal{X}, \mathcal{Y})$.

Consistent with prior work in unlearning on graphs [42, 72], we focus on semi-supervised node classification using GNNs. GNNs use the message-passing mechanism, where each node aggregates features from its immediate neighbors. The effect of this aggregation process propagates through multiple successive layers, effectively expanding the receptive field of each node with network depth. This architecture inherently exploits the principle of homophily, a common property in many real-world graphs where nodes with similar features or labels are more likely to be connected than not. While assuming homophily is extremely useful for learning representations from graph data, annotation mistakes or adversarial manipulations that create dissimilar neighborhoods or connect otherwise dissimilar nodes can easily harm the learned representations [87]. This motivates our study of post-hoc correction strategies like unlearning for GNNs. Following [25], we adopt an adversarial formulation that subsumes correcting more benign mistakes.

## 4.1 Adversary's Perspective

The adversary aims to reduce model accuracy on a target distribution by manipulating parts of the clean training data $\mathcal{G}$. This can be done in the following ways: (1) adding spurious edges $\hat{\mathcal{E}}$, resulting in $\mathcal{E}' = \mathcal{E} \cup \hat{\mathcal{E}}$; or (2) manipulating node information, $\mathcal{V}' = f_m(\mathcal{V})$, where $f_m$ manipulates a subset of nodes by changing their features or labels. We define $S_m$ as the set of manipulated entities, either the manipulated subset of nodes or the added spurious edges $\hat{\mathcal{E}}$. The final manipulated graph is denoted as $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$.

## 4.2 Unlearner's Perspective

After training, model developers may observe that desired properties like fairness and robustness are compromised in the trained model $\mathcal{M}$, which can be modeled as lower accuracy on some data distributions. The objective, then, is to remove the influence of the manipulated training data $S_m$ on the affected distribution while maintaining performance on the remaining entities. By utilizing data monitoring strategies on a subset of the training data or using incorrect data detection techniques like [52], it may be possible to identify a part of the manipulated entities $S_f \subseteq S_m$. For unlearning to be feasible, $S_f$ must be a representative subset of $S_m$. We only assume the type of affected entity (edges or nodes) is known to the model developer but do not assume any knowledge about the nature of manipulation.

An unlearning method $U(\mathcal{M}, S_f, \mathcal{G}')$ is then used to mitigate the adverse effects of $S_m$, ideally by improving the accuracy on unseen samples from the affected distribution. An effective unlearning method should remove the impact of certain training data samples without degrading performance on the rest of the data or incurring the cost of retraining from scratch.

Moreover, while Retrain was previously considered a gold standard in privacy-oriented unlearning and graph unlearning, [25] showed that when the whole manipulated set is not known, retraining on the remaining data can reinforce the manipulation, implying it's not a gold standard for corrective unlearning.

### 4.2.1 Metrics

To evaluate the performance of unlearning methods, we use the metrics proposed by [25]:

1. $\mathbf{Acc_{aff}}$: It measures the clean-label accuracy of test set samples from the affected distribution. This metric captures the method's ability to *correct* the influence of the manipulated entities on unseen data through unlearning. As the affected distribution differs for each manipulation, we specify it when describing each evaluation.

2. $\mathbf{Acc_{rem}}$: It is defined as the accuracy of the remaining entities. This metric measures whether the unlearning maintains model performance on clean entities.

The metrics $\mathrm{Acc}_{\mathrm{aff}}$ and $\mathrm{Acc}_{\mathrm{rem}}$ were termed "Corrected Accuracy" ($\mathrm{Acc}_{\mathrm{corr}}$) and "Retain Accuracy" ($\mathrm{Acc}_{\mathrm{retain}}$) respectively by [25]. We chose alternative names to explicitly state which data distribution accuracy is measured. In Section 7, we further specify what the "affected distribution" and "remaining entities" are for the different evaluation types we study.

### 4.2.2 Goal

An ideal corrective unlearning method should have high $\mathrm{Acc}_{\mathrm{aff}}$ even when a small fraction of manipulated set ($S_m$) is identified for deletion ($S_f$) without big drops in $\mathrm{Acc}_{\mathrm{rem}}$, all while being computationally efficient.

*Chapter 5*

# Preliminary Work: Sanity Checks For Graph Unlearning*

Our preliminary work motivated the bulk of this research into more comprehensive evaluations for assessing the true utility of graph unlearning algorithms, and informed the development of our novel method. While graph unlearning methods previously posited that *i.i.d.* methods are simply unsuitable for the graph domain, and hence not worth evaluating, we still sought to see how much we could achieve with no graph awareness. For this initial experiment, we chose SCRUB, and compared it against MEGU.

We found that SCRUB performs comparably or better than MEGU on random node removal and on removing an adversarial node injection attack. Our results suggest that 1) graph unlearning studies should incorporate general unlearning methods like SCRUB as baselines, and 2) there is a need for more rigorous behavioral evaluations that reveal the differential advantages of proposed graph unlearning methods.

## 5.1 Problem Setting

We perform a semi-supervised multi-label classification using a Graph Convolutional Network (GCN) [35] and compute loss using cross-validation. Node unlearning in graph neural networks involves removing the influence of specific nodes, which may have manipulated features or perturb the topology of the original graph. We perform node-level unlearning in two settings: the random node removal (a benign form of unlearning request common in graph unlearning evaluations) and node injection attack removal, an adversarial form of unlearning requests.

### 5.1.1 Desiderata

The overarching objective for all unlearning requests is to forget efficiently, while minimising the impact on the overall model utility [37]. Ideally, the performance of the unlearned GCN should resemble that of a fresh GCN retrained solely on clean samples. We aim to move away from the current forget set labels without assuming ground truth label knowledge.

*This chapter is based on our paper: Kolipaka, V., Sinha, A., Mishra, D., Kumar, S., Arun, A., Goel, S., & Kumaraguru, P. (2024). Sanity Checks for Evaluating Graph Unlearning CoLLAs Workshop https://lifelong-ml.cc/Conferences/2024/acceptedpapersandvideos/conf-2024-71

In graph unlearning literature, various proxies measure "forgetting" - Accuracy [28], F1 scores [16, 72], and Membership Inference Attacks [14,53,78] (commonly used in privacy settings). Our evaluation considers two settings: random deletion requests and poisoned node deletion. We compute Micro-F1 score, which is the main metric used to report MEGU's results [42], as well as Accuracy and Poison Success Rate [43].

### 5.1.2 Formulation and Notation

***Graph Notation.*** We formalize node unlearning similar to [42],

- $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ represents a graph $\mathcal{G}$ with $|\mathcal{V}| = n$ nodes, $|\mathcal{E}| = m$ edges, and $\mathcal{X}(i)$ corresponding to the feature vector of node $i$, where $V_{tr}$ corresponds to the set of nodes used for training and $V_{test}$ corresponds to the set of nodes used for testing.
- *Forget Set*: Let $\mathcal{V}_f \subseteq \mathcal{V}_{tr}$ denote the set of nodes to be unlearnt.
- *Retain Set*: Let $\mathcal{V}_r \subseteq \mathcal{V}_{tr}$ denote the set of nodes to be retained.

***Model Utility.*** We operationalise this by computing the Micro-F1 score on $\mathcal{V}_{test}$. An ideal unlearning method should minimally impair the performance of the model, as tested against clean data points.

***Influence of Samples to be Removed.*** We operationalize this as the Micro-F1 score on samples that are in the forget set ($\mathcal{V}_f \subseteq \mathcal{V}_{tr}$). An effective unlearning method aims to minimise this score. In case of unlearning injection attacks, however, we measure this by applying the poison to a copy of the test nodes, where it is also referred to as *Poison Success Rate.*

## 5.2 Experiments

### 5.2.1 Setup

We evaluate on two benchmark datasets, Cora and CiteSeer [5], commonly used in graph unlearning literature. Cora contains 2,708 nodes, each with 1,433 features belonging to one of its 7 classes and 5,429 unique edges. CiteSeer contains 3,327 nodes, each with 3,703 features belonging to one of its 6 classes and 4,732 unique edges. We use two methods for this preliminary experiment: ***SCalable Remembering and Unlearning unBound***, which we benchmark to test the value in benchmarking general methods,***Mutual Evolution Graph Unlearning (MEGU)***, a SOTA graph unlearning method, both of which are described in Chapter 7.

We split both datasets following the guidelines of recent graph unlearning approaches [14, 42], randomly assigning 80% of nodes for training and 20% for testing. For simplicity and generalizability, we limit the scope of our experiments to the GCN model. In all experiments, we train a 2-layer GCN with a hidden dimension of 16, using an Adam Optimizer with a learning rate of 0.025 and weight decay of $5 \times 10^{-4}$, and train for 200 epochs.

To ensure a fair comparison between the unlearning methods, both SCRUB and MEGU are run for 100 epochs. For each dataset, we also conduct an extensive search on the hyperparameters recommended in the respective papers, as described in Appendix 5.3.

### 5.2.2 Evaluation 1 - Random Unlearning

***Motivation.*** Unlearning random nodes is the prevalent evaluation to benchmark Graph Unlearning works [59]. This form of unlearning request is typical in privacy related scenarios. In this setting, a subset of nodes belonging to $\mathcal{V}_{tr}$ are randomly chosen to be deleted from the graph. We vary the size of the deletion set from 5% to 50% of the total nodes in the training set.

**Table 5.1** ***Random node unlearning results on Cora and CiteSeer.*** $F1_{\mathcal{V}_{\text{test}}}$ indicates F1 score on the $V_{\text{test}}$ where higher indicates more model utility. $F1_{\mathcal{V}_f}$ indicates the F1 score on $\mathcal{V}_f$, where lower indicates more forgetting.

| **Deletion Size** | **Retrain From Scratch** | | **SCRUB** | | **MEGU** | |
|---|---|---|---|---|---|---|
| | $F1_{\mathcal{V}_{test}}$ | $F1_{\mathcal{V}_f}$ | $F1_{\mathcal{V}_{test}}$ | $F1_{\mathcal{V}_f}$ | $F1_{\mathcal{V}_{test}}$ | $F1_{\mathcal{V}_f}$ |
| **Cora** | | | | | | |
| 5% | 86.24% | 83.33% | 85.87% | 87.50% | 85.87% | 87.50% |
| 10% | 86.61% | 84.97% | 85.87% | 87.56% | 85.69% | 87.56% |
| 20% | 85.32% | 86.53% | 86.06% | 89.64% | 85.87% | 89.64% |
| 30% | 85.14% | 87.22% | 86.06% | 89.81% | 85.87% | 87.50% |
| 40% | 86.24% | 83.33% | 85.87% | 87.50% | 85.87% | 90.80% |
| 50% | 85.69% | 86.74% | 86.24% | 90.36% | 85.87% | 90.36% |
| **CiteSeer** | | | | | | |
| 5% | 81.91% | 72.36% | 80.57% | 75.61% | 80.12% | 75.61% |
| 10% | 81.32% | 74.09% | 81.02% | 76.11% | 80.27% | 76.11% |
| 20% | 83.11% | 72.93% | 80.57% | 75.56% | 80.42% | 75.56% |
| 30% | 82.36% | 73.22% | 81.02% | 75.64% | 80.42% | 75.64% |
| 40% | 81.91% | 74.27% | 81.02% | 76.59% | 80.42% | 76.19% |
| 50% | 82.06% | 74.01% | 80.57% | 76.92% | 79.97% | 76.76% |

***Results.*** Random node unlearning results are presented in Table 5.1. SCRUB performs on par with MEGU on both retaining model utility and lowering accuracy on the forget set across deletion sizes. Both methods are worse than the retrain-from-scratch gold standard. These results suggest that general unlearning methods may perform just as well as specialised graph unlearning methods under this evaluation.

### 5.2.3 Evaluation 2 - Unlearning Injection Attacks

***Motivation.*** The interconnected nature of graph data introduces a critical vulnerability: a small number of manipulated or poisoned nodes in the graph can adversely affect the performance across

the entire network. These manipulations can arise from various sources, such as adversarial attacks, data biases, or even errors in the feature or label collection process [25]. Consequently, there may be scenarios where we identify these manipulated nodes after training a GNN model and wish to remove their influence without retraining the entire model from scratch.

***Adversary's Perspective.*** From the adversary's point of view, they can inject an arbitrary number of nodes with a feature vector of their choice into the graph. This simulates scenarios that necessitate unlearning requests, such as removing backdoors, biases, and confusions, manifesting as spurious correlations learned due to mislabeling or feature errors.

***Poisoning Attack.*** For *Cora*, we inject 50 poisoned nodes with a trigger size of 50 features, targeting Class 1 as the poisoned label. For *CiteSeer*, we inject 30 poisoned nodes with a trigger size of 150 features, targeting Class 0 as the poisoned label. The target classes are chosen to be consistent with MEGU's evaluation setting.

**Table 5.2** Poisoned node unlearning results on Cora and CiteSeer. SCRUB as proficient as MEGU at retaining model utility, and even outperforms MEGU in deleting the effects of the poison as measured by the decrease in the poison success rate. F1 Score results, which indicate model utility, are presented on the clean test set of the data. (↑) indicates a higher value is better, while (↓) indicates a lower value is better. Best performance is indicated in **bold**.

| | Cora | | CiteSeer | |
|---|---|---|---|---|
| **Unlearning Method** | **F1 Score** (↑) | **Poison Success Rate** (↓) | **F1 Score** (↑) | **Poison Success Rate** (↓) |
| No Unlearning | 90.83% | 98.53% | **80.41%** | 99.70% |
| SCRUB | 91.19% | **0%** | 78.62% | **0%** |
| MEGU | **93.39%** | 9.17% | 76.53% | 11.95% |

***Metrics.*** We evaluate the unlearning methods using the following metrics:

1. **Retention of Model Utility (F1)** To assess the model's overall performance after unlearning, we measure the F1 score on the original test set $\mathcal{V}_{test}$. The F1 score, which aggregates the contributions of all classes, is particularly useful for evaluating multi-class accuracy, especially in imbalanced datasets.

2. **Poison Success Rate (PSR)** The proportion of times the model predicts the adversary's intended labels (poison labels) in the poisoned dataset, or formally, $F1_{\mathcal{V}_f}$. A higher PSR (closer to 1) indicates a more successful poison attack. On the other hand, the unlearning method's ability to mitigate the adverse effects of the poisoned nodes is measured by the decrease in PSR after unlearning.

***Results.*** The results of the evaluation are presented in Table 5.2. We observe that GCN is highly susceptible to these attacks, as the classical poisoning attack achieves a high poison trigger success rate while maintaining the overall model utility. We find that not only is SCRUB as proficient as MEGU at retaining model utility, it even **outperforms MEGU in deleting the effects of the poison.** These

results suggest that general unlearning methods are an important baseline even for graph unlearning settings. Further, given that both evaluations fail to differentiate SCRUB and MEGU, there is a need for evaluations that better measure the unlearning of propagated poisoning due to message passing in GNNs.

## 5.3 Hyperparameter Tuning

This section contains details about the hyperparameter tuning performed for both MEGU and SCRUB. We follow the authors and use the same set of hyperparameters for evaluations.

**Table 5.3** Candidate values used for hyperparameter tuning

| Hyperparameter | Values |
|---|---|
| **SCRUB** | |
| $\alpha$ | [0.5, 1, 2] |
| $msteps$ | [10, 25, 50] |
| Learning Rate | [0.025, 0.01, 0.001] |
| **MEGU** | |
| $\alpha 1$ | [0.1, 0.5, 0.8, 0.24] |
| $\alpha 2$ | [0.1, 0.5, 0.8, 0.12] |
| $\kappa$ | [0.01, 0.1, 0.05] |
| Learning Rate | [0.01, 0.05, 0.09, 0.1] |

**Table 5.4** Selected hyperparameters for hyperparameter tuning for SCRUB and MEGU over Cora and CiteSeer

| Dataset | SCRUB | | | MEGU | | | |
|---|---|---|---|---|---|---|---|
| | $\alpha$ | $msteps$ | Learning Rate | $\alpha 1$ | $\alpha 1$ | $\kappa$ | Learning Rate |
| Cora | 1 | 10 | 0.01 | 0.8 | 0.5 | 0.01 | 0.09 |
| CiteSeer | 0.05 | 25 | 0.01 | 0.8 | 0.5 | 0.1 | 0.1 |

## 5.4 Conclusion

In this preliminary work, we compare a state-of-the-art method proposed specifically for graph unlearning (MEGU) with a generic classification unlearning method (SCRUB) and note that their performances are comparable in the commonly tested setting of random node deletions. We hypothesize that this may be because the random node deletion evaluation is not strong enough to test the interesting aspect of graph unlearning, i.e., the interdependence of node representations. We then test the unlearning using a simple node injection poisoning attack and find that on the same metrics used to report MEGU's

performance, SCRUB outperforms on this task. Thus, our work shows the importance of including standard unlearning baselines in the graph unlearning setting, as they may be surprisingly effective. It also follows that there is a need to design rigorous evaluations for graph unlearning, and this formed a strong consideration for the rest of our work that followed this preliminary experiment.

*Chapter 6*

# Our Method: *Cognac*[*]

Our proposed unlearning method, *Cognac*, requires access to the underlying graph $\mathcal{G}'$, the known set of entities to be deleted $S_f$, and the original model $\mathcal{M}$. We define $\mathcal{V}_f$ as the set of nodes whose influence is to be removed. For node unlearning, $\mathcal{V}_f = S_f$; for edge unlearning, $\mathcal{V}_f$ is the set of vertices connected to the edge set to be deleted.

Manipulated data has two main adverse effects on the trained GNN: 1) Message passing can propagate the influence of the manipulated entities $S_m$ on their neighborhood, and 2) The layers learn transformations to fit potentially wrong labels in $S_m$. Mitigating the effects of attacks first requires analysis of the impact of such attacks. The most general form of attack is Interclass Confusion [24], which we show entangles the representations of two classes, below.

**Theorem 6.1.** *Let $G = (V, E, X)$ be a graph with node set $V$, edge set $E$, and features $X$. Let $C_1, C_2 \subset V$ be two distinct classes with ground-truth labels $y_i \in \{C_1, C_2\}$. Suppose an Interclass Confusion (IC) attack is applied.*

*Let $\phi_M(C_1), \phi_M(C_2) \in \mathbb{R}^d$ denote the mean embeddings of $C_1$ and $C_2$, and $\mathcal{D}(\phi_M(C_1), \phi_M(C_2))$ be the Wasserstein-2 distance between their embedding distributions. Then, there exists a degradation term $\Delta > 0$ such that:*

$$\mathbb{E}\left[\mathcal{D}(\phi_M(C_1), \phi_M(C_2))\right] \leq \mathbb{E}\left[\mathcal{D}(\phi_{M_{clean}}(C_1), \phi_{M_{clean}}(C_2))\right] - \Delta$$

Complete list of assumptions and attack details in Appendix A.3. We tackle these two problems using separate components - CoGN and AC$\not{z}$DC.

## 6.1 Removing Effects on Neighbors with CoGN

***First,*** *how can we remove the influence of manipulated entities on their neighboring nodes?*

[*]Chapters 2-4 and 6-8 are based on our paper: Kolipaka, V., Sinha, A., Mishra, D., Kumar, S., Arun, A., Goel, S., & Kumaraguru, P. (2024). A Cognac shot to forget bad memories: Corrective Unlearning in GNNs. arXiv preprint arXiv:2412.00789

First, this requires us to identify the nodes affected by the manipulations ($\mathcal{V}_{\text{aff}}$) and then mitigate the influence on their representations. Identifying affected nodes is challenging, as the impact of message passing from manipulated entities $S_m$ depends on the interference from messages of other neighboring nodes. Therefore, we use an empirical estimation to identify the affected nodes from each entity in the deletion set. On these nodes, we then perform contrastive unlearning, simultaneously pushing the representations of the affected nodes away from nodes in $\mathcal{V}_f$ while keeping them close to other nodes in their neighborhood. We call this component *Contrastive unlearning on Graph Neighborhoods* (***CoGN***), formalized below.

### 6.1.1 Affected Node Identification

To first identify the affected samples, $\mathcal{V}_{\text{aff}}$, we first observe that any manipulated node $s \in V_f$ can only affect the representation of any other node $v \in \mathcal{V}' \backslash V_f$ in $\mathcal{G}'$ only if $s$ is part of the *receptive field* of $v$, a widely known result for GNNS. Formally,

**Lemma 6.2.** *Let $G = (V, E)$ be an undirected graph, and let $\mathcal{N}(v)$ denote the 1-hop neighborhood of node $v$. For a node $s \in V$, let $\mathcal{N}^n(s)$ denote the $n$-hop neighborhood of $s$, defined recursively as:*

$$\mathcal{N}^n(s) = \begin{cases} \{s\} & \text{if } n = 0, \\ \bigcup_{v \in \mathcal{N}^{n-1}(s)} \mathcal{N}(v) & \text{if } n > 0. \end{cases}$$

*In an $n$-layer GNN, the representation $z_s$ of node $s$ can affect the representations $z_v$ of nodes $v$ only if $v \in \mathcal{N}^n(s)$. For any $v \notin \mathcal{N}^n(s)$, $z_v$ is independent of message passing effects of $z_s$.*

The proof is provided in Appendix A.1. The main takeaway is that manipulations only propagate within an $n$-hop neighborhood of poisoned nodes. This locality drastically reduces the search space for identifying affected nodes.

***Second,*** *are all the nodes in the $n$-hop neighborhood of the manipulated nodes may be affected enough by the attack?*

Studies have shown that some nodes are more robust to the perturbations than others even within this n-hop neighbourhood [2, 26]. This observation serves two purposes: 1) Identifying affected nodes outside of $S_f$ that are either manipulated or affected due to the structural dependency 2) We can restrict our candidates for unlearning to only those entities that are functionally affected. To find the most affected nodes we employ a cheap, and simple heuristic. We invert the features of $v \in V_f$ and select neighboring nodes where final output logits are changed the most. Formally, the inversion is performed by the transformation $\vec{1} - \mathcal{X}_v \; \forall \; v \in \; \mathcal{V}_f$, leading to a new feature matrix $\chi'$, where $\mathcal{X}_v$ represents a one-hot-encoding vector. We then compute the difference in the original output logits $\mathcal{M}(\chi)$, and those obtained by on the new feature matrix, $\mathcal{M}(\chi')$ given by: $\Delta\chi = |\mathcal{M}(\chi') - \mathcal{M}(\chi)|$. The top $k\%$ nodes with the most change, $\Delta\chi$, are selected as the affected set of entities $\mathcal{V}_{\text{aff}}$. Appendix E.2 varies our

design choices, confirming that we retain the same performance as using the entire $n$-hop neighborhood while being more efficient (Figure E.3). Our method works robustly even if the original GNN was under-trained (Figure E.2).

### 6.1.2 Contrastive Unlearning

To remove the influence of the deletion set $\mathcal{V}_f$ on the affected nodes $\mathcal{V}_{\text{aff}}$ identified in the previous step, we must ensure that our model maps the hidden representations of nodes in $\mathcal{V}_{\text{aff}}$ far away from that of nodes in $\mathcal{V}_f$ [25].

However, satisfying this property alone will lead to unrestricted separation and damage the quality of learned representations. To achieve this balance and restore homophily [45], we constrain $\mathcal{V}_{\text{aff}}$ to stay close to its unaffected neighbors $\mathcal{N}_{\mathcal{V}_{\text{aff}}} \backslash \mathcal{V}_f = \mathcal{V}_{pos}$, ensuring alignment with the local structure while distancing from $\mathcal{V}_f$. Formally, we can state this as the following optimization problem over the parameters of a GNN which is necessary for corrective unlearning,

$$\begin{aligned} &\max_{\theta} \left( \mathbb{E}_{v \in \mathcal{V}_{\text{aff}},\, p \in \mathcal{V}_{pos}} \left[ \mathbf{z}_v^\top \mathbf{z}_p \right] - \mathbb{E}_{v \in \mathcal{V}_{\text{aff}},\, n \in \mathcal{V}_f} \left[ \mathbf{z}_v^\top \mathbf{z}_n \right] \right) \\ &= \max_{\theta} \left( \mathbb{E}_{v \in \mathcal{V}_{\text{aff}}, p \in \mathcal{V}_{pos}, n \in \mathcal{V}_f} \left[ \mathbf{z}_v^\top \mathbf{z}_p - \mathbf{z}_v^\top \mathbf{z}_n \right] \right) \end{aligned} \tag{6.1}$$

Equation 6.1 offers a direct way to enforce separation between positive and negative pairs, but it has notable shortcomings. In particular, it does not sufficiently penalize small margins, as it only considers the raw similarity difference without emphasizing cases where $z_v^T z_p$ and $z_v^T z_n$ are nearly equal. This can lead to weak separation and reduced robustness, especially when positive and negative embeddings are closely aligned. Additionally, the lack of non-linear scaling creates an uneven optimization landscape, increasing the risk of convergence issues.

To overcome these limitations, we use a log-based loss function that applies non-linear sigmoid terms, providing stronger penalization for small margins and a probabilistic similarity interpretation [29]. This enhances the separation between positive and negative pairs while ensuring smoother gradients for more stable optimization [39]. Formally,

$$\mathcal{L}_{v,p,n} = -\log(\sigma(z_v^T z_p)) - \log(\sigma(-z_v^T z_n)) \tag{6.2}$$

Under assumptions of differentiability, convexity, and bounded gradients, we can guarantee that optimizing this function achieves a better separation between the positive and negative dot products equivalent to Equation 6.1.

**Table 6.1 Dataset statistics**. In our evaluations, we include citation networks, co-author networks, and co-purchase networks. The number of nodes and edges reported here is for the whole dataset. From this, we use a 60/20/20 split for train/validation/test. The manipulation statistics are presented in Table B.1.

| DATASET | CLASSES | NODES | EDGES |
|---|---|---|---|
| CORAFULL | 70 | 18,800 | 125,370 |
| CORA | 7 | 2,485 | 10,138 |
| CITESEER | 6 | 2,120 | 7,358 |
| DBLP | 4 | 16,191 | 103,826 |
| PUBMED | 3 | 19,717 | 88,648 |
| OGB-ARXIV | 40 | 169,343 | 1,166,243 |
| CS | 15 | 18,333 | 163,788 |
| PHYSICS | 5 | 34,493 | 495,924 |
| PHOTOS | 8 | 7,487 | 238,086 |
| COMPUTERS | 10 | 13,381 | 491,556 |

**Theorem 6.3.** *Let $\theta \in \mathbb{R}^d$ parameterize a GNN generating embeddings $z_v, z_p, z_n \in \mathbb{R}^k$ for triplets $(v, p, n)$. Let $\mathcal{L}(\theta) = -\mathbb{E}\left[\log \sigma(z_v^\top z_p) + \log \sigma(-z_v^\top z_n)\right]$ and $S(\theta) = \mathbb{E}\left[z_v^\top z_p - z_v^\top z_n\right]$.*

*Then under the assumptions stated above, gradient descent on $\mathcal{L}(\theta)$ with step size $\eta \leq \frac{1}{L}$ (where $L$ is the Lipschitz constant of $\nabla_\theta \mathcal{L}$) guarantees: $S(\theta_{t+1}) \geq S(\theta_t) \quad \forall t \geq 0$, and at convergence, $S(\theta^*) > S(\theta_0)$.*

The proof is presented in Appendix A.2. We also empirically validate this theorem for non-linear GNNs, with Figure 6.1 showing on the *Cora* dataset that the separation between the positive dot product and negative dot product monotonically increases as the loss converges.

## 6.2 Unlearning Old Labels with AC↯DC

Next, we ask: *Can we undo the effect of the task loss $\mathcal{L}_{\text{task}}$ explicitly learning to fit the node representations of $S_m$ to potentially wrong labels?*

We do this by performing gradient ascent on $S_f$, which non-directionally maximizes the training loss concerning the old labels. Ascent alone aggressively leads to arbitrary forgetting of useful information, so we counterbalance it by alternating with steps that minimize the task loss on the remaining data. More precisely, we perform gradient ascent on $\mathcal{V}_f$ and gradient descent on $\mathcal{V}_r$, iteratively on the original GNN $\mathcal{M}$.

$$\mathcal{L}_a = -\mathcal{L}_{\text{task}}(\mathcal{V}_f),\ \mathcal{L}_d = \mathcal{L}_{\text{task}}(\mathcal{V}_r) \tag{6.3}$$

While variants of ascent on $S_f$ and descent on remaining data have been studied for image classification [38] and language models [77], we find a specific optimization strategy useful to achieve corrective unlearning on graphs. The challenge arises when $S_f \subset S_m$, as the remaining data could still contain manipulated entities, which we aim to avoid reinforcing. However, in realistic scenarios, the manipulated entities $S_m$ typically are a small fraction of the training data, allowing us to mitigate their impact through ascent on the representative subset $S_f$.

This requires a careful balance between ascent and descent, which we can achieve by using two different optimizers and starting learning rates for these steps. This insight is similar to prior work in Generative Adversarial Networks (GANs) [30]. The starting learning rates for both ascent and descent are hyperparameters to be tuned, and usually, we find a lower learning rate for ascent leads to better results. The importance of decoupling optimizers is shown by results in Table E.1. Thus, we call this component *Ascent Descent de↯coupled* (***AC↯DC***) to emphasize the distinction from existing variants of ascent on $S_f$ and descent on remaining data. Convergence of this formulation has been shown in previous works [38]. We show the empirical convergence of AC↯DC in Figure E.1.

For our final method ***Cognac***, we alternate steps of *CoGN*, which fixes representations of affected neighborhood nodes, and *AC↯DC*, which unlearns potentially wrong labels introduced by $S_m$.

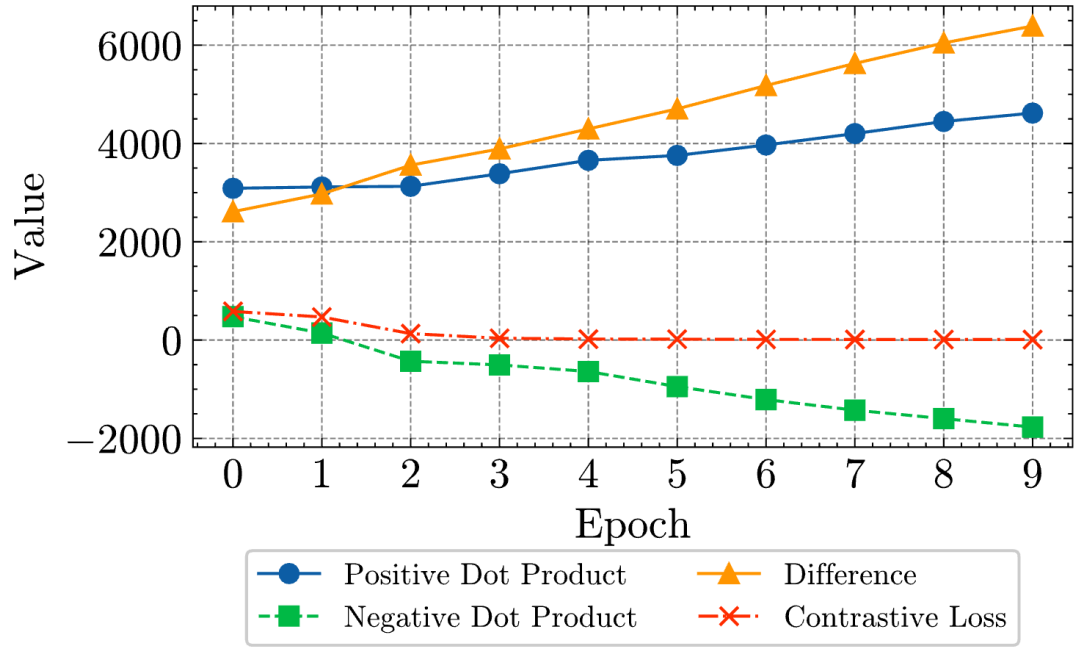


**Figure 6.1 Empirical convergence of CoGN.** The average positive and negative dot products across samples increase and decrease respectively over epochs, resulting in overall convergence.

*Chapter 7*

# Experimental Setup

## 7.1 Benchmarking Details

We now describe design choices made for benchmarking, first specifying the datasets and architectures, then how to ensure a fair comparison between methods.

### 7.1.1 Models and Datasets.

We report results using the Graph Convolutional Network (GCN) [35] architecture and evaluate the methods on ten benchmark datasets used in prior literature: Cora, PubMed, DBLP, Coauthor CS, Coauthor Physics, Amazon Photos (Photos), Amazon Computers (Computers), CiteSeer [14, 42] and additionally CoraFull [5] and OGB-arXiv [67].

### 7.1.2 Hyperparameter Tuning.

Ensuring a fair comparison of unlearning methods can be tricky as there are multiple desiderata: unlearning, maintaining utility, and computational efficiency, and hyperparameter tuning of the methods can particularly affect results on GNNs. We describe our efforts towards this in Appendix sections F.1, F.2.

## 7.2 Evaluations

Given a fixed budget of samples to manipulate, ideal corrective unlearning evaluations should maximally deteriorate model performance on the affected distribution so that there is a wide gap between clean and poisoned model performance to measure unlearning method progress. We thus evaluate unlearning on attacks not constrained by stealthiness.

[44] show that binary label flip manipulation attacks, where a fraction of labels are swapped between two chosen classes, are stronger than multi-class manipulations, theoretically and empirically, on GNNs. Building on this, we use two targeted attacks to evaluate corrective unlearning on graph data.

### 7.2.1 Spurious Edge Addition.

Prior GNN unlearning works [42, 72] have evaluated adversarial edge attacks, but in an untargeted setting, making their evaluations weak. We, instead, simulate targeted adversarial edge insertions between nodes of two classes, violating homophily assumptions and entangling their representations. This models attacks like fake social connections [6] or knowledge graph manipulations [75, 83, 85]. Unlearning aims to recover accuracy on the targeted classes $\mathrm{Acc}_{\mathrm{aff}}$ while preserving performance on others $\mathrm{Acc}_{\mathrm{rem}}$.

### 7.2.2 Label Manipulation.

We implement the *Interclass Confusion (IC) Test* [24], by systematically swapping labels between two targeted classes to entangle their representations. Once again, the unlearning goal is to improve $\mathrm{Acc}_{\mathrm{aff}}$ on the two targeted classes, while preserving performance, $\mathrm{Acc}_{\mathrm{rem}}$, on the remaining classes.

## 7.3 Baselines

We evaluate four popular graph unlearning methods and adapt one popular *i.i.d.* unlearning method for graphs. For reference, we also report results for the Original model, Retrain, which trains a new model without $S_f$, and Finetune, which continues training the poisoned model on data without $S_f$ for additional epochs.

Following [25], we find that retraining from scratch is not the gold standard in corrective unlearning. Thus, we introduce Oracle, trained on the whole training set without manipulations, indicating an upper bound on what can be achieved. The Oracle has correct labels for the unlearning entities, information that the unlearning methods cannot access.

We choose five methods as baselines where unlearning incorrect data explicitly motivates the technique.

(1) ***GNNDelete*** [14] adds a deletion operator after each GNN layer and trains them using a loss function to randomize the prediction probabilities of deleted edges while preserving their local neighborhood representation, keeping the original GNN weights unchanged.

(2) ***GIF*** [72] draw from a closed-form solution for linear-GNN to measure the structural influence of deleted entities on their neighbors. Then, they provide estimated GNN parameter changes for unlearning using the inverse Hessian of the loss function.

(3) ***MEGU*** [42] finds the highly influenced neighborhood (HIN) of the unlearning entities and removes their influence over the HIN while maintaining predictive performance and forgetting the deletion set using a combination of losses.

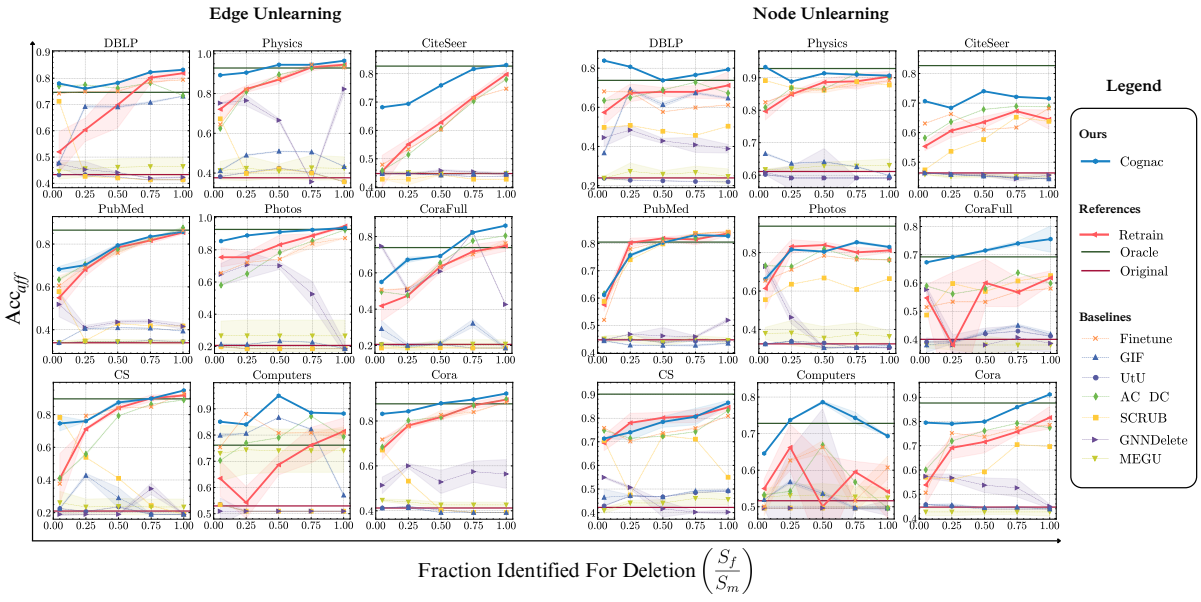


**Figure 7.1 Corrective Unlearning Results.** We report the accuracy on the affected classes $\text{Acc}_{\text{aff}}$ across different fractions of the manipulation set known for deletion $(S_f/S_m)$. Baseline methods perform poorly, except for GNNDelete, which achieves reasonable unlearning performance in some settings. AC↯DC and Finetune, despite not being graph-specific, perform much better. *Cognac*, which adds graph awareness to AC↯DC, archives SOTA across datasets and corrective fractions, unlearning the effect of the manipulation with just 5% of the manipulation set known.

(4) ***UtU*** [65] proposes *zero-cost* edge-unlearning by removing the edges to be deleted during inference for blocking message propagation from nodes linked to these edges.

Finally, we include a popular unlearning method studied in *i.i.d.* classification settings.

(5) ***SCRUB*** [38] employs a teacher-student framework with alternate steps of distillation away from the forget set and towards the retain set. For edge unlearning, we use SCRUB by taking the nodes connected to spuriously added edges as the forget set and the rest as the retain set.

*Chapter 8*

# Results and Discussion

We now report our main results comparing our method to existing methods across the manipulation types and datasets. Detailed method ablations and analyses of what can be achieved in this setting are reported in Appendix E.4. We also present consistent results on the GAT architecture in Appendix D.1. We show the robustness of *Cognac* to large $S_f$ sizes, showing strong performance even with $38.96\%$ of total training nodes in $S_f$ for the *PubMed* dataset.

**Table 8.1 Accuracy averaged across $S_f/S_m$ on remaining distribution relative to the Original model.** We find prior methods, especially GNNDelete, lead to large drops in $\mathrm{Acc}_{\mathrm{rem}}$, while our methods *Cognac* and AC↯DC minimize the loss in $\mathrm{Acc}_{\mathrm{rem}}$.

| METHOD | CS | | CORA | |
|---|---|---|---|---|
| | EDGE | LABEL | EDGE | LABEL |
| ORIGINAL | 90.6 | 89.6 | 61.2 | 61.4 |
| ORACLE | −0.1 | +0.5 | −0.7 | −3.0 |
| RETRAIN | −1.4 | −0.1 | −2.4 | −5.7 |
| *Cognac* | −1.4 | −0.4 | −4.5 | −2.9 |
| AC↯DC | −0.7 | −0.8 | −2.2 | −0.8 |
| GNNDELETE | −6.0 | −1.9 | −10.9 | −8.4 |
| GIF | −1.6 | −0.9 | −4.2 | −0.6 |
| MEGU | −0.5 | −2.8 | +0.0 | −6.5 |
| UTU | +0.0 | +0.0 | +0.0 | +0.0 |
| SCRUB | −0.9 | −0.7 | +0.0 | −4.8 |

Figure 7.1 shows unlearning performance on the test set for manipulated classes ($\mathrm{Acc}_{\mathrm{aff}}$) upon varying the fraction of the manipulation set known for unlearning ($S_f/S_m$). Table 8.1 accompanies this showing side-effects on utility.

**1. Existing unlearning methods perform poorly even when $|S_f| = |S_m|$.** Observing the rightmost points in Figure 7.1, we can see across manipulation types and datasets that existing methods fail to improve $\mathrm{Acc}_{\mathrm{aff}}$ even when the whole manipulation set is known. UtU fails to unlearn the effects of either of the attacks, as simply unlinking on the forward pass does not sufficiently counteract the influence on neighbors and weights. Both SCRUB and MEGU use a KL Divergence Loss term to keep predictions on the remaining data close to the original model, which could be detrimental when done on unidentified manipulation set entities and other affected neighbors. Even though MEGU and GIF were evaluated on removing adversarial edges and GNNDelete also mentioned incorrect data as one of its key applications, they failed to recover performance when presented with targeted data manipulation. Interestingly, despite extensive hyperparameter searches, **they are beaten by methods with no special graph components**: the best baselines are AC↯DC and naive finetuning (Finetune) on the retain set: both achieve a performance similar to retraining in some of the evaluations.

**2. Corrective unlearning methods need to perform better than Retrain** While both AC↯DC and Finetune match Retrain, all of them still fall far behind the Oracle model's performance in most evaluations, and aren't consistent enough. This scope for improvement motivates the design of our method which performs well across attacks, datasets, and fractions identified for deletion.

**3. *Cognac* beats Retrain, and can sometimes match the Oracle's performance.** We observe that *Cognac* consistently achieves state-of-the-art performance across all datasets and manipulations, convincingly and consistently beating existing graph methods, and often exceeds Retrain. Notably, *Cognac* occasionally even surpasses Oracle, our introduced gold standard, on multiple datasets and identified fractions - despite having access to less data (and unknown manipulated samples) than Oracle. This highlights the effectiveness of our contrastive approach in achieving robust unlearning.

**4. *Cognac* performs strongly even with only $5\%$ of $\mathbf{S_m}$ known.** *Cognac* effectively recovers most of the accuracy on the affected distribution even when only $5\%$ of the manipulated set is known. Notably, it outperforms Retrain in realistic scenarios where $S_m$ is only partially known. We attribute this to *Affected Neighborhood Identification*, which leverages graph structure to infer manipulated nodes and edges beyond those explicitly identified. Additionally, CoGN plays a crucial role by pushing influenced neighbors away from the deletion set and aligning them with their unaffected neighbors, thereby correcting representations even for unknown manipulated samples.

**5. *Cognac* scales well to significantly larger datasets**.We evaluate *Cognac* on OGB-Arxiv, a significantly larger dataset than our other benchmarks with $170,000$ nodes and over $1\mathrm{M}$ edges, and observe that it continues to achieve strong performance. Despite the increased scale, *Cognac* maintains a substantial lead over Retrain across different fractions of $S_m$ (Figure 8.1 (a)), while the baseline methods

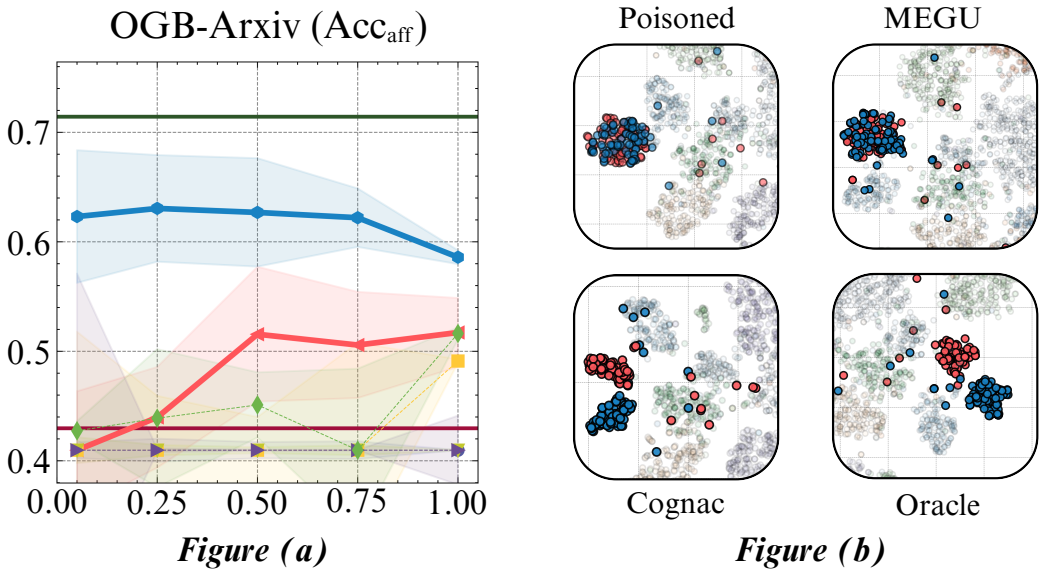


**Figure 8.1 (a) Results on a larger dataset, OGB-Arxiv, and (b) Visualization of hidden layer embeddings after unlearning on *CS* dataset for node unlearning**. (a) On the OGB-Arxiv dataset, *Cognac* outperforms retraining from scratch by more than 10%, while most baselines fail to achieve any performance gains beyond the Original model. (b) The affected distribution embeddings (highlighted by red and blue) are fully entangled in the original trained model while after unlearning with *Cognac* the embeddings are well separated and clustered, matching Oracle.

fail to improve $\mathrm{Acc}_{\mathrm{aff}}$ over the original model. Even as dataset size grows, *Cognac* effectively retains its advantage, demonstrating its scalability and robustness.

Overall, our work makes progress on the problem of corrective unlearning in graph neural networks with remarkably minimal training signal: we achieve strong unlearning with the knowledge of as little as $(5\%)$ of the manipulation set $S_m$. The visualization of the hidden GNN layer embeddings after unlearning (Figure 8.1 (b)) shows that *Cognac*, like the Oracle model, achieves clear clustering of the manipulated class data points, validating our theoretical results shown in Section 6.1.

*Chapter 9*

# Conclusion, Limitations, and Future Work

Our work addresses corrective unlearning for GNNs, focusing on removing the effects of manipulated training data when only a fraction is identified. As our method relies on the homophily assumption and is evaluated primarily on homophilic datasets like previous works [42], it is left for future works to expand this to heterophilic datasets and heterogenous datasets (this includes knowledge graphs). It would also be interesting to explore the applicability of *Cognac* to other domains, such as images.

While our method does not guarantee successful unlearning against arbitrary real-world attacks, our experiments suffice to show that existing unlearning methods struggle even with complete knowledge of manipulations, which is an unrealistic scenario. It would be valuable for future works to incorporate diverse attacks that represent both adversarial situations, and systematic biases, and human errors found in training data to evaluate unlearning.

*Cognac* pushes the frontiers of unlearning by consistently beating retraining-from-scratch, and nearly matching the performance of a strong oracle on unlearning class confusion manipulations with access to as little as $5\%$ of the manipulated data. We hope this sparks interesting future work on developing stronger evaluations and theoretical understanding for graph corrective unlearning in the GNN Robustness and Machine Unlearning community.

*Chapter 10*

# Appendix

# Appendix Overview and Contents

*Appendix A*

# Proof of Theoretical Results

## A.1 Proof of Lemma 6.2

**Theorem A.1.** *Let $G = (V, E)$ be an undirected graph, and let $\mathcal{N}(v)$ denote the 1-hop neighborhood of node $v$. For a node $s \in V$, let $\mathcal{N}^n(s)$ denote the $n$-hop neighborhood of $s$, defined recursively as:*

$$\mathcal{N}^n(s) = \begin{cases} \{s\} & \text{if } n = 0, \\ \bigcup_{v \in \mathcal{N}^{n-1}(s)} \mathcal{N}(v) & \text{if } n > 0. \end{cases}$$

*In an $n$-layer GNN, the representation $z_s$ of node $s$ can affect the representations $z_v$ of nodes $v$ only if $v \in \mathcal{N}^n(s)$. For any $v \notin \mathcal{N}^n(s)$, $z_v$ is independent of $z_s$.*

*Proof.* We prove the claim by induction on the number of layers $l$ in the GNN.

**Base Case ($l = 0$):**

At layer $l = 0$, the representation of a node $v$, $h_v^{(0)}$, is initialized based only on the node's features. Thus, $z_s = h_s^{(0)}$ can only affect $h_s^{(0)}$ itself, and no other node $v \neq s$ is affected.

**Inductive Hypothesis:**

Assume that at layer $l$, the representation $z_s = h_s^{(l)}$ affects only the representations $h_v^{(l)}$ of nodes $v \in \mathcal{N}^l(s)$, the $l$-hop neighborhood of $s$.

**Inductive Step:**

At layer $l+1$, the representation $h_v^{(l+1)}$ of a node $v$ is computed as:

$$h_v^{(l+1)} = \phi\left(h_v^{(l)}, \psi\left(\{h_u^{(l)} : u \in \mathcal{N}(v)\}\right)\right).$$

By the inductive hypothesis, $h_u^{(l)}$ depends only on nodes in $\mathcal{N}^l(u)$. Therefore, $h_v^{(l+1)}$ depends on nodes in:

$$\bigcup_{u \in \mathcal{N}(v)} \mathcal{N}^l(u).$$

Since $\mathcal{N}^{l+1}(v) = \bigcup_{u \in \mathcal{N}(v)} \mathcal{N}^l(u)$, $h_v^{(l+1)}$ depends only on nodes in $\mathcal{N}^{l+1}(v)$.

For $v$ to be influenced by $z_s$, there must exist a path of length at most $l+1$ from $s$ to $v$, i.e., $v \in \mathcal{N}^{l+1}(s)$.

**Conclusion:**

By induction, after $n$ layers, $z_s$ can only affect nodes in $\mathcal{N}^n(s)$. For any $v \notin \mathcal{N}^n(s)$, the representation $z_v$ is independent of $z_s$. □

## A.2 Proof of Theorem 6.3

**Theorem A.2.** *Let $\theta \in \mathbb{R}^d$ parameterize a GNN generating embeddings $z_v, z_p, z_n \in \mathbb{R}^k$ for triplets $(v, p, n)$. Let $\mathcal{L}(\theta) = -\mathbb{E}\left[\log \sigma(z_v^\top z_p) + \log \sigma(-z_v^\top z_n)\right]$ and $S(\theta) = \mathbb{E}\left[z_v^\top z_p - z_v^\top z_n\right]$.*

*Then under the assumptions stated below, gradient descent on $\mathcal{L}(\theta)$ with step size $\eta \leq \frac{1}{L}$ (where $L$ is the Lipschitz constant of $\nabla_\theta \mathcal{L}$) guarantees:*

$$S(\theta_{t+1}) \geq S(\theta_t) \quad \forall t \geq 0,$$

*and at convergence, $S(\theta^*) > S(\theta_0)$.*

**Assumption A.3** (Differentiability)**.** *$z_v^\top z_p$ and $z_v^\top z_n$ are differentiable in $\theta$.*

**Assumption A.4** (Convexity)**.** *$\mathcal{L}(\theta)$ is convex in $\theta$.*

**Assumption A.5** (Bounded Gradients)**.** *$\|\nabla_\theta(z_v^\top z_p)\| \leq G$ and $\|\nabla_\theta(z_v^\top z_n)\| \leq G$ for some $G > 0$.*

*Proof.* Let $a = z_v^\top z_p$ and $b = z_v^\top z_n$. Then:

$$\mathcal{L}(\theta) = -\mathbb{E}\left[\log \sigma(a) + \log \sigma(-b)\right].$$

Using $\frac{d}{dx} \log \sigma(x) = 1 - \sigma(x)$ and $\frac{d}{dx} \log \sigma(-x) = -\sigma(x)$, we compute:

$$\nabla_\theta \mathcal{L} = -\mathbb{E}\left[\underbrace{(1 - \sigma(a))}_{\text{Positive}} \nabla_\theta a - \underbrace{\sigma(b)}_{\text{Positive}} \nabla_\theta b\right].$$

The gradient of $S(\theta)$ is:

$$\nabla_\theta S = \mathbb{E}\left[\nabla_\theta a - \nabla_\theta b\right].$$

The gradient of $\mathcal{L}$ can be rewritten as:

$$\nabla_\theta \mathcal{L} = -\mathbb{E}\left[\sigma(-a)\nabla_\theta a + \sigma(b)\nabla_\theta b\right].$$

Both $\sigma(-a)$ and $\sigma(b)$ are positive for all finite $a, b$. Thus, $\nabla_\theta \mathcal{L}$ is a *negatively weighted combination* of $\nabla_\theta a$ and $\nabla_\theta b$, while $\nabla_\theta S$ is their *unweighted difference*.

Next, we show the monotonic improvement of $S(\theta)$. Consider a gradient descent update:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}.$$

The change in $S(\theta)$ is:

$$\Delta S = S(\theta_{t+1}) - S(\theta_t)$$

To simplify the analysis, we'll use a first-order approximation. Specifically, for small updates, we approximate:

$$S(\theta_{t+1}) \approx S(\theta_t) + \langle \nabla_\theta S, \theta_{t+1} - \theta_t \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. Substituting $\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$, we get:

$$\Delta S \approx -\eta \langle \nabla_\theta S, \nabla_\theta \mathcal{L} \rangle.$$

Substitute $\nabla_\theta S$ and $\nabla_\theta \mathcal{L}$:

$$\langle \nabla_\theta S, \nabla_\theta \mathcal{L} \rangle = \mathbb{E}[(\nabla_\theta a - \nabla_\theta b)^\top ((1-\sigma(a))\nabla_\theta a - \sigma(b)\nabla_\theta b)]$$

$$\Delta S \approx \eta \mathbb{E}[\sigma(-a)\|\nabla_\theta a\|^2 + \sigma(b)\|\nabla_\theta b\|^2 - \sigma(-a)\nabla_\theta a^\top \nabla_\theta b - \sigma(b)\nabla_\theta a^\top \nabla_\theta b].$$

This expression contains both positive and negative terms. However, we now use the fact that the sigmoid function $\sigma(x)$ satisfies $0 < \sigma(x) < 1$ for all real values of $x$, which means:

$(1-\sigma(a)) > 0$ for $a \in \mathbb{R}$,

$\sigma(b) > 0$ for $b \in \mathbb{R}$.

Thus, both terms $(1-\sigma(a))\|\nabla_\theta a\|^2$ and $\sigma(b)\|\nabla_\theta b\|^2$ are positive. On the other hand, the cross-product terms $\nabla_\theta a^\top \nabla_\theta b$ might be negative, but their magnitude is bounded by the gradients $\|\nabla_\theta a\|$ and $\|\nabla_\theta b\|$, which are constrained by the assumption of bounded gradients. Under this assumption, the gradients $\|\nabla_\theta a\|$ and $\|\nabla_\theta b\|$ are bounded by $G$, i.e., $\|\nabla_\theta a\| \leq G$ and $\|\nabla_\theta b\| \leq G$. Therefore, the cross-product terms are also bounded:

$$|\nabla_\theta a^\top \nabla_\theta b| \leq G^2.$$

Now we can write the change in $S(\theta)$ as:

$$\Delta S \approx \eta \mathbb{E}\left[(1-\sigma(a))\|\nabla_\theta a\|^2 + \sigma(b)\|\nabla_\theta b\|^2 - 2G^2\right].$$

For sufficiently small $\eta$, the positive terms $(1-\sigma(a))\|\nabla_\theta a\|^2$ and $\sigma(b)\|\nabla_\theta b\|^2$ will dominate the cross-product terms, ensuring that $\Delta S \geq 0$.

$$\Delta S \geq \eta \mathbb{E}\left[\sigma(-a)\|\nabla_\theta a\|^2 + \sigma(b)\|\nabla_\theta b\|^2 - 2G^2\right].$$

For sufficiently small $\eta$, $\Delta S \geq 0$. Thus, $S(\theta_{t+1}) \geq S(\theta_t)$.

By convexity (Assumption 2), gradient descent converges to a global minimum $\theta^*$ where $\nabla_\theta \mathcal{L}(\theta^*) = 0$. At $\theta^*$:

$$\mathbb{E}\left[\sigma(-a)\nabla_\theta a\right] = \mathbb{E}\left[\sigma(b)\nabla_\theta b\right].$$

This equality holds *only* if $\sigma(-a) \to 0$ and $\sigma(b) \to 0$, which occurs when $a \to \infty$ and $b \to -\infty$. Thus:

$$S(\theta^*) = \mathbb{E}[z_v^\top z_p - z_v^\top z_n] \geq S(\theta_0).$$

Strict inequality $S(\theta^*) > S(\theta_0)$ follows from the monotonic improvement at each step.

□

## A.3 Proof of Theorem 6.1

**Theorem A.6.** *Let $G = (V, E, X)$ be a graph with node set $V$, edge set $E$, and features $X$. Let $C_1, C_2 \subset V$ be two distinct classes with ground-truth labels $y_i \in \{C_1, C_2\}$. Suppose an Interclass Confusion (IC) attack is applied as follows:*

1. *Select subsets $S'C_1 \subset C_1$ and $S'C_2 \subset C_2$, each of size $\frac{n}{2}$, forming $S' = S'C_1 \cup S'C_2$.*

2. *Swap labels of $\alpha =$ fraction of $S'$, creating a confused set $S_f$.*

3. *Train a GNN model $M$ on $(S \setminus S') \cup S_f$.*

*Let $\phi_M(C_1), \phi_M(C_2) \in \mathbb{R}^d$ denote the mean embeddings of $C_1$ and $C_2$, and $\mathcal{D}(\phi_M(C_1), \phi_M(C_2))$ be the Wasserstein-2 distance between their embedding distributions. Assume:*

**Assumption A.7** (Homophily preserving GNNs)**.** *The GNN uses $L$-layers of homophily-preserving message passing (e.g., mean aggregation).*

**Assumption A.8** (Homophily)**.** *The graph exhibits $\eta$-homophily, where intra-class edge density dominates inter-class.*

**Assumption A.9** (Cross-Entropy Loss)**.** *The loss function $\mathcal{L}$ includes cross-entropy and graph smoothness regularization.*

*Then, there exists a degradation term $\Delta > 0$ such that:*

$$\mathbb{E}\left[\mathcal{D}(\phi_M(C_1), \phi_M(C_2))\right] \leq \mathbb{E}\left[\mathcal{D}(\phi_{M_{clean}}(C_1), \phi_{M_{clean}}(C_2))\right] - \Delta,$$

*where $\Delta \propto \alpha \cdot \eta \cdot \left(1 - \frac{1}{L}\right)$.*

*Proof.* **Step 1: Embedding Process Formalization**

The GNN computes node embeddings via $L$-layer message passing. For node $v$, the embedding $\phi^{(l)}(v)$ at layer $l$ is:

$$\phi^{(l)}(v) = \sigma\left(\mathbf{W}^{(l)} \cdot \text{AGG}\left(\{\phi^{(l-1)}(u) \mid u \in \mathcal{N}(v)\}\right)\right),$$

where AGG is a mean aggregator, $\mathbf{W}^{(l)}$ are learnable weights, and $\sigma$ is a nonlinearity. The final embedding $\phi_M(v) = \phi^{(L)}(v)$.

**Step 2: Loss Function and Label Noise**

The training loss $\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda\mathcal{L}_{\text{reg}}$, where:

- $\mathcal{L}_{\text{CE}} = -\frac{1}{|S|}\sum_{v\in S} y_v \log \hat{y}_v$ (cross-entropy).
- $\mathcal{L}_{\text{reg}} = \frac{1}{|E|}\sum_{(u,v)\in E} \|\phi_M(u) - \phi_M(v)\|^2$ (smoothness regularization)

For $S_f$, labels $y_v$ are swapped between $C_1$ and $C_2$. Let $\mathcal{E}_{\text{noise}} = \{v \in S_f \mid y_v \text{ is incorrect}\}$. The corrupted $\mathcal{L}_{\text{CE}}$ forces conflicting gradients for $v \in \mathcal{E}_{\text{noise}}$, pulling $\phi_M(v)$ toward the wrong class centroid.

**Step 3: Bias in Mean Embeddings**

Let $\mu_{C_1}, \mu_{C_2}$ be the mean embeddings of $C_1, C_2$ under $M_{\text{clean}}$. After IC attack, for $v \in \mathcal{E}_{\text{noise}}$:

$$\mathbb{E}[\phi_M(v)] = \alpha\mu_{C_2} + (1-\alpha)\mu_{C_1} \quad (\text{if } v \in C_1).$$

By linearity of expectation, the perturbed mean for $C_1$ becomes:

$$\mu'_{C_1} = \mu_{C_1} - \alpha(\mu_{C_1} - \mu_{C_2}).$$

Similarly, $\mu'_{C_2} = \mu_{C_2} + \alpha(\mu_{C_1} - \mu_{C_2})$. Thus, the distance between means reduces by $2\alpha\|\mu_{C_1} - \mu_{C_2}\|$.

**Step 4: Message-Passing Amplification**

Under $\eta$-homophily (where $\eta$ is the 'extent' of homophily), neighbors of $v \in \mathcal{E}_{\text{noise}}$ are likely in $C_1$. The smoothness term $\mathcal{L}_{\text{reg}}$ propagates the corrupted embedding of $v$ to its neighbors, perturbing their embeddings. After $L$ layers, the influence of $\mathcal{E}_{\text{noise}}$ spreads to $\sim \eta^L|V|$ nodes. This amplifies the mean embedding shift by a factor $\eta\left(1 - \frac{1}{L}\right)$.

**Step 5: Wasserstein Distance Bound**

The Wasserstein-2 distance between $\phi_M(C_1)$ and $\phi_M(C_2)$ is dominated by the mean shift and covariance distortion. Using the Fréchet inequality:

$$\mathcal{D}^2 \leq \|\mu'_{C_1} - \mu'_{C_2}\|^2 + \text{Tr}(\Sigma_{C_1} + \Sigma_{C_2} - 2(\Sigma_{C_1}\Sigma_{C_2})^{1/2}).$$

From Steps 3–4, $\|\mu'_{C_1} - \mu'_{C_2}\|^2 = (1 - 2\alpha\eta(1 - \frac{1}{L}))\|\mu_{C_1} - \mu_{C_2}\|^2$. Thus,

$$\mathbb{E}[\mathcal{D}(\phi_M(C_1), \phi_M(C_2))] \leq \mathbb{E}[\mathcal{D}(\phi_{M_{\text{clean}}}(C_1), \phi_{M_{\text{clean}}}(C_2))] - \Delta,$$

where $\Delta = \alpha\eta\left(1 - \frac{1}{L}\right)\|\mu_{C_1} - \mu_{C_2}\|^2$.

**Conclusion**

The IC attack reduces the separability of $C_1$ and $C_2$ in the embedding space by biasing their mean embeddings toward each other and amplifying this bias via graph convolutions. The degradation $\Delta$ scales with $\alpha$, $\eta$, and the depth $L$, confirming representation entanglement. □

*Appendix B*

# Data Manipulation Statistics

Table B.1 presents the manipulation statistics for all datasets used in our study. We introduce a significant variation in the degree of manipulation to achieve two key objectives: (1) in some datasets, a lower percentage of manipulated nodes or edges is sufficient to induce noticeable performance degradation, while others require more extensive modifications, and (2) we aim to evaluate unlearning performance across a broad spectrum of manipulation scales.

For instance, PubMed undergoes the highest level of manipulation, with nearly 39% of training nodes affected and 33.84% additional edges introduced, whereas datasets like CS and CoraFull experience minimal modifications. Additionally, for OGB-Arxiv, a significantly larger dataset, only 3.14% of training nodes are manipulated. This diverse manipulation strategy ensures a rigorous and comprehensive evaluation of unlearning effectiveness across different graph structures.

**Table B.1 Dataset manipulation statistics**. Statistics of manipulations across datasets. The percentage of nodes and edges added are relative to the existing number of training nodes and the number of edges in the graph, respectively.

| DATASET | TRAINING NODES | NODES MANIPULATED (%) | EDGES ADDED (%) |
|---|---|---|---|
| CORAFULL | 11,280 | 1.45 | 1.20 |
| CORA | 1,491 | 14.88 | 17.26 |
| CITESEER | 1,272 | 14.78 | 20.38 |
| DBLP | 9,714 | 10.50 | 5.78 |
| PUBMED | 11,830 | 38.96 | 33.84 |
| OGB-ARXIV | 90,941 | 3.14 | - |
| CS | 10,999 | 2.25 | 1.83 |
| PHYSICS | 20,695 | 8.17 | 5.04 |
| PHOTOS | 4,492 | 11.40 | 5.04 |
| COMPUTERS | 8,028 | 3.14 | 0.814 |

*Appendix C*

# Formal Description of *Cognac*

In this section, we outline the procedure of our proposed unlearning method, *Cognac*, designed to effectively remove the influence of manipulated data from GNNs. First, the algorithm identifies the nodes affected by the manipulation, as well as their corresponding positive and negative samples, and then alternates between applying CoGN and AC↯DC to unlearn their influence.

The key steps include identifying the affected nodes, performing contrastive learning to re-optimize the embeddings, and minimizing classification loss on the unaffected nodes while maximizing it on the discovered manipulated set ($S_f$). The complete algorithm is detailed in Algorithm 1.

**Algorithm 1** COGNAC

**Require:** GNN $M$, Graph $G = (V, E, X)$, Deletion set $S_f$, Hyperparameters $\Theta$
**Ensure:** Unlearned GNN $M^*$

$S \leftarrow \text{IdentifyAffectedNodes}(M, X, S_f, E, \Theta)$
$P \leftarrow \text{SAMPLEPOSITIVES}(S, E, S_f)$
$N \leftarrow \text{SAMPLENEGATIVES}(S, S_f)$

// Overall unlearning process

**for** outer_epoch $= 1$ **to** $\Theta$.total_epochs **do**

// Contrastive unlearning phase

  **for** epoch $= 1$ **to** $\Theta$.contrast_epochs **do**
    $Z \leftarrow M(X)$
    $L_c \leftarrow \sum_{v \in S} \left( -\log(\sigma(Z_v^\top Z_{P_v})) - \log(\sigma(-Z_v^\top Z_{N_v})) \right)$
    $M \leftarrow \text{OPTIMIZE}(M, L_c)$
  **end for**

// Gradient ascent on $S_f$, and gradient descent on $V \setminus S_f$

  **for** epoch $= 1$ **to** $\Theta$.ascent_descent_epochs **do**
    $L_a \leftarrow -\text{CROSSENTROPY}(M(X)_{S_f}, Y_{S_f})$
    $M \leftarrow \text{OPTIMIZE}(M, L_a)$
    $L_d \leftarrow \text{CROSSENTROPY}(M(X)_{V \setminus S_f}, Y_{V \setminus S_f})$
    $M \leftarrow \text{OPTIMIZE}(M, L_d)$
  **end for**
**end for**

**return** $M$

**function** IDENTIFYAFFECTEDNODES($M, X, S_f, E, \Theta$)
  $X' \leftarrow \text{INVERTFEATURES}(X, S_f, E)$
  $\Delta \leftarrow |M(X') - M(X)|$
  **return** $\text{TOPK}(\Delta, \Theta.k)$
**end function**

*Appendix D*

# Results Showing The Breadth of Applicability of *Cognac*

## D.1 Results on GAT

To provide a comprehensive comparison between *Cognac* and other methods, we provide results on commonly used GNN backbone architectures - GCN and GAT.

**Graph Convolutional Network (GCN)** is a method for semi-supervised classification of graph-structured data. It employs an efficient layer-wise propagation rule derived from a first-order approximation of spectral convolutions on graphs.

**Graph Attention Network (GAT)** employs computationally efficient masked self-attention layers that assign varying importance to neighborhood nodes without needing the complete graph structure upfront, thereby overcoming many theoretical limitations of earlier spectral-based methods.

Figure D.1 shows that *Cognac* also performs competitively with a GAT backbone. When 5% of $S_m$ is known, SCRUB performs similarly to *Cognac*, within the standard deviation. For higher fractions, we achieve greater $\mathrm{Acc}_{\mathrm{aff}}$ than the benchmark graph unlearning methods with large margins, often beating the performance of retraining the GNN from scratch.

These results indicate that benchmark graph unlearning methods used for comparison cannot recover from the impact of the label flip poison. In contrast, our method is much closer to Oracle's performance.

## D.2 Additional Experiments on Large $S_f$

To stress-test *Cognac*'s performance - as methods could potentially degrade as the size of $S_f$ grows - we conduct an experiment where we choose a significant fraction of the training nodes of the Amazon,

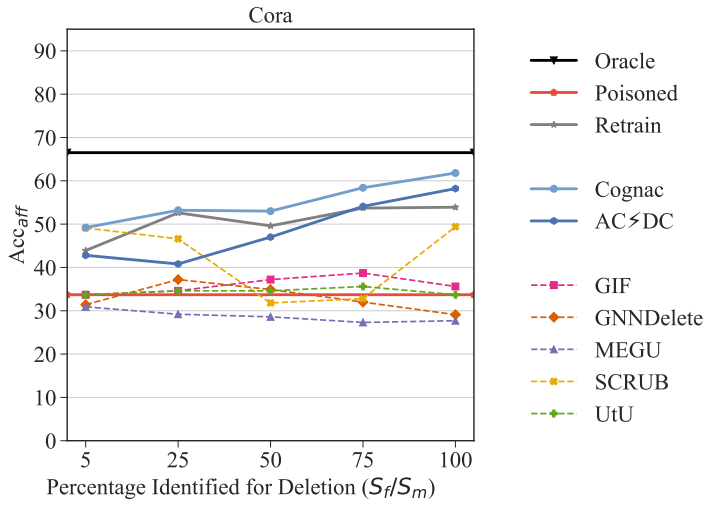


**Figure D.1 Node Unlearning results for Cora with GAT backbone.** Comparison of $\text{Acc}_{\text{aff}}$ for the unlearning methods on GAT trained on Cora for different values of $(S_f/S_m)$. We see that *Cognac* outperforms all baselines.

DBLP, and Physics datasets, to be attacked (by the binary label flip attack) and marked for deletion. The method performs competitively even at this large deletion size. Table D.1 demonstrates these results.

**Table D.1 Evaluation on larger sizes of $S_f$.** *Cognac* performs well across datasets (Amazon, DBLP, Physics), even when a significant fraction of the total training nodes are present in $S_f$.

| METHOD | AMAZON (25%) | | DBLP (29.4%) | | PHYSICS (14.8%) | |
|---|---|---|---|---|---|---|
| | $\text{Acc}_{\text{rem}}$ | $\text{Acc}_{\text{aff}}$ | $\text{Acc}_{\text{rem}}$ | $\text{Acc}_{\text{aff}}$ | $\text{Acc}_{\text{rem}}$ | $\text{Acc}_{\text{aff}}$ |
| ORACLE | 92.9 | 95.6 | 72.9 | 86.0 | 95.2 | 95.1 |
| ORIGINAL | 92.5 | 49.0 | 74.9 | 57.9 | 58.9 | 95.4 |
| *Cognac* | **92.4** | **83.7** | **82.3** | 81.7 | **90.7** | **95.0** |
| GNNDELETE | 27.7 | 49.7 | 45.0 | 49.6 | 1.4 | 37.3 |
| SCRUB | 92.3 | 72.6 | 77.5 | **82.1** | 77.9 | 94.9 |

*Appendix E*

# Convergence and Ablations of *Cognac*

## E.1 Convergence

We now discuss the convergence properties of *Cognac*. Plots in Figure E.1 describe the losses of each of the components of our method (contrastive, ascent, descent) after the last epoch of every *step*, the meaning of which should be clear from Algorithm 1: Line 4 (which we denote as `num_steps`). The loss plots are constructed over the best hyperparameters, and we would likely not see such convergence trends with sub-optimal hyperparameters, which may provide insights to improve performance when it's used in other settings as well.

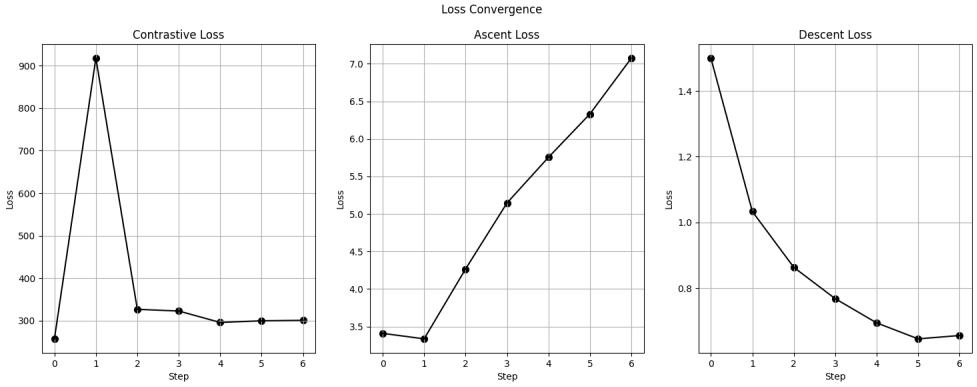


**Figure E.1 Convergence of the losses across unlearning steps**. The ascent loss on $S_m$ continually increases as expected, the descent loss on $S \setminus S_m$ converges, and the contrastive loss exhibits a low plateau after an initial overshoot, implying it may have learned discriminative features.

## E.2 Analysis of method used to find affected neighbors

Our strategy to find affected neighbors is likely not perfect for finding the most affected nodes and more sophisticated influence functions such as the one presented in [13] could be used to potentially improve performance.

Still, we note that it achieves a $5\%$ higher $\mathrm{Acc}_{\mathrm{aff}}$ than while choosing random $k\%$ nodes in the $n$-hop neighborhood (where $n$ is the number of layers of message passing) while being cheap to compute: we only require a single forward pass over the model with the inverted features.

Interestingly, Figure E.2 also shows that even if the GNN is not well-trained, if we choose the top $k\%$ affected nodes, the unlearning performance does not change much, while still being noticeably better than when we use a random $k\%$ of the neighbors.

Figure E.3 (left) shows that there are no noticeable changes in taking a smaller or larger $k\%$. However, removing this step entirely ($k = 0\%$) results in worse performance, suggesting performing contrastive unlearning on even a small $k\%$ is significant. Additionally, by keeping this percentage small, we ensure computational efficiency without diminishing performance (Figure E.3 (right)), which is essential for unlearning methods.

## E.3 Importance of Decoupled Optimizers for Ascent and Descent

While methods incorporating gradient ascent and gradient descent have been studied in prior unlearning work like SCRUB, we find that combining these objectives into one optimizer performs suboptimally in our experiments. Furthermore, we notice this can be fixed with a simple trick: using different learning rates for the ascent and descent steps. The final version of *Cognac* uses two separate optimizers: two learning rates and different instances of Adam [34] instead of coupling the optimization of both steps.

To validate the necessity of separate optimizers and alternating ascent-descent we perform two ablations: (1) using a single optimizer with dual learning rates and (2) using a combined loss function instead of alternating between ascent and descent steps.

In the first ablation, since PyTorch does not provide a straightforward way to implement different LRs for the same set of parameters, we multiply the LR during ascent with a tunable constant. This is functionally equivalent to having 2 LRs as we use a linear decay schedule, which despite its simplicity was recently shown to outperform more complicated schedulers [18].

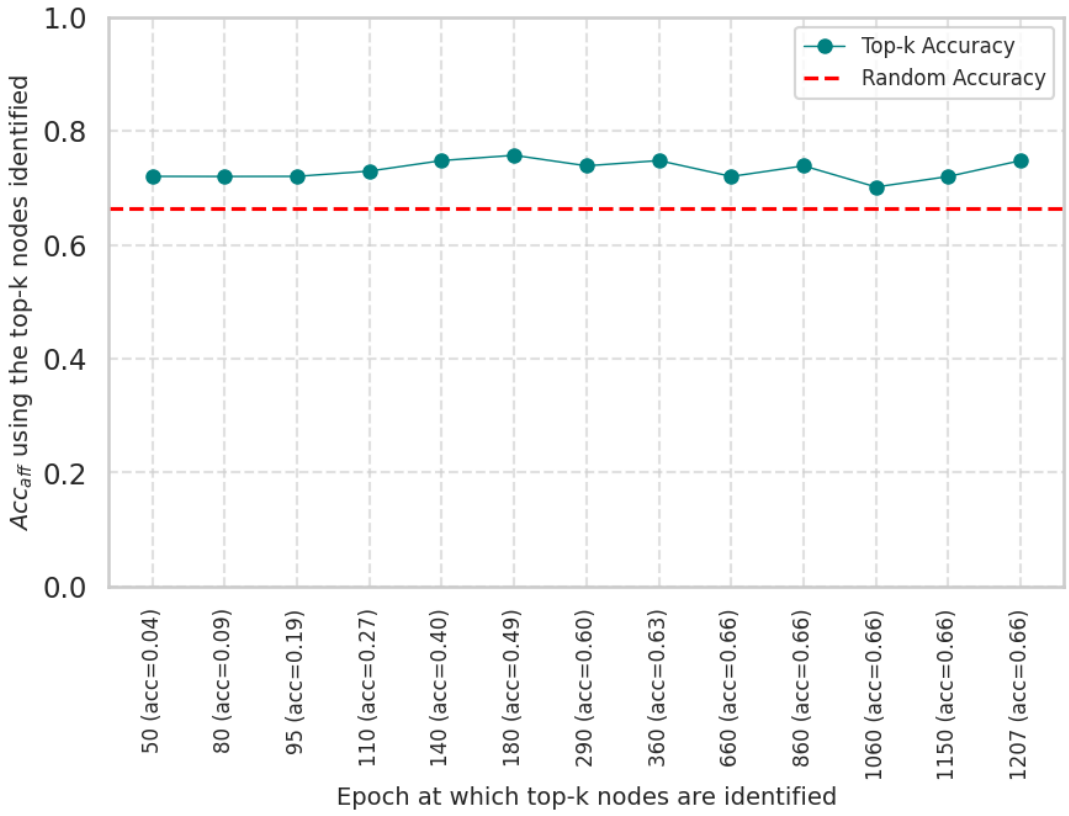


**Figure E.2 Effect of how well-trained the GNN is on top-$k\%$ affected neighbour identification.** Here $k = 4$. The x-axis represents the epoch at which we used GNN representations to identify the most affected neighbors for *Cognac*. The y-axis reports the unlearning performance after contrastive unlearning on these identified nodes using the final model. The red line contains performance after picking a random subset from the $n$-hop neighborhood. Affected neighborhood identification using top-$k\%$ logit change is more effective even with an extremely undertrained GNN.

Adding another hyperparameter for the ascent learning rate is necessary as the ascent is not always needed when the original training data labels are correct. For example, we do find that the ascent learning rate is set to nearly zero during automatic hyperparameter selection for our edge unlearning evaluation, where labels are not manipulated.

In Table E.1, we report results on $25\%$ and $100\%$ of the manipulated nodes identified for Cora. We find AC↯DC leads to almost $20\%$ better $\mathrm{Acc}_{\mathrm{aff}}$ than using a single optimizer with alternating ascent descent, and $5 - 10\%$ better $\mathrm{Acc}_{\mathrm{aff}}$ than using a combined loss function. This justifies our contribution of decoupling optimizers in AC↯DC.

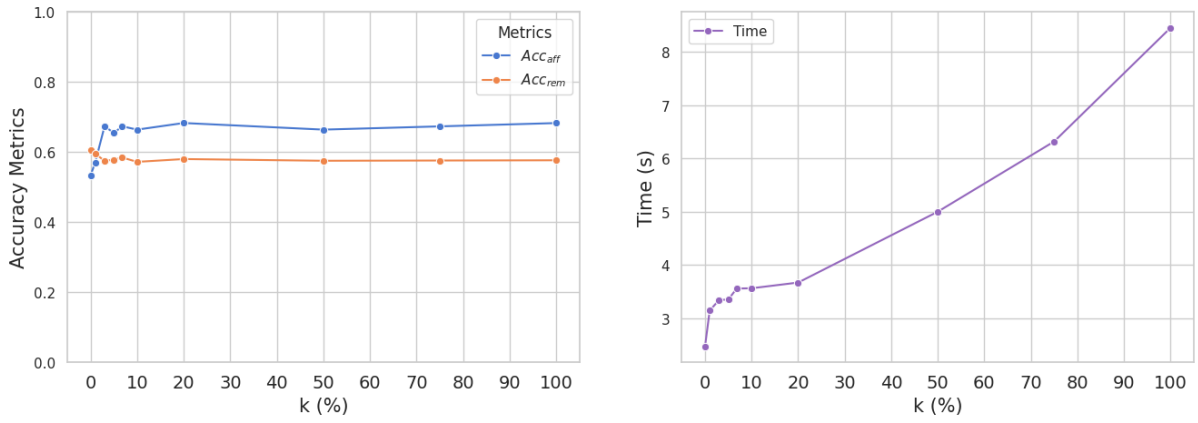


**Figure E.3 Effect of $k$ on unlearning ($\mathrm{Acc}_{\mathrm{aff}}$), utility ($\mathrm{Acc}_{\mathrm{rem}}$) and efficiency when identifying top-$k\%$ affected nodes for contrastive unlearning.** (Left) *Cognac* effectiveness sharply improves beyond $k = 0\%$, suggesting that performing contrastive unlearning on even a small percentage of nodes ($k$) significantly enhances the algorithm's effectiveness. However, using higher values of $k$ yields similar performance with the added downside of increasing computational time (Right).

**Table E.1 Ablations of AC↯DC at $(S_f/S_m)$ = 0.25 and 1.00 on Cora node unlearning**. We ablate the AC↯DC part of our method, even when the losses are combined in a single loss term.

| **Method** | **0.25** | | **1.00** | |
|---|---|---|---|---|
| | $\mathrm{Acc}_{\mathrm{aff}}$ | $\mathrm{Acc}_{\mathrm{rem}}$ | $\mathrm{Acc}_{\mathrm{aff}}$ | $\mathrm{Acc}_{\mathrm{rem}}$ |
| 1 Optimizer 1 LR Alternating Ascent Descent | 38.3 | 56.0 | 29.0 | 56.2 |
| 1 Optimizer 2 LRs Combined Loss | 47.5 | 56.7 | 49.6 | 55.1 |
| AC↯DC | 58.8 | 57.4 | 54.3 | 56.1 |

## E.4 Why does $\mathrm{Acc}_{\mathrm{aff}}$ sometimes reduce as identified manipulated entities increase?

We find an interesting trend that sometimes, as more of the manipulation set $(S_m)$ is known and used as the deletion set $(S_f)$ (going left to right in Figure 7.1), $\mathrm{Acc}_{\mathrm{aff}}$ reduces. This can seem counterintuitive, as one would expect the accuracy of affected classes to improve as more samples are used for unlearning.

We hypothesize that unlearning a larger fraction of the manipulation set reduces $\mathrm{Acc}_{\mathrm{aff}}$ due to two factors that adversely affect the neighborhoods of the nodes removed, which typically have other nodes of the affected classes due to homophily. First, in the case of label manipulation, when we model it as node unlearning for consistency with prior work, we lose correct information about the graph structure. Second, when modifying the graph structure, i.e., removing some edges or nodes changes the feature distribution of their neighboring nodes after the message passes, making it out of distribution for the learned GNN layers. The same rationale is why the test nodes are kept in the graph structure (without optimizing the task loss for them) during training [35].

We investigate this by adding an ablation where in the unlearning of the label manipulation, instead of unlearning the whole node, we keep the structure, i.e., the node and connected edges, but unlearn the features and labels.

**Table E.2 Ablating node unlearning performance on label manipulation with and without unlinking.** We report the accuracy on the affected classes $\mathrm{Acc}_{\mathrm{aff}}$ for unlearning the label manipulation on Cora both when the full and a subset (25%) of the manipulated set is used for deletion. We find not removing the structural information leads to a significant improvement in $\mathrm{Acc}_{\mathrm{aff}}$, especially when more entities are deleted. This illustrates that the unlearning methods can achieve improved performance when precise information about the manipulated data is available.

| **Method** | **0.25** | | **1.00** | |
|---|---|---|---|---|
| | **Linked** | **Unlinked** | **Linked** | **Unlinked** |
| Oracle | $73.0_{\pm 0.0}$ | $73.0_{\pm 0.0}$ | $73.0_{\pm 0.0}$ | $73.0_{\pm 0.0}$ |
| Original | $42.0_{\pm 0.0}$ | $42.0_{\pm 0.0}$ | $42.0_{\pm 0.0}$ | $42.0_{\pm 0.0}$ |
| *Cognac* | $\mathbf{64.8_{\pm 0.9}}$ | $\mathbf{67.8_{\pm 3.2}}$ | $\mathbf{77.2_{\pm 1.0}}$ | $\mathbf{69.3_{\pm 1.3}}$ |
| GNNDelete | $35.2_{\pm 2.5}$ | $50.2_{\pm 1.9}$ | $21.9_{\pm 4.5}$ | $30.2_{\pm 5.3}$ |
| MEGU | $40.8_{\pm 1.6}$ | $33.4_{\pm 0.4}$ | $41.2_{\pm 1.5}$ | $32.1_{\pm 1.3}$ |
| SCRUB | $45.7_{\pm 0.0}$ | $60.7_{\pm 0.0}$ | $41.1_{\pm 0.0}$ | $29.0_{\pm 0.0}$ |
| AC⅟2DC | $61.7_{\pm 0.0}$ | $58.8_{\pm 0.0}$ | $63.7_{\pm 0.0}$ | $54.3_{\pm 0.0}$ |

As observed in Table E.2, retaining the node structure leads to large improvements in $\mathrm{Acc}_{\mathrm{aff}}$ when the deletion set is larger (the full set of manipulated entities), while not benefiting much when the deletion set is smaller.

In the full manipulation set deletion setting, *Cognac* even slightly outperforms Oracle. This highlights how—unlike conventional node unlearning in graphs—removing the nodes is not always the best

way to unlearn manipulations. They can simply be moved from the train set to the test set to still partake in message passing, so the task loss is not optimized over wrong labels.

*Appendix F*

# Detailed report of the experimental setup

## F.1 Hyperparameter Tuning

We perform extensive hyperparameter tuning for all unlearning methods using Optuna [1] with a TPESampler (Tree-structured Parzen Estimator) Algorithm. We ensure the hyperparameter ranges searched include any values specified by the authors that proposed the methods.

The optimization target is an average of $\mathrm{Acc}_{\mathrm{aff}}$ and $\mathrm{Acc}_{\mathrm{rem}}$, computed on the validation set. We report averaged results across five seeds. Method-specific hyperparameter ranges and scatter plots across hyperparameters for each method are provided in Appendix F.1.

We perform hyperparameter tuning for each combination for attack, dataset, unlearning method, and the identified fraction of deletion set ($S_f$). The optimization target is an average of $\mathrm{Acc}_{\mathrm{aff}}$ and $\mathrm{Acc}_{\mathrm{rem}}$, computed on the validation set. For each setting, we run 100 trials with hyperparameters selected using the TPESampler (Tree-structured Parzen Estimator) algorithm.

In Figures F.1, F.2 and F.3, we report $\mathrm{Acc}_{\mathrm{aff}}$ and $\mathrm{Acc}_{\mathrm{rem}}$ scores for each hyperparameter tuning trial. Across hyperparameter runs, existing graph-based unlearning methods, barring MEGU, vary drastically across different sets of hyperparameters. On the other hand, our proposed method *Cognac* and its ablations show consistently high scores across hyperparameters, showcasing *Cognac*'s robustness to hyperparameter tuning.

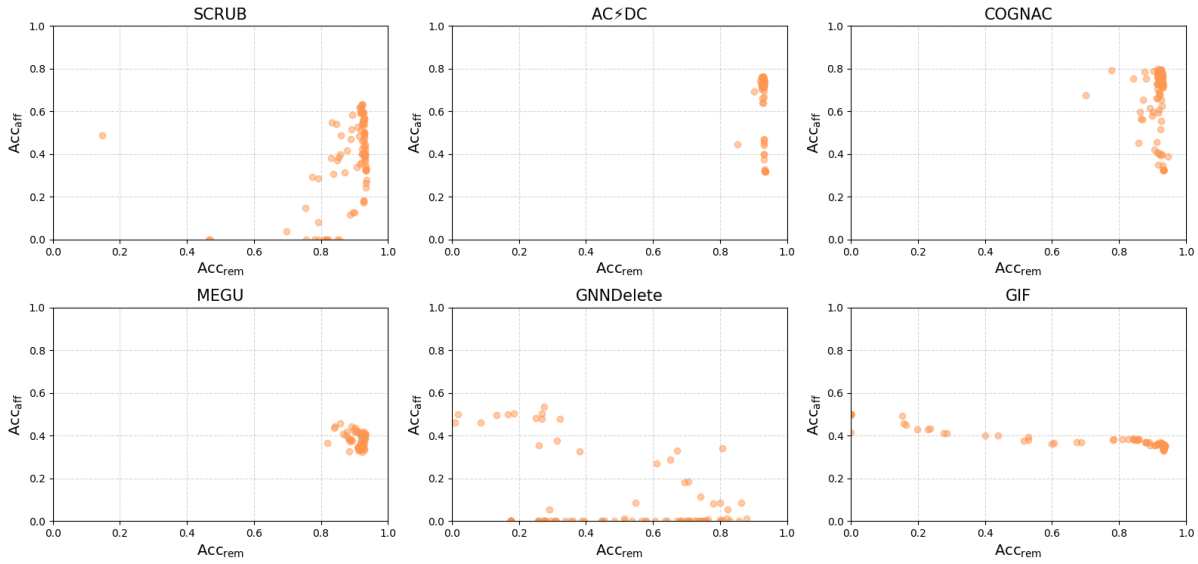


**Figure F.1 Hyperparameter runs for $S_f = 1.0$ on Amazon**. Scores of various hyperparameter trial runs. The best hyperparameters are selected according to the run achieving the best value for the average of $\mathrm{Acc}_{\mathrm{rem}}$ and $\mathrm{Acc}_{\mathrm{aff}}$.

## F.2 Unlearning Times

To ensure the practicality of inexact unlearning methods, they must achieve greater efficiency than retraining from scratch. As illustrated in Figure F.4, *Cognac* demonstrates competitive efficiency compared to alternative methods, offering substantial speedups over retraining from scratch.

**Measuring Unlearning Time.**

To simplify comparisons to just two axes, $\mathrm{Acc}_{\mathrm{aff}}$, and $\mathrm{Acc}_{\mathrm{rem}}$, we fix a maximum cutoff of time an unlearning method can take, as motivated by [46]. We chose this cutoff as 25% of the original model training time. We pick the best model checkpoint during training for each method, which could be achieved earlier than this.

Average run times for each method reported in Figure F.4 under Appendix F.2 show that *Cognac*'s efficiency is comparable to or better than baselines. All experiments were run on a machine with Intel Xeon CPUs and two dedicated RTX 5000 GPUs.

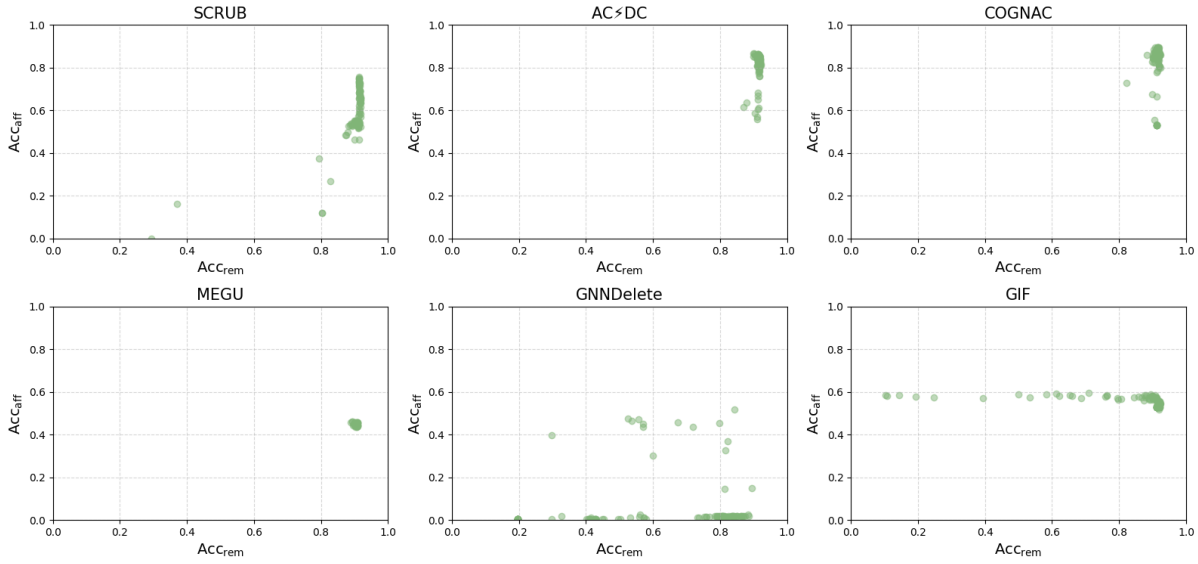


**Figure F.2 Hyperparameter runs for $S_f = 1.0$ on CS**. Scores of various hyperparameter trial runs. The best hyperparameters are selected according to the run achieving the best value for the average of $\mathrm{Acc}_{\mathrm{rem}}$ and $\mathrm{Acc}_{\mathrm{aff}}$.

**Table F.1 Results for unlearning the trigger-based feature manipulation.** *Cognac* performs strongly across datasets, maintaining a high $\mathrm{Acc}_{\mathrm{aff}}$ and $\mathrm{Acc}_{\mathrm{rem}}$. Retrain fails to recover accuracy on the victim class for the *Cora* dataset, while SCRUB fails to do so for both *Cora* and *CS*. GNNDelete and MEGU, the graph unlearning baselines, fail to remove the poison.

| METHOD | PHOTOS | | CORA | | CS | |
|---|---|---|---|---|---|---|
| | $\mathrm{Acc}_{\mathrm{rem}}$ | $\mathrm{Acc}_{\mathrm{aff}}$ | $\mathrm{Acc}_{\mathrm{rem}}$ | $\mathrm{Acc}_{\mathrm{aff}}$ | $\mathrm{Acc}_{\mathrm{rem}}$ | $\mathrm{Acc}_{\mathrm{aff}}$ |
| ORIGINAL | $95.0 \pm 0.0$ | $33.9 \pm 0.0$ | $67.4 \pm 0.0$ | $25.6 \pm 0.0$ | $89.3 \pm 0.0$ | $0.0 \pm 0.0$ |
| RETRAIN | $94.1 \pm 0.5$ | $92.0 \pm 1.7$ | $\mathbf{68.3 \pm 0.4}$ | $46.9 \pm 6.8$ | $93.0 \pm 0.2$ | $92.8 \pm 2.3$ |
| GNNDELETE | $17.8 \pm 0.0$ | $0.0 \pm 0.0$ | $65.7 \pm 0.2$ | $40.8 \pm 9.1$ | $68.0 \pm 0.0$ | $0.0 \pm 0.0$ |
| MEGU | $81.5 \pm 8.7$ | $17.5 \pm 23.1$ | $61.5 \pm 0.3$ | $1.0 \pm 1.0$ | $88.7 \pm 0.1$ | $0.0 \pm 0.0$ |
| SCRUB | $93.8 \pm 0.0$ | $\mathbf{92.7 \pm 0.0}$ | $68.0 \pm 0.0$ | $64.1 \pm 0.0$ | $92.7 \pm 0.1$ | $72.1 \pm 21.8$ |
| *Cognac* | $\mathbf{94.6 \pm 0.0}$ | $91.1 \pm 0.0$ | $67.3 \pm 0.0$ | $\mathbf{78.2 \pm 0.0}$ | $\mathbf{93.3 \pm 0.0}$ | $\mathbf{93.6 \pm 0.6}$ |

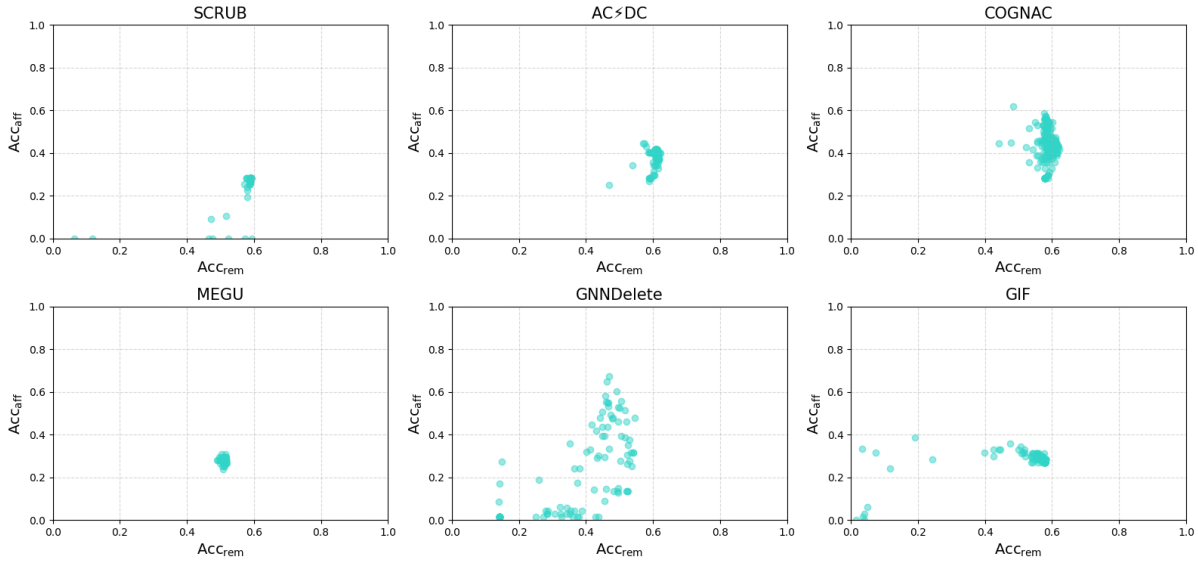


**Figure F.3 Hyperparameter runs for $S_f = 1.0$ on Cora**. Scores of various hyperparameter trial runs. The best hyperparameters are selected according to the run achieving the best value for the average of $\mathrm{Acc}_{\mathrm{rem}}$ and $\mathrm{Acc}_{\mathrm{aff}}$.

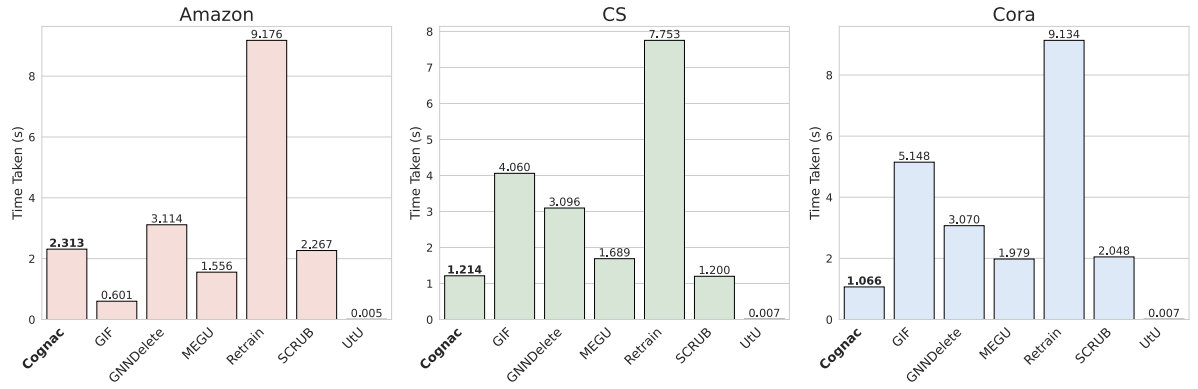


**Figure F.4 Time taken to unlearn.** We report the time taken by each unlearning method for all datasets, in the setting where all manipulated samples are known for deletion. *Cognac* provides significant speedups over Retrain and has performance similar to other unlearning methods.

**Table F.2 Evaluating *Cognac* with a different strategy to identify affected nodes on *Cora*.** We ablate *Cognac* by replacing Affected Node Identification (Section 3.1.1) with the sampling strategy introduced by MEGU. We find that our strategy outperforms the baseline, while also being 8x faster. Both variants are hyperparameter tuned for 100 runs.

| METHOD | $\mathrm{Acc}_{\mathrm{rem}}$ | $\mathrm{Acc}_{\mathrm{aff}}$ | SAMPLING TIME (3160 NODES) |
|---|---|---|---|
| ORIGINAL | $61.4 \pm 0.00$ | $40.0 \pm 0.00$ | - |
| ORACLE | $58.4 \pm 0.00$ | $69.2 \pm 0.00$ | - |
| *Cognac* (MEGU'S SAMPLING STRATEGY) | $54.8 \pm 0.00$ | $48.7 \pm 0.00$ | $0.432\ s$ |
| *Cognac* (OUR SAMPLING STRATEGY) | $\mathbf{56.6 \pm 0.00}$ | $\mathbf{75.5 \pm 0.04}$ | $\mathbf{0.054\ s}$ |

**Table F.3 Ablating both components of *Cognac* across datasets.** CoGN alone does not improve $\mathrm{Acc}_{\mathrm{aff}}$ over the original model, showing that while it effectively moves affected nodes, it lacks correction from labels. Conversely, AC↯DC achieves better performance than CoGN, but is still far from matching the Oracle, confirming that the components are complementary. AC/DC weakens incorrect learning signals and preserves task-relevant representations, while CoGN steers affected nodes away from manipulated ones.

| METHOD | PHOTOS | | CORA | | CS | |
|---|---|---|---|---|---|---|
| | $\mathrm{Acc}_{\mathrm{aff}}$ | $\mathrm{Acc}_{\mathrm{rem}}$ | $\mathrm{Acc}_{\mathrm{aff}}$ | $\mathrm{Acc}_{\mathrm{rem}}$ | $\mathrm{Acc}_{\mathrm{aff}}$ | $\mathrm{Acc}_{\mathrm{rem}}$ |
| ORIGINAL | 35.7 | 94.7 | 51.4 | 60.6 | 44.3 | 89.8 |
| ORACLE | 93.5 | 94.5 | 71.1 | 60.7 | 90.1 | 90.0 |
| COGN | 34.7 | 90.7 | 42.1 | 60.5 | 47.2 | **89.8** |
| AC↯DC | 76.2 | **94.8** | 59.9 | **61.3** | 67.2 | 86.6 |
| *Cognac* | **82.9** | 92.1 | **75.5** | 56.6 | **79.3** | 82.3 |

# Bibliography


[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[2] Arvindh Arun, Aakash Aanegola, Amul Agrawal, Ramasuri Narayanam, and Ponnurangam Kumaraguru. CAFIN: Centrality Aware Fairness inducing IN-processing for unsupervised representation learning on graphs. In *Proceedings of the 26th European Conference on Artificial Intelligence*, 2023.

[3] Kenneth Atz, Leandro Cotos, Clemens Isert, Maria Håkansson, Dorota Focht, Mattis Hilleke, David F. Nippa, Michael Iff, Jann Ledergerber, Carl C. G. Schiebroek, Valentina Romeo, Jan A. Hiss, Daniel Merk, Petra Schneider, Bernd Kuhn, Uwe Grether, and Gisbert Schneider. Prospective de novo drug design with deep interactome learning. *Nature Communications*, 15(1), April 2024.

[4] Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. LEACE: Perfect linear concept erasure in closed form. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[5] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[6] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*, pages 695–704. PMLR, 2019.

[7] Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *IEEE S&P*, 2021.

[9] Carla E Brodley and Mark A Friedl. Identifying mislabeled training data. *Journal of artificial intelligence research*, 11:131–167, 1999.

[10] Payal Chandak, Kexin Huang, and Marinka Zitnik. Building a knowledge graph to enable precision medicine. *Scientific Data*, 10(1), February 2023.

[11] Chong Chen, Fei Sun, Min Zhang, and Bolin Ding. Recommendation unlearning. In *Proceedings of the ACM Web Conference 2022*, pages 2768–2777, 2022.

[12] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph unlearning. In *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security*, pages 499–513, 2022.

[13] Zizhang Chen, Peizhao Li, Hongfu Liu, and Pengyu Hong. Characterizing the influence of graph elements. In *The Eleventh International Conference on Learning Representations*, 2023.

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

[15] Eli Chien, Chao Pan, and Olgica Milenkovic. Efficient model updates for approximate unlearning of graph-structured data. In *The Eleventh International Conference on Learning Representations*, 2022.

[16] 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*, pages 6674–6703. PMLR, 2023.

[17] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1115–1124. PMLR, 10–15 Jul 2018.

[18] Aaron Defazio, Ashok Cutkosky, Harsh Mehta, and Konstantin Mishchenko. When, why and how much? adaptive learning rate scheduling by refinement, 2023.

[19] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[20] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021.

[21] Shaohua Fan, Xiao Wang, Yanhu Mo, Chuan Shi, and Jian Tang. Debiasing graph neural networks via learning disentangled causal substructure. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[22] Simon Geisler, Tobias Schmidt, Hakan Şirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. Robustness of graph neural networks at scale. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2024. Curran Associates Inc.

[23] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[24] Shashwat Goel, Ameya Prabhu, Amartya Sanyal, Ser-Nam Lim, Philip Torr, and Ponnurangam Kumaraguru. Towards adversarial evaluations for inexact machine unlearning. *arXiv preprint arXiv:2201.06640*, 2022.

[25] Shashwat Goel, Ameya Prabhu, Philip Torr, Ponnurangam Kumaraguru, and Amartya Sanyal. Corrective machine unlearning. *Data-centric Machine Learning Research (DMLR) Workshop, ICLR*, 2024.

[26] Lukas Gosch, Daniel Sturm, Simon Geisler, and Stephan Günnemann. Revisiting robustness in graph machine learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[27] Stephan Günnemann. Graph neural networks: Adversarial robustness. *Graph neural networks: foundations, frontiers, and applications*, pages 149–176, 2022.

[28] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*, 2019.

[29] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[30] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[31] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022.

[32] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 66–74, 2020.

[33] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman,

Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, July 2021.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[35] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[36] Nikola H Konstantinov and Christoph Lampert. Fairness-aware pac learning from corrupted data. *JMLR*, 2022.

[37] Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. Towards unbounded machine unlearning. *Advances in Neural Information Processing Systems*, 36, 2024.

[38] Meghdad Kurmanji, Peter Triantafillou, and Eleni Triantafillou. Towards unbounded machine unlearning. *NeurIPS*, 2023.

[39] Chungpa Lee, Joonhwan Chang, and Jy-yong Sohn. Analysis of using sigmoid loss for contrastive learning. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 1747–1755. PMLR, 02–04 May 2024.

[40] Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, et al. The wmdp benchmark: Measuring and reducing malicious use with unlearning. *arXiv preprint arXiv:2403.03218*, 2024.

[41] Wenjie Li, Jiawei Li, Christian Schroeder de Witt, Ameya Prabhu, and Amartya Sanyal. Delta-influence: Unlearning poisons via influence functions, 2024.

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

[43] Jing Lin, Long Dang, Mohamed Rahouti, and Kaiqi Xiong. Ml attack models: Adversarial attacks and data poisoning attacks. *arXiv preprint arXiv:2112.02797*, 2021.

[44] Vijay Lingam, Mohammad Sadegh Akhondzadeh, and Aleksandar Bojchevski. Rethinking label poisoning for GNNs: Pitfalls and attacks. In *The Twelfth International Conference on Learning Representations*, 2024.

[45] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*, 2022.

[46] Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary Chase Lipton, and J Zico Kolter. TOFU: A task of fictitious unlearning for LLMs. In *First Conference on Language Modeling*, 2024.

[47] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph foundation models are already here. In *Forty-first International Conference on Machine Learning*, 2024.

[48] Jesus Arjona Martinez, Olmo Cerri, Maurizio Pierini, Maria Spiropulu, and Jean-Roch Vlimant. Pileup mitigation at the large hadron collider with graph neural networks, 2019.

[49] Amil Merchant, Simon Batzner, Samuel S. Schoenholz, Muratahan Aykol, Gowoon Cheon, and Ekin Dogus Cubuk. Scaling deep learning for materials discovery. *Nature*, 624(7990):80–85, November 2023.

[50] V. Mikuni and F. Canelli. Abcnet: an attention-based method for particle tagging. *The European Physical Journal Plus*, 135(6), June 2020.

[51] Rajat Mishra and S. Shridevi. Knowledge graph driven medicine recommendation system using graph neural networks on longitudinal medical records. *Scientific Reports*, 14(1), October 2024.

[52] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. In *NeurIPS*, 2021.

[53] 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. IEEE, 2021.

[54] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[55] Daniel Paleka and Amartya Sanyal. A law of adversarial risk, interpolation, and label noise. In *ICLR*, 2023.

[56] Martin Pawelczyk, Jimmy Z Di, Yiwei Lu, Gautam Kamath, Ayush Sekhari, and Seth Neel. Machine unlearning fails to remove data poisoning attacks. *arXiv preprint arXiv:2406.17216*, 2024.

[57] Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, and Pascal Friederich. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1), November 2022.

[58] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[59] Anwar Said, Tyler Derr, Mudassir Shabbir, Waseem Abbas, and Xenofon Koutsoukos. A survey of graph unlearning. *arXiv preprint arXiv:2310.02164*, 2023.

[60] Amartya Sanyal, Puneet K. Dokania, Varun Kanade, and Philip Torr. How benign is benign overfitting? In *ICLR*, 2021.

[61] Amartya Sanyal, Yaxi Hu, and Fanny Yang. How unfair is private learning? In *Uncertainty in Artificial Intelligence*, 2022.

[62] Stefan Schoepf, Jack Foster, and Alexandra Brintrup. Potion: Towards poison unlearning. *arXiv preprint arXiv:2406.09173*, 2024.

[63] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, January 2021.

[64] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. Node injection attacks on graphs via reinforcement learning. *arXiv preprint arXiv:1909.06543*, 2019.

[65] 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 on Web Conference 2024*, pages 489–492, 2024.

[66] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[67] 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.

[68] Kun Wang, Guibin Zhang, Xinnan Zhang, Junfeng Fang, Xun Wu, Guohao Li, Shirui Pan, Wei Huang, and Yuxuan Liang. The heterophilic snowflake hypothesis: Training and empowering gnns for heterophilic graphs, 2024.

[69] Nicholas C. Wang, Douglas C. Noll, Ashok Srinivasan, Johann Gagnon-Bartsch, Michelle M. Kim, and Arvind Rao. Simulated mri artifacts: Testing machine learning failure modes. *BME Frontiers*, 2022, January 2022.

[70] Alexander Warnecke, Lukas Pirch, Christian Wressnegger, and Konrad Rieck. Machine unlearning of features and labels. In *Proc. of the 30th Network and Distributed System Security (NDSS)*, 2023.

[71] Baoning Wu and Brian D. Davison. Identifying link farm spam pages. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, WWW '05, page 820–829, New York, NY, USA, 2005. Association for Computing Machinery.

[72] 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 2023*, pages 651–661, 2023.

[73] 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*, pages 2606–2617, 2023.

[74] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Comput. Surv.*, 55(5), December 2022.

[75] Zhaohan Xi, Tianyu Du, Changjiang Li, Ren Pang, Shouling Ji, Xiapu Luo, Xusheng Xiao, Fenglong Ma, and Ting Wang. On the security risks of knowledge graph reasoning. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA, 2023. USENIX Association.

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

[77] Yuanshun Yao, Xiaojun Xu, and Yang Liu. Large language model unlearning. *arXiv preprint arXiv:2310.10683*, 2023.

[78] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.

[79] Ying Yu, Yuanbang Mai, Yuanting Zheng, and Leming Shi. Assessing and mitigating batch effects in large-scale omics studies. *Genome Biology*, 25(1), October 2024.

[80] Xiangchi Yuan, Chunhui Zhang, Yijun Tian, Yanfang Ye, and Chuxu Zhang. Mitigating emergent robustness degradation while scaling graph learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[81] Yang Yuan, Ziyi Chen, Tianyu Feng, Fei Xiong, Jue Wang, Yangang Wang, and Zongguo Wang. Tripartite interaction representation algorithm for crystal graph neural networks. *Scientific Reports*, 14(1), October 2024.

[82] Chunhui Zhang, Yijun Tian, Mingxuan Ju, Zheyuan Liu, Yanfang Ye, Nitesh Chawla, and Chuxu Zhang. Chasing all-round graph representation robustness: Model, training, and optimization. In *The Eleventh International Conference on Learning Representations*, 2023.

[83] Hengtong Zhang, Tianhang Zheng, Jing Gao, Chenglin Miao, Lu Su, Yaliang Li, and Kui Ren. Data poisoning attack against knowledge graph embedding. In *Proceedings of the Twenty-Eighth*

*International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 4853–4859. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[84] Zehong Zhang, Lifan Chen, Feisheng Zhong, Dingyan Wang, Jiaxin Jiang, Sulin Zhang, Hualiang Jiang, Mingyue Zheng, and Xutong Li. Graph neural network approaches for drug-target interactions. *Current Opinion in Structural Biology*, 73:102327, 2022.

[85] Tianzhe Zhao, Jiaoyan Chen, Yanchi Ru, Qika Lin, Yuxia Geng, and Jun Liu. Untargeted adversarial attack on knowledge graph embeddings, 2024.

[86] 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 & Data Mining*, KDD '18, page 2847–2856, New York, NY, USA, 2018. Association for Computing Machinery.

[87] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations (ICLR)*, 2019.