Saarland University
Department of Computer Science

Understanding and Assessment of Privacy Risks in Machine Learning Systems

Dissertation
zur Erlangung des Grades
der Doktorin der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von

Min Chen

Saarbrücken, 2024

Tag des Kolloquiums:	22. Mai 2024
Dekan:	Prof. Dr. Roland Speicher

Prüfungsausschuss:

Vorsitzender:	Prof. Dr. Thorsten Herfet
Berichterstattende:	Prof. Dr. Michael Backes
	Prof. Ninghui Li
	Dr. Yang Zhang
Akademischer Mitarbeiter:	Dr. Mingjie Li

Zusammenfassung

Datenschutz ist aufgrund wachsender Bedenken um den Schutz persönlicher Informationen zu einem bedeutenden Thema geworden. Angesichts der vielschichtigen Natur des Paradigmas der Datenverarbeitung ist ein entscheidender Forschungsbereich die Erfassung und Minderung von Datenschutzrisiken im komplexen Lebenszyklus der Daten. Ein wichtiger Meilenstein im Verständnis von Datenschutzrisiken bei Daten ist die Datenschutz Regulierung.

Viele Länder und Regionen haben Datenschutzvorschriften erlassen, um die Privatsphäre der Nutzer zu schützen, aber ihnen fehlen konkrete Umsetzungs Details der Rechte in der Praxis des maschinellen Lernens (ML). Diese Dissertation präsentiert eine Arbeitslinie, die zunächst den Datenschutz von ML-Systemen im Licht der Datenschutzvorschriften versteht und dann die Datenschutzrisiken in zwei praktischen ML-Systemen bewertet. Wir beginnen mit der Datenschutzverordnung "Recht auf Vergessenwerden" und konzentrieren uns auf dessen technische Umsetzung in ML-Systemen, die als "maschinelles Verlernen" bezeichnet wird. Konkret entwerfen wir GraphEraser für das effiziente und nutzen erhaltende Widerrufen von Knoten/Kanten und deren Auswirkungen auf Graph-Neuronale-Netzwerke. Zweitens beantworten wir die Frage, ob bestehende Algorithmen zum maschinellen Verlernen zusätzliche Informationen über die Trainingsdaten preisgeben können. Wir stellen fest, dass das maschinelle Verlernen ursprünglich dazu entworfen wurde, die Datenschutzrisiken von widerrufenen Daten zu verringern, aber der Löschprozess zusätzliche Risiken aufdeckt. Drittens untersuchen wir eine gängige Praxis beim Teilen von Graph-Einbettungen mit Dritten, die Informationen über den Trainingsgraphen stark preisgibt. Schließlich bewerten wir die Datenschutzrisiken im Few-Shot-Gesichtserkennungssystem, und wir schlagen ein Prüfungstool namens Face-Auditor vor, um Probleme bei der nicht genehmigten Datennutzung anzugehen.

Unsere Ergebnisse veranschaulichen die unbeabsichtigten Datenschutzrisiken in den bestehenden Algorithmen zum maschinellen Verlernen, enthüllen empirisch die Datenschutzrisiken in ML-Systemen und können dazu beitragen, vertrauenswürdige, datenschutz orientierte ML-Systeme zu entwerfen.

Abstract

Data privacy has emerged as a significant issue due to the growing concern about safeguarding personal information. Given the multifaceted nature of the data processing paradigm, a crucial area of research pertains to comprehending and mitigating privacy risks throughout the intricate data life cycle. An important milestone in understanding data privacy risks is the privacy regulations.

Many countries and regions have enacted privacy regulations to protect users' privacy, but they lack concrete implementation details of the rights in the machine learning (ML) practice. This dissertation presents a line of work that first understands the privacy of ML systems through the lens of privacy regulations and then assesses the privacy risks in two practical ML systems. We start with the "right to be forgotten" regulation and focus on its technical implementation in ML systems, referred to as *machine unlearning*. Concretely, we design GraphEraser for efficient and utility-preserving revoking nodes/edges and their impacts on graph neural networks. Second, we answer whether existing machine unlearning algorithms can leak extra information about the training data. We find that machine unlearning was initially designed to lower the privacy risks of revoked data but the deleting process exposes more risks. Third, we examine a common practice when sharing graph embedding with third parties that severely leaks information about the training graph. Lastly, we assess the privacy risks in the few-shot facial recognition system, and we propose an auditing tool called Face-Auditor to address the unconsent data misuse problems.

Our results illustrate the unintended privacy risks in the existing machine unlearning algorithms, empirically reveal the privacy risks in ML systems, and can shed light on designing trustworthy, privacy-preserving ML systems.

Background of this Dissertation

This dissertation is based on the four papers [P1, P2, P3, P4] presented in the following. I contributed to all papers as the main author and was responsible for all major parts, including the technical design, empirical evaluation, and paper writing.

Chapter 3 is based on the work of [P1]. The idea of designing a machine unlearning framework for graph data originated from a random question in the group meeting - How to design a machine unlearning paradigm for graph data. Min Chen, Zhikun Zhang, and Yang Zhang jointly discuss the designing goal of graph unlearning. Min Chen and Zhikun Zhang searched for the solutions to balanced graph partition and designed the framework of GraphEraser. Min Chen implemented the design and performed all the evaluations. Tianhao Wang and Mathias Humbert provide valuable suggestions on refining the design. Min Chen, Zhikun Zhang, Tianhao Wang, Mathias Humbert, Michael Backes, and Yang Zhang participated in the writing and reviewing of the paper.

Chapter 4 is based on the work of [P2]. The research question assessing the unintended privacy leakage originated from a joint discussion between Yang Zhang, Zhikun Zhang, Tianhao Wang, and Min Chen. Min Chen, Zhikun Zhang, Tianhao Wang, and Yang Zhang jointly designed the attack methods. Min Chen implemented the attack on multiple model architectures and four defense methods. Mathias Humbert and Tianhao Wang provided valuable feedback on the different stages of the design and evaluation of attacks and defenses. Min Chen, Zhikun Zhang, Tianhao Wang, Mathias Humbert, Michael Backes, and Yang Zhang participated in improving and reviewing the paper.

Chapter 5 is based on the work of [P3]. The idea of inferring the graph attributes from observing the Graph Embedding is from a joint discussion of Zhikun Zhang, Yun Shen, Yang Zhang, and Min Chen. A research paper on privacy leakage in computer vision models inspires the initial research questions - What information can we infer from the graph embedding? Zhikun Zhang and Min Chen implemented the three attacks, the perturbation-based defense, and drafted the paper. Zhikun Zhang, Min Chen, Yun Shen, Michael Backes, and Yang Zhang participated in improving and reviewing the paper.

Chapter 6 is based on the work of [P4]. The idea of using a user-level membership inference attack as an auditing tool for data misuse detection was proposed by Min Chen. Min Chen carried out the design, implementation, and evaluation and drafted the paper. Zhikun Zhang and Tianhao Wang provide valuable suggestions for organizing

the paper. Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, and Yang Zhang participated in improving and reviewing the paper.

[P1] Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. Graph Unlearning. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2022, 499–513.

[P2] Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. When Machine Unlearning Jeopardizes Privacy. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021, 896–911.

[P3] Zhang, Z., Chen, M., Backes, M., Shen, Y., and Zhang, Y. Inference Attacks Against Graph Neural Networks. In: *USENIX Security Symposium (USENIX Security)*. 2022.

[P4] Chen, M., Zhang, Z., Wang, T., Backes, M., and Zhang, Y. FACE-AUDITOR: Data Auditing in Facial Recognition Systems. In: *USENIX Security Symposium (USENIX Security)*. 2023.

Further Contributions of the Author

The author also contributed to the following paper.

[S1] Yun Shen, Yufei Han, Zhikun Zhang, **Min Chen**, Ting Yu, Michael Backes, Yang Zhang, Gianluca Stringhini. Finding MNEMON: Reviving Memories of Node Embeddings. In ACM SIGSAC Conference on Computer and Communications Security (CCS), 2022.

[S2] Linkang Du, Zhikun Zhang, **Min Chen**, Minyang Sun, Shouling Ji, Peng Cheng, Jiming Chen, Michael Backes, Yang Zhang. HyperThief: Stealing Hyper-parameters from Deep Reinforcement Learning Models. Under review.

[S3] **Min Chen**, Karl Wuest, Michael Backes. SoK: Privacy Analysis of Blockchain in the Wild. 2023

Acknowledgments

Writing this dissertation means approaching the end of my Ph.D. journey. I would like to take this opportunity to express my gratitude to the person who companied and helped me in this long journey.

First, I want to give my deepest gratitude to my supervisor Prof. Michael Backes, for your attitude toward doing high-impact research and your endeavor to build CISPA into such a wonderful place for researchers. I had a fantastic journey when pursuing my Ph.D. at CISPA.

Second, I would like to thank my dissertation committee members: Prof. Dr. Michael Backes, Prof. Ninghui Li, and Dr. Yang Zhang. I appreciate your expertise and effort in reviewing this dissertation and providing inspiring comments to improve my work.

Thanks to my collaborators, Dr. Zhikun Zhang, Dr. Tianhao Wang, Dr. Yang Zhang, Prof. Mathias Humbert, and Dr. Yun Shen, I learned many things from you as a new-bite researcher. You have shown me great examples of formulating interesting research questions, critical thinking, the art of writing academic papers, and doing rigorous and high impacts research.

Thanks to all my colleagues inside and outside CISPA. Thanks to Julia, Olga, and Bettina, for taking plenty of care whenever I need help. Thanks to colleagues from other departments, including but not limited to IT, scientific support, travel department, HR department, and the front office, your professional work makes doing research at CISPA so engaging. Thanks to Tin, Ahmed, Jie, Yugeng, and Allen, who have made this journey fun and lovely. Also, I would like to thank CISPA Sparta, whenever I take a short break and stare at you, I can feel the strength and courage to deal with any complicated situation.

I would like to express my heartfelt gratitude to my family for their unwavering support and endless love as I embark on this Ph.D. journey. They're always my energy charger. Special thanks to my husband, who plays many roles in my life. He is my labmate, playfellow, roommate, research mentor, faithful listener, trouble solver, and the only person to whom I could talk about everything. He shares in my joys and sadness, respecting and standing by every decision I make. His unwavering support has been instrumental in allowing me to pursue my dream and strive for success in this Ph.D. endeavor.

Contents

List of Figures

List of Tables

1

Introduction

The growing concern over personal information has led many countries and regions to enact privacy regulations aimed at safeguarding users' privacy. Examples of these regulations include the General Data Protection Regulation (GDPR) [1] in the European Union, the California Consumer Privacy Act (CCPA) [2] in California, the United States, the Personal Information Protection and Electronic Documents Act (PIPEDA) [3] in Canada, the Brazilian General Data Protection Law (LGPD) [4] in Brazil, Personal Information Protection Law of the People's Republic of China (PIPL) [5], and many others.

There are two main roles in these privacy regulations: The data provider and the data controller. The data providers originally generate and have authority over the data, and the data controllers determine the purposes and means of processing personal data in providing services or other commercial activities. These privacy laws govern many rights of data providers and the obligations of data controllers. One essential aspect of these regulations is the concept of **the right to be forgotten**, which allows users to withdraw their data from a company's databases or data products. However, complying with this right can pose significant challenges for companies using the data, as it often requires deleting multiple copies, updating or regenerating commercial models, and more. Another crucial provision is **the right of access**, which authorizes users to know how their data is processed, and companies must ensure the transparency of the data processing. Furthermore, **the right to restriction of processing** comes into play when data providers discover inaccuracies or unlawful processing of their data. In such cases, the data controller has the obligation to inform the data providers and protect their rights.

However, applying these rights to ML models proves to be more complex than other data processing methods. ML models have a powerful memorization ability, making them vulnerable to various privacy and security attacks, such as membership inference attacks [145, 132], model stealing attacks [162, 116], data reconstruction attacks [47, 131], poisoning attacks [75, 136], etc. The challenge lies in adapting these rights to the unique characteristics of ML practices, as concrete implementation details are scarce. It remains uncertain whether existing standard practices (such as in search engines or databases) can be directly applied to ML paradigms and whether new privacy risks may arise as a result. Finding the right balance between privacy protection and the effective use of ML models requires further exploration and understanding.

1.1 Contributions

In the following, we summarize the major contributions of this dissertation, which consist of four works. We first introduce an efficient and applicable method to implement "the right to be forgotten" in ML systems, then we further explore the privacy risks in the known machine unlearning systems (the practice of implementing the right to be forgotten in ML systems). We then assess privacy risks in two widely-used ML systems, i.e., the graph embedding sharing system and the few-shot facial recognition system.

1.1.1 Technical Implementation of Machine Unlearning

We start with the "right to be forgotten" regulation and its technical implementations in real-world ML systems. Specifically, the *right to be forgotten* entitles data owners to the right to delete their data from an entity storing it. In the context of ML, researchers have argued that, under the requirement of the right to be forgotten, the *model provider* has an obligation to eliminate any impact of the data whose owner requested to be forgotten, which is referred to as *machine unlearning* [28, 25].

While approximate machine unlearning methods can remove the impacts of the revoked samples from a trained model and deal with the large underlying training dataset, they either require specific model architecture [28, 51] or face the risks of incompletely remove [53, 59, 73, 112]. A deterministic machine unlearning approach consists in removing the revoked sample and retraining the ML model from scratch. Among the existing solutions, the SISA (Sharded, Isolated, Sliced, and Aggregated) is the most general one in terms of model architecture [25] and can guarantee the complete remove. The basic idea of SISA is to randomly split the training dataset into several disjoint shards and train each shard model separately. Upon receiving an unlearning request, the model provider only needs to retrain the corresponding shard model.

SISA is effective for data that can be easily partitioned into small shards, but it is unsuitable for scenarios where the model's training data is non-iid (non-independent and identically distributed) or when partitioning the data leads to a significant loss of utility, particularly in cases involving graph data. Graph data often exhibit strong interdependencies and structural correlations between nodes and edges, such as social networks, financial networks, biological networks, recommender systems, and transportation networks. Partitioning graph data can break these relationships and result in the loss of crucial information necessary for accurate and meaningful analysis. In such situations, SISA's effectiveness may diminish.

In Chapter 3, we propose an end-to-end machine unlearning framework tailed to graph data, namely GraphEraser, which can achieve high unlearning efficiency and low model utility degradation. This framework calls attention to balancing the model utility loss and the unlearning efficiency and provides insights to improve the model utility loss due to the data partition.

1.1.2 Understanding the Privacy Risks in Machine Unlearning

Machine unlearning naturally generates two versions of ML models, namely the *original model* and the *unlearned model*, and creates a discrepancy between them due to the target sample's deletion. While originally designed to protect the target sample's privacy, we argue that machine unlearning may leave some imprint of it and thus create unintended privacy risks. Specifically, while the original model may not reveal much private information about the target sample, additional information might be leaked through the unlearned model. We then study to what extent data is indelibly imprinted in an ML model by quantifying the additional information leakage caused by machine unlearning.

In Chapter 4, we propose a novel membership inference attack against multiple "machine unlearning" settings where an adversary can access both the original and unlearned models. Our empirical results showed that machine unlearning was initially designed to lower the privacy risks of revoked data. On the contrary, the deleting process exposures more chances of being inferred the membership status.

We show that machine unlearning degrades the privacy of the target sample in general. This discovery sheds light on the risks of implementing the right to be forgotten in the ML context. We believe that our attack and metrics can help develop more privacy-preserving machine unlearning approaches in the future.

1.1.3 Assessing the Privacy Risks in Graph Embedding Sharing System

Graph embedding is a powerful tool to solve the graph analytics problem by transforming the graph data into low-dimensional vectors. These vectors could also be shared with third parties to gain additional insights into what is behind the data. Graph embedding is empirically considered sanitized since the whole graph is compressed to a single vector.

Although sharing graph embeddings for downstream graph analysis tasks is intriguing and practical, the associated security and privacy implications remain unanswered. In

Chapter 5, we systematically investigate the information leakage of the graph embedding by mounting three inference attacks. First, we can successfully infer basic graph properties, such as the number of nodes, the number of edges, and graph density, of the target graph with up to 0.89 accuracy. Second, given a subgraph of interest and the graph embedding, we can determine with high confidence whether the subgraph is contained in the target graph. Third, we propose a novel graph reconstruction attack that can reconstruct a graph with similar graph structural statistics to the target graph. We further propose an effective defense mechanism based on graph embedding perturbation to mitigate the inference attacks without noticeable performance degradation for graph classification tasks.

As the research community has identified some information leakage from the raw data, sharing a compression version of raw data is often considered safer. However, for the first time, we examine a common practice when sharing graph embedding with third parties that could severely leak information about the training graph. This work sheds light on discovering unintended privacy risks in graph embedding systems and provides effective inference attacks to assess the privacy risks systematically.

1.1.4 Assessing the Privacy Risks in Few-shot Facial Recognition System

Machine learning has been widely adopted in real-world automatic decision-making applications. With the power of facial recognition systems, entities with moderate resources can canvas the Internet for face images and build well-performed facial recognition models without people's awareness and consent. Such kinds of face data misuse are potentially disastrous [106] and infringe privacy laws. GDPR states that *the personal data must only be processed if the individual has given explicit consent* (Article 6(1)(a)), and *the processing of personal data must be lawful, fair, and transparent* (Article 5(1)(a)) [1]. This means that if the third parties want to use the data owner's face images, they need to obtain consent from the data owner and inform the data owner how their face images are processed.

To prevent face images from being misused, one straightforward method is to modify the raw face images before uploading them to the Internet, such as distorting the face images [90], producing adversarial patches [158], or adding imperceptible pixel-level cloaks [137]. However, these approaches inevitably destroy the semantic information of the face images and also increase the difficulty of retroactivity. Also, researchers have argued that such defenses can be bypassed by newer technologies [126], which leads to

an endless arms race between the attacker and defender. In such cases, an auditing tool enables users to check whether a model is using/misusing their data is essential.

In Chapter 6, we investigate the auditing approach that enables ordinary users to detect whether their private face images are being used to train a facial recognition system when only similarity metric information is accessible. We propose Face-Auditor based on the notion of user-level membership inference and provide practical guidelines for using it in facial recognition systems. Face-Auditor can serve as a complementary tool for existing privacy-protective actions. For instance, governments and regulators can use Face-Auditor as a tool for enforcing privacy regulations by determining if models are misusing data and violating individuals' privacy rights. Individuals can adopt Face-Auditor as an auditing tool to detect potential face data misuse. If a misuse happens, they can legally correct or withdraw their data (according to GDPR Articles 15, 16, 17, 18). Besides, Face-Auditor can also be employed by model developers to conduct self-inspection and ensure that their models are compliant with privacy regulations while demonstrating transparency in their data processing practices.

1.2 Organization

The rest of this dissertation is organized as the following. In Chapter 2, we first provide the needed preliminaries and background knowledge to ground this dissertation. Then we summarize the related work to motivate our work and contributions to understand the vulnerabilities in ML models. We explore the technical implementation of the right to be forgotten in machine learning in Chapter 3. We investigate the unintended privacy risks in machine unlearning in Chapter 4. In Chapter 5, we assess the privacy risks of sharing graph embeddings. We propose a face auditor for data misuse detection and assess the technical implementation in Chapter 6. Finally, we look into some future work and conclude the dissertation in Chapter 7.

2

Preliminaries and Background

2.1 Machine Learning

2.1.1 Classification Models

Machine learning classification is the most common ML task. An ML classifier $\mathcal{M}$ maps a data sample x to posterior probabilities $\mathbb{P}$, where $\mathbb{P}$ is a vector of entries indicating the probability of x belonging to a specific class y according to the model $\mathcal{M}$. The sum of all values in $\mathbb{P}$ is 1 by definition. To construct an ML model, one needs to collect a set of data samples, referred to as the training set $\mathcal{D}$. The model is then built through a training process that aims at minimizing a predefined loss function with some optimization algorithms, such as stochastic gradient descent.

2.1.2 Graph Neural Networks

Numerous important real-world datasets are represented in the form of graphs, such as social networks [125], financial networks [91], biological networks [81], or transportation networks [39], recommender systems [117, 184], To take advantage of the rich information contained in graphs, a new family of ML models, graph neural networks (GNNs), has been recently proposed and has already shown great promise [15, 37, 82, 164, 160, 117, 179, 39, 67, 175, 26]. The core idea of GNNs is to transform the graph data into low-dimensional vectors by aggregating the feature information from neighboring nodes.

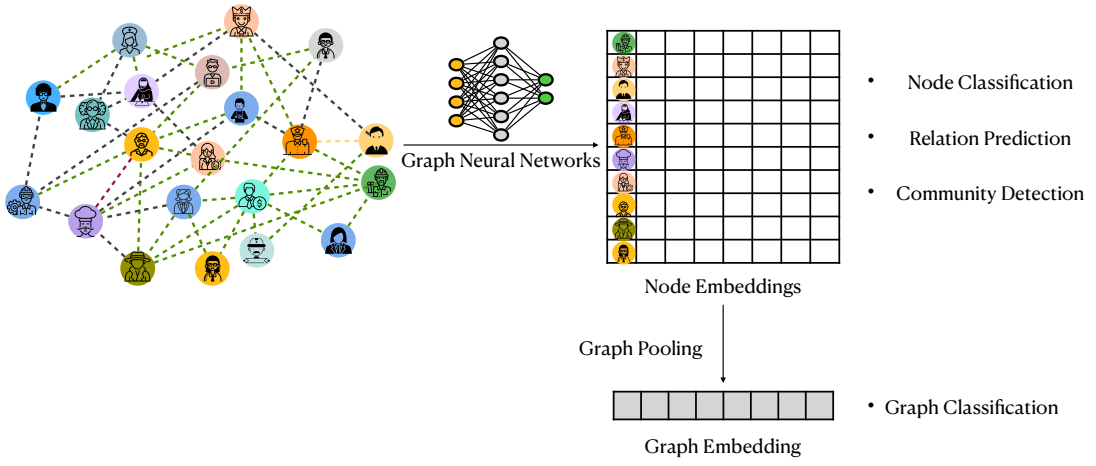


Figure 2.1: An illustration of GNN Model.

In a graph, a node represents an entity, and an edge connects different entities. We denote an attributed graph by $\mathcal{G} = \langle \mathcal{V}, A, F \rangle$, where $\mathcal{V}$ is the set of all nodes in graph $\mathcal{G}$,

$A \in \{0,1\}^{n \times n}$ is the corresponding adjacency matrix ($n = |\mathcal{V}|$), and $F \in \mathbb{R}^{n \times d_F}$ is the feature matrix with d_F being the dimension of node features. We denote $u, v \in \mathcal{V}$ as two nodes in $\mathcal{G}$, denote $e_{u,v}$ as an edge between u and v in $\mathcal{G}$. The notations frequently used in this paper are summarized in Table 2.1.

The basic intuition behind GNNs is that the neighboring nodes in a graph tend to have similar features; the GNN models are designed to aggregate the feature information from the neighborhood of each node to generate the node's embedding (e.g., a size-512 vector). The node embeddings can then be used to conduct downstream tasks, such as node classification [64, 164, 117], link prediction [172, 188], and graph classification [177, 174].

Table 2.1: Summary of the notations used in this section.

Notation	Description
$\mathcal{M}$	A machine learning model
$\mathcal{D}$	Dataset
P	Posteriors of a query sample
$\mathcal{G} = \langle \mathcal{V}, A, F \rangle$	Graph
$u, v \in \mathcal{V}$	Nodes in $\mathcal{G}$
$e_{u,v}$	Edge that connects u and v
A	Adjacency matrix of $\mathcal{G}$
F	Attributes associated with $\mathcal{V}$
$\mathcal{N}_u$	Neighborhood nodes of u
H_u	Node embedding of u
d_F / d_H	Dimension of attributes / embeddings
Φ	Aggregation operation in message passing
Ψ	Updating operation in message passing
$\mathbf{m}$	Message received from neighbors

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

2.1.2.1 Message Passing

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

operation Φ and *updating operation* Ψ:

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

where $\mathbf{m}_{\mathcal{N}_u}^i$ is the "message" received from u's neighbors. There are several possible implementations of Φ and Ψ as described below.

2.1.2.2 Aggregation Operation

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

- **Graph Isomorphism Networks (GIN) [177].** GIN directly sums up the embeddings of u's neighbors $\mathcal{N}_u$, i.e., $\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} H_v$.

- **Graph SAmple and aggreGatE (SAGE) [64].** The SAGE method takes an average over u's neighbors' embeddings rather than summing them up, i.e., $\mathbf{m}_{\mathcal{N}_u} = \frac{\sum_{v \in \mathcal{N}_u} H_v}{|\mathcal{N}_u|}$.

- **Graph Convolution Networks (GCN) [82].** The GCN method uses the symmetric normalization for aggregation, i.e., $\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} \frac{H_v}{\sqrt{|\mathcal{N}_u| \cdot |\mathcal{N}_v|}}$.

- **Graph Attention Networks (GAT) [164].** GAT assigns an attention weight or importance score to each neighbor during the aggregation, i.e., $\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} \alpha_{u,v} H_v$, where the attention weight $\alpha_{u,v}$ is defined as follows:

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

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

2.1.2.3 Updating Operation

The updating operation Ψ combines the node embeddings from node u and the message from u's neighborhood $\mathcal{N}_u$. The most straightforward updating operation is to calculate the weighted combination [134]. Formally, we denote the basic updating operation as $\Psi_{base} = \sigma(W_{self} H_u + W_{neigh} \mathbf{m}_{\mathcal{N}_u})$, where W_{self} and W_{neigh} are learnable parameters,

σ is a non-linear activation function. Another method is to treat the basic updating operation as a building block and concatenate it with the current embedding [64]. We denote the concatenation-based updating operation as $\Psi_{concat} = \Psi_{base}||H_u$, where $||$ is the concatenation operation. An alternative is to use the weighted average of the basic updating method and the current embedding [123], which is referred to as *interpolation-based* updating operation and is formally defined as $\Psi_{inter} = \alpha_1 \circ \Psi_{base} + \alpha_2 \circ H_u$. We introduce three popular updating operations.

- **Linear Combination [134].** The most basic updating method is to calculate the linear combinations, i.e.,

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

  where W_{self} and W_{neigh} are learned during the training process and σ is a non-linear activation function. The main issue of the basic method is over-smoothing, resulting in embeddings of all nodes being similar to each other after several steps of aggregation.

- **Concatenation [64].** One practical approach to handle the over-smoothing issue is to concatenate the result of the linear combination method with the current node embedding, i.e., $\Psi_{concat} = \Psi_{linear}||H_u$.

- **Interpolation [123].** Another method is to use the weighted average of the linear combination method and the current embedding for updating, i.e., $\Psi_{inter} = \alpha_1 \circ \Psi_{linear} + \alpha_2 \circ H_u$.

2.1.2.4 Graph Pooling

The *graph pooling operation* Σ aggregates the embeddings of all nodes in the graph to form a whole graph embedding, i.e., $H_{\mathcal{G}} = \Sigma\left(H_u, \forall u \in \mathcal{G}\right)$.

Global Pooling. The most straightforward approach for graph pooling is to directly aggregate all the node embeddings, which is called global pooling, such as *max pooling* and *mean pooling*. Although simple and efficient, the global pooling operation could lose the graph structural information, leading to unsatisfactory performance [185, 23].

Hierarchical Pooling. To better capture the graph structural information, researchers have proposed many hierarchical pooling methods [185, 23]. The general idea is to aggregate n node embeddings to one graph embedding hierarchically instead of aggregating them in one step as global pooling. Concretely, we obtain n node embeddings

using message-passing modules and find m clusters according to the node embeddings, where $1 < m < n$. Next, we treat each cluster as a node with features being the graph embedding of this cluster, then iteratively apply the message passing and clustering operations until there is only one graph embedding.

Formally, in the ℓ-th pooling step, we need to learn a cluster assignment matrix $S^\ell \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$, which provides a soft assignment of each node at layer ℓ to a cluster in the next coarsened layer $\ell + 1$. Suppose S^ℓ in layer ℓ has already been computed. We can use the following equations to compute the coarsened adjacency matrix $A^{\ell+1}$ and a new matrix of node embeddings $H^{\ell+1}$:

$$H^{\ell+1} = {S^\ell}^T H^\ell \in \mathbb{R}^{n_{\ell+1} \times d_H}$$
$$A^{\ell+1} = {S^\ell}^T A^\ell S^\ell \in \mathbb{R}^{n_{\ell+1} \times n_{\ell+1}}$$

The main challenge lies in how to learn the cluster assignment matrix S^ℓ. In the following, we introduce two state-of-the-art methods.

- **Differential Pooling [185].** The DiffPool method uses a message-passing module to calculate the assignment matrix as $S^\ell = \text{softmax}\Big(\text{GNN}(A^\ell, H^\ell)\Big)$. In practice, it can be difficult to train the GNN models using only gradient signals from the output layer. To alleviate this issue, DiffPool introduces an auxiliary link prediction objective to each pooling layer, which encodes the intuition that nearby nodes should be pooled together. In addition, DiffPool introduces another objective to each pooling layer that minimizes the entropy of the cluster assignment.

- **MinCut Pooling [23].** The MinCutPool method uses an MLP (multi-layer perceptron) module to compute the assignment matrix as $S^\ell = \text{softmax}\Big(\text{MLP}(A^\ell, H^\ell)\Big)$. Different from DiffPool, MinCutPool introduces the minimum cut objective to each pooling layer that aims to remove the minimum volume of edges, which aligns with the objective of graph pooling (aiming to assign the closely connected nodes into the same cluster).

2.1.2.5 Implementation of GNN Model

Typically, each step of message passing is referred to as a *GNN module*, and a GNN model can be implemented by stacking multiple layers of the GNN module and one layer of the softmax module for node classification. We denote a GNN model by $\mathcal{M}$, which can take as input the feature matrix F and the adjacency matrix A of a set of

nodes $\mathcal{V}$, and output a posterior matrix P. Here each row of P is the *posterior* of one node $u \in \mathcal{V}$, which is a vector of entries indicating the probability of node u belonging to a certain class. All values in each row of P sum up to 1 by definition.

The *graph-level* GNN models consist of a graph embedding module, which encodes the graph into the graph embedding, and a multi-class classifier, which predicts the label of the graph using the graph embedding. To train the GNN model, we normally adopt the cross-entropy loss. For graph embedding modules containing hierarchical pooling operations, we need to incorporate additional loss, such as minimum cut loss in MinCutPool.

Once the GNN model is trained, it can be used as a feature extractor to transform any graph into node embeddings or graph embeddings and perform downstream tasks, such as node classification, link prediction, and graph classification.

- **Node Classification.** Node classification aims to predict the labels for a node in a graph, given a partial training graph ($\mathcal{G}_{train}$). To accomplish this task, a graph neural network (GNN) model first generates a node embedding for each node by transforming a high-dimensional relationship into a low-dimensional vector. The model then employs a deep neural network (DNN) to map the relationship between the node embedding and the known label. The success of node classification depends on the assumption that similar nodes (in terms of their node embedding space) are likely to share the same label.

- **Link Prediction.** Link prediction tasks aim to find whether a connection exists between two entities given a partial graph. When the similarity of a pair of node embeddings is high, it indicates that a link exists between them. In the standard setup for link prediction, a model is given a set of nodes ($\mathcal{V}$) and an incomplete set of edges $\mathcal{E}_{train} \in \mathcal{E}$. It learns from the partial graph to establish a similarity threshold for making predictions and inferring the missing edges $\mathcal{E} \setminus \mathcal{E}_{train}$.

- **Graph Level Tasks.** This type of task needs an input of a graph embedding H. For instance, graph label prediction; graph clustering finds similar graphs by calculating the similarities of a pair of graph embeddings; and graph regression.

2.2 Machine Unlearning

One of the most important and controversial articles in existing privacy regulations is *the right to be forgotten*, which entitles data subjects to the right to delete their data

from an entity storing it to preserve their privacy. The "right to be forgotten" allows individuals to request the deletion of their data by the model owner to preserve their privacy. It also implies *data controller* has an obligation to eliminate any impact of the data whose owner requested to be forgotten. In the context of machine learning, e.g., MLaaS, this implies that the model owner should remove the *target sample* x from its training set $\mathcal{D}$. Moreover, any influence of x on the model $\mathcal{M}$ should also be removed. This process is referred to as *machine unlearning*.

Retraining from Scratch. The most legitimate way to implement machine unlearning involves removing the revoked sample and retraining the ML model from scratch. Formally, denoting the *original model* as $\mathcal{M}_o$ and its training dataset as $\mathcal{D}o$, this approach consists of training a new model $\mathcal{M}_u$ on dataset $\mathcal{D}u = \mathcal{D}o \setminus x$. We call this $\mathcal{M}_u$ the *unlearned model*.

Retraining from scratch is easy to implement. However, when the size of the original dataset $\mathcal{D}$o is large, and the model is complex, the computational overhead of retraining will be too large. To reduce the computational overhead, several approximate approaches have been proposed [28, 51, 29, 53, 59, 95, 73, 25, 112].

SISA. SISA [25] is an efficient and general method to implement machine unlearning that works in an ensemble way. The basic idea of SISA is to randomly split the training dataset into several disjoint shards and train each shard model separately. Upon receiving an unlearning request, the model provider only needs to retrain the corresponding shard model. Concretely, the training dataset $\mathcal{D}$o in SISA is partitioned into k disjoint parts $\mathcal{D}o1, \mathcal{D}o2, \cdots, \mathcal{D}ok$. The model owner trains a set of original ML models $\mathcal{M}_o^1, \mathcal{M}_o^2, \cdots, \mathcal{M}_o^k$ on each corresponding dataset $\mathcal{D}oi$. When the model owner receives a request to delete a data sample x, it just needs to retrain the sub-model $\mathcal{M}_o^i$ that contains x, resulting in unlearned model $\mathcal{M}_u^i$. Sub-models that do not contain x remain unchanged. Notice that the size of dataset $\mathcal{D}oi$ is much smaller than $\mathcal{D}o$; thus, the computational overhead of SISA is much smaller than the "retraining from scratch" method.

At inference time, the model owner aggregates predictions from the different sub-models to provide an overall prediction. The most commonly used aggregation strategy is the majority vote and posterior average. In Chapter 3, we show a learning-based aggregation strategy and use majority vote and posterior average as baselines to evaluate the impacts of the aggregator. In Chapter 4, we use the posterior average as the default aggregation method to evaluate our membership inference attack against SISA-based machine unlearning.

2.3 Privacy Attacks against Machine Learning Models

This dissertation mainly focuses on inference attacks against ML models. We use sample-level membership inference attack as a privacy risk evaluation tool to understand a model's training data privacy risks (in Chapter 3 and Chapter 4). We use a user-level membership inference attack as an auditing tool for potential data misuse without legal consent (in Chapter 6). We also use property inference attack, data reconstruction attack, and subgraph inference attack to obtain a fine-grained understanding of information leakage (in Chapter 5). The subgraph inference attack can be seen as a subset-level membership inference attack, and we describe the details of it in Section 5.3.

Sample-level Membership Inference. Given a model $\mathcal{M}_\theta$ trained on dataset $\mathcal{D}^T_{train}$, a target data point x, and some auxiliary information, the goal of a sample-level membership inference is to predict whether the data point x was used to train the model $\mathcal{M}_\theta$, i.e., $x \in \mathcal{D}^T_{train}$ (member sample) or $x \notin \mathcal{D}^T_{train}$ (non-member sample).

Subset-level Membership Inference. Given a model $\mathcal{M}_\theta$ trained on dataset $\mathcal{D}^T_{train}$, a target data subset $S = \{x_1, x_2, ..., x_n\}$, and some auxiliary information, the goal of a subset-level membership inference is to predict whether the subset S was used to train the model $\mathcal{M}_\theta$, i.e., $S \in \mathcal{D}^T_{train}$ (member subset) or $x \notin \mathcal{D}^T_{train}$ (non-member subset). In a graph, the subset represents a subgraph structure. The subset-level membership inference predicts whether a distinct subgraph structure was part of the model's training graph.

User-level Membership Inference Attack. Unlike the sample-level membership inference, a *user-level* membership aims to infer whether a user's any data sample was used to train a machine learning model $\mathcal{M}_\theta$. Formally, assume the target user u has a set of data samples $\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$, the user-level membership inference aims to distinguish between $\mathcal{U} \cap \mathcal{D}^T_{train} \neq \varnothing$ (member user) and $\mathcal{U} \cap \mathcal{D}^T_{train} = \varnothing$ (non-member user), where $\mathcal{D}^T_{train}$ is the training dataset of the target model $\mathcal{M}_\theta$.

Property Inference Attack. Given a model's output P or middle-layer representation H, a property inference attack aims to infer the statistical distribution of a model's training data based on auxiliary information.

Data reconstruction Attack. Given a middle-layer representation H_{x_o} that is generated from a data x_o, the goal of the data reconstruction attack aims to recover a data x_r that has the maximum similarity or minimum dissimilarity as x_o.

2.4 Applications of ML models

2.4.1 Facial Recognition System

Facial recognition is widely used for identification [169, 155, 135, 62]. Modern facial recognition system utilizes machine learning models to determine whether a face image being verified belongs to the authorized users (the complete system also includes other components such as face detection [180] and liveness detection [114]). In the training phase, a facial recognition model takes in multiple images for each user (e.g., from different angles) in advance. In the identification phase, the facial recognition model compares the image being examined with the pre-existing pictures to determine whether it belongs to the authorized users and (if yes) to which authorized user this image belongs.

The objective of the facial recognition system is to identify face images. Formally, there is a pre-defined set of persons, which we call *authorized users*. They each contribute multiple face images (which we call *anchor images*) for the system to "memorize" them so that when a new face image comes, the system knows which (if any) authorized user this image corresponds to. A straightforward approach is to train a classification model with the anchor face images. However, the classification model oftentimes requires a large amount of training data, while it is difficult to collect many face images from each authorized user in practice. Furthermore, the set of authorized users often changes over time, for example, when new colleagues join or leave a company. The classification model needs to be retrained when the set of authorized users changes.

2.4.2 Few-shot Learning for Facial Recognition

To address the challenges of generalizing to unknown data distribution and improve the scalability of facial recognition systems, companies have turned to using few-shot learning techniques [170]. More recently, few-shot learning [83, 147, 153, 46] dominate the traditional learning in facial recognition systems because it requires only *a few* "anchor" face images from the authorized users. Few-shot learning is a machine learning paradigm that aims to obtain good learning performance given limited supervised information in the training set [33]. The high-level idea of few-shot learning is to exploit prior knowledge to help train, thus reducing the size of the actual training set. An important branch of commonly used few-shot learning algorithms is based on *metric learning*, which learns the similarity/relation (measured by some metric) among the

images instead of (in the traditional classification problem) learning the mapping from an image to a specific label.

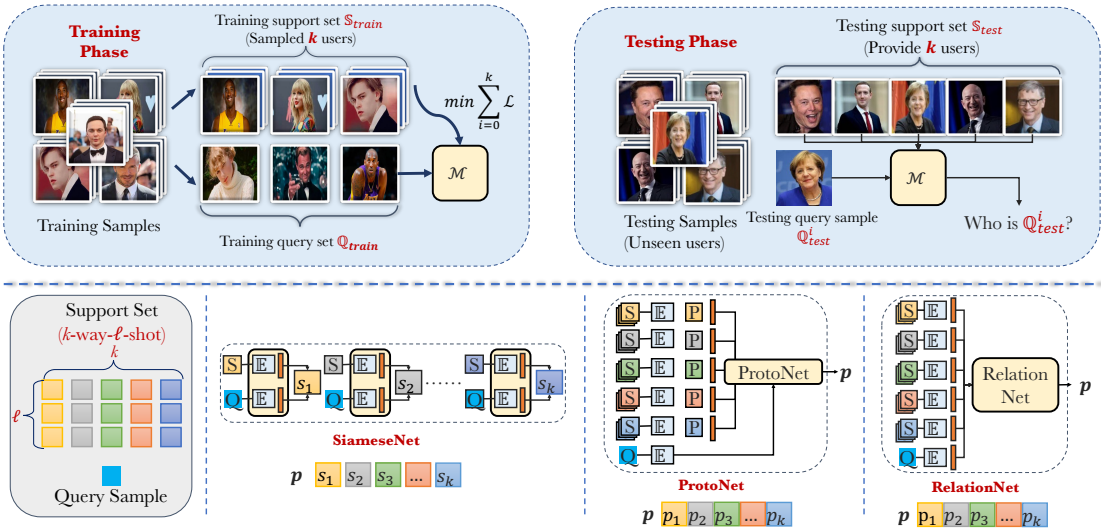


Figure 2.2: Illustration of the metric-based few-shot facial recognition models.

Figure 2.2 illustrates the general pipeline of metric-based few-shot learning algorithms, which consists of training and testing phases (also called the deployment phase in facial recognition systems). The training phase takes a large, publicly available training dataset $\mathcal{D}_{train}$ (which consists of samples for many classes) and runs in multiple iterations. In each iteration, we construct a *support set* $\mathbb{S}_{train}$, which consists of randomly selected k classes, each with ℓ samples, from $\mathcal{D}_{train}$ (this is referred to as k-way-ℓ-shot few-shot learning). We also construct a *query set* $\mathbb{Q}_{train}$ similar to $\mathbb{S}_{train}$ by sampling from the same classes (note that *query set* might be a bit confusing when used in training, but this is the standard terminology in few-shot learning). Our goal is to train a feature extractor F so that the features (embeddings) of the images from $\mathbb{S}_{train}$ and $\mathbb{Q}_{train}$ are optimized to be similar/close (in terms of some metric) if they belong to the same class, and dissimilar if they are from different classes. In the testing/deployment phase, we have a new support set $\mathbb{S}_{test}$ (anchor images from authorized users). Since F has already been trained to perform well in distinguishing samples from different classes, given one query image, we predict it as the closest class in $\mathbb{S}_{test}$.

2.5 Related Work

We first summarize existing privacy attacks against machine learning models to provide a general understanding of the privacy risks in Section 2.5.2. Then we discuss the practice of using privacy attacks as an evaluation tool for accessing the vulnerabilities of ML systems in Section 2.5.3.

2.5.1 Machine Unlearning

The notion of machine unlearning is first proposed in [28], which is the application of the right to be forgotten in the machine learning context. The most legitimate approach to implement machine unlearning is to remove the revoked samples from the original training dataset and retrain the ML model from scratch. However, retraining from scratch incurs very high computational overhead when the dataset is large and when the revoke requests happen frequently. Thus, most of the previous studies in machine unlearning focus on reducing the computational overhead of the unlearning process [28, 25, 20, 73].

For instance, Cao et al. proposed to transform the learning algorithms into a summation form that follows statistical query learning, breaking down the dependencies of training data [28]. To remove a data sample, the model owner only needs to remove the transformations of this data sample from the summations that depend on this sample. However, this algorithm does not apply to learning algorithms that cannot be transformed into summation forms, such as neural networks. Bourtoule et al. [25] proposed a more general algorithm named SISA. The main idea of SISA is to split the training data into disjoint shards, with each shard training one sub-model. To remove a specific sample, the model owner only needs to retrain the sub-model that contains this sample. To further speed up the unlearning process, the authors proposed to split each shard into several slices and store the intermediate model parameters when the model is updated by each slice.

Another line of machine unlearning study aims to verify whether the model owner complies with the data deletion request. Sommer et al. [148] proposed a backdoor-based method. The main idea is to allow the data owners to implant a backdoor in their data before training the ML model in the MLaaS setting. When the data owners later request to delete their data, they can verify whether it has been deleted by checking the backdoor success rate.

The research problem in this dissertation is orthogonal to previous studies. In

Chapter 3, we design a technical implementation of machine unlearning tailed to graph data. In Chapter 4, our goal is to quantify the unintended privacy risks for the deleted samples in machine learning systems when the adversary has access to both the original model and the unlearned model. To the best of our knowledge, we are the first to investigate this problem. Although quantifying privacy risks of machine unlearning has not been investigated yet, there are multiple studies on quantifying the privacy risks in the general right-to-be-forgotten setting. For example, Xue et al. [178] demonstrate that in search engine applications, the right to be forgotten can enable an adversary to discover deleted URLs when there are inconsistent regulation standards in different regions. Ellers et al. [43] demonstrate that, in network embeddings, the right to be forgotten enables an adversary to recover the deleted nodes by leveraging the difference between the two versions of the network embeddings.

2.5.2 Attacks against Machine Learning

The field of machine learning is not immune to privacy breaches, and several types of privacy attacks against machine learning (ML) models have been documented in the literature. On the one hand, these attacks demonstrate the vulnerabilities of ML models and emphasize the importance of protecting privacy in the ML paradigm. On the other hand, privacy attacks against ML model also offer a valuable tool for evaluating privacy risks, as they provide a comprehensive assessment of information leakages. By analyzing the vulnerabilities these attacks exploit, researchers can develop strategies and countermeasures to mitigate privacy breaches and enhance the overall security and privacy of ML models. We next summarize three types of privacy attacks against ML models. In Section 2.5.2.1, we discuss the membership inference attack. We discuss property inference attacks in Section 2.5.2.2 and data reconstruction attacks in Section 2.5.2.3. In the end, we also discuss some security attacks that can amplify the privacy risks in Section 2.5.2.4.

2.5.2.1 Membership Inference

Membership inference attacks aim to infer whether a sample was used to train a target model. Based on the information an adversary can access, existing membership inference attacks can be categorized into three categories: White-box, grey-box, and label-only. The inference methods rely on multiple assumptions, such as auxiliary datasets, target models' training dataset distribution, target models' architecture, shadow models, or

multiple queries to the target model.

Sample-level Membership Inference. Previous studies on membership inference attacks mainly focus on sample-level membership inference [145, 132, 94, 104, 65]. The first membership inference attack was proposed by Shokri et al. [145], which uses shadow models to mimic the target model's behavior and generate training data for the attack model. Salem et al. [132] gradually removed the assumptions of [145] by proposing three different attack methods. More recently, membership inference has been extensively investigated in various ML models and tasks, such as federated learning [104], white-box classification [110], generative adversarial networks [65, 32], and computer vision segmentation [69], recommender system [187], graph neural networks [P3], contrastive learning [68], reinforcement learning [118, 54], and diffusion models [195, 30].

To mitigate the threat of membership inference, a plethora of defense mechanisms have been proposed. The incremental defenses do not modify the training pipeline but slightly modify different training procedures. The methods can be classified into three classes: Reducing overfitting, perturbing posteriors, and adversarial training. There are several ways to reduce overfitting in the machine learning field, such as ℓ_2-regularization [145], dropout [132], and model stacking [132]. In [88], the authors proposed to explicitly reduce the overfitting by adding to the training loss function a regularization term, which is defined as the difference between the output distributions of the training set and the validation set. Jia et al. [77] proposed a posterior perturbation method inspired by adversarial examples. Nasr et al. [109] proposed an adversarial training defense to train a secure target classifier. During the training of the target model, a defender's attack model is trained simultaneously to launch the membership inference attack. The optimization objective of the target model is to reduce the prediction loss while minimizing the membership inference attack accuracy. Another defense method that modifies the training pipeline of the model can better negate the model utility and membership inference attack performance. A recent paper summarizes the state-of-the-art defense mechanisms against sample-level membership inference attacks and proposes a self-distillation-based ensemble architecture [157].

User-level Membership Inference. Compared to the sample-level membership inference, the user-level inference is less investigated. The first user-level membership inference was proposed by Song et al. [151] for the natural language models, including next-word prediction, neural machine translation, and dialog generation. They design and evaluate a black-box auditing method that can detect, with very few queries to a model, if a particular user's texts were used to train it. Miao et al. [105] then investigate

the user-level membership inference against the automatic speech recognition model.

2.5.2.2 Property Inference Attacks

Ganju et al. [49] proposed the first property inference attack aiming at inferring general properties of the training data (such as the proportion of each class in the training data). An adversary needs white-box access to the victim model and trains a multi-class meta-classifier to infer the underlying property. The meta-classifier is trained on multiple shadow models from different datasets distribution, and a successful attack is transferred from the shadow knowledge. Later, this attack is extended to collaborative learning [104], generative adversarial networks [194]. A successful property inference attack, on the one hand, can violate the intellectual property of the model owner by revealing the victim model's confidential training data distribution. On the other hand, it can also serve as a stepping stone to launch other privacy attacks, such as membership inference attacks [194].

2.5.2.3 Data Extraction Attacks

Data extraction attacks focus on inferring the missing attributes of the target ML model. Specifically, **model inversion attacks** [48, 47] aim to find the feature of a data sample when an adversary can access the label of that sample, for instance, know the name of a person, and reconstruct how the person looks. **Attribute inference attacks** aim to infer the private attribute of a data sample when its public attribute is known. For instance, inferring the race of a person given the age or gender.

2.5.2.4 Security Attacks

The aforementioned attacks are all privacy attacks, which aim to infer the private properties from the training data or distribution of training data. Another important attack category against ML models is the security attacks, such as evasion attacks, backdoor attacks, poisoning attacks, and model stealing attacks. These attacks were originally designed to impact the model utility; the attack methods can be used to reveal some private information.

Evasion attacks design adversarial examples [120, 119, 161, 121] in the model deployment phase. In this setting, an adversary adds carefully crafted noise to samples to mislead the target classifier. An example shows that using adversarial examples

as a distance metric can successfully launch a membership inference attack under the label-only setting [92].

In a backdoor attack, the adversary (as a model trainer) embeds a trigger into the model during the model training phase, aiming for targeted output for a specific triggered data sample when the model is deployed [58, 166]. While the initial backdoor is simple and brute force. Lately, clean-label backdoors have been proposed to keep the modification stealthy [130]. Inspired by the intuition of backdoor attacks that it behaves differently on clean data and poisoned data, Hu et al. propose a membership inference attack that is enhanced by backdoor [70].

In a model stealing attack, the adversary aims to copy a model with minimum cost. Tramèr et al. [162] proposed the first attack on inferring a model's parameters. Other works focus on inferring a model's hyperparameters [115, 165, 140]. Model stealing attacks can harm the intellectual property of the victim model. In the meantime, knowing more information about the model also increase the risks of revealing private information about the victim's training data [115].

2.5.3 Attacks as an Auditing Tool for AI Systems

Using attacks against machine learning as an auditing tool has been a growing trend in trustworthy AI [151, 96]. "Desirable attacks" [13] against ML, as an example, can be used for legitimate concerns like human rights and civil liberties. Determining whether a given image is present in a facial recognition database [47] can help individuals determine whether they can bring a court case against the service provider. Model inversion [47] can detect potential bias decision-making in credit risk evaluation systems. Adversarial examples can be used as an obfuscation tool to make users less likely to be tracked [6] or re-identified [137]. A similar notion of "subversive AI" adopts human-centered enhanced adversarial machine learning to evade algorithmic surveillance before publishing content online. Protective Optimization Technologies (POTs) [85] offer a more general terminology for repurposing the original system to enhance privacy, evade discrimination, or avoid surveillance.

2.5.4 Privacy of Facial Recognition System

With the proliferation of facial recognition systems, their privacy issues have attracted increasing attention [156, 31, 173]. To protect users' privacy, one strategy is to make the face images difficult for a facial recognition system to recognize by relying on adversarial

examples [45, 31]. Sharif et al. [138] show that adding specially printed glasses can cause the wearer to be misidentified. Komkov et al. [84] propose to add carefully computed adversarial stickers on a hat to reduce its wearer's likelihood of being recognized. Others propose to add adversarial patches to make it difficult for facial recognition systems to recognize the user as a person in an image [158, 176]. An alternative is to evade the facial recognition models by poisoning their training samples. One representative method is Fawkes [137]. However, these approaches can inevitably destroy the semantic information of the face images and is still vulnerable to advanced adversaries [126].

3

Technical Implementation of Machine Unlearning

Machine unlearning is a difficult task. One natural research question is whether we can satisfy the requirement of the right to be forgotten, design an efficient unlearning solution, and does minimum side effects on the machine learning model performance. Specifically, we focus on graph data because numerous important real-world datasets are represented in the form of graphs, and this type of data brings more challenges than IID data (i.e., images). Graph neural network (GNN) is the state-of-the-art way to analyze graph data. To overcome the non-Euclidean essence, GNN transforms each node into a low-dimension vector, which is calculated by combining the node feature and aggregating information from its neighboring nodes. Similar to other ML models, GNNs can be trained on sensitive graphs such as social networks [117, 125], where the data subject may request to revoke their data. However, learning representative GNNs rely on graph structural information. Randomly partitioning the nodes into sub-graphs (as in SISA) could severely damage the resulting model utility. Therefore, there is a pressing need for novel methods for unlearning previously seen – but revoked – data samples in the context of GNNs.

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

To design the framework, we meet several challenges. First, to permit efficient retraining, the shard graph size should be balanced. However, as we show in Section 3.2, highly unbalanced shard sizes might exist due to the structural properties of real-world graphs [190, 52]. In such case, many (if not most) of the revoked samples would belong to the largest shard whose retraining time would be substantial, and the unlearning process would become highly inefficient. Second, the graph partition methods must preserve the graph structural information to the greatest extent. Otherwise, the model utility might be jeopardized or cause a useless model. Besides, the framework should be general to any graph type and GNN model structure and robust to practical unlearning scenarios.

In the following, we first define the graph unlearning problem in Section 3.1. We then introduce the graph partition module for GraphEraser in Section 3.2 and the learning-based aggregation method in Section 3.3. We perform extensive experiments in Section 3.5 to evaluate the unlearning efficiency and model utility of GraphEraser. We discuss the practical implication of GraphEraser in Section 3.6 and conclude the chapter

in Section 3.7.

3.1 Graph Unlearning

3.1.1 Problem Definition

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

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

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

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

Challenges of Unlearning in GNNs. As mentioned before, the state-of-the-art approach for machine unlearning is SISA [25], which randomly partitions the training set into multiple shards and trains a constituent model for each shard. SISA has achieved

high unlearning efficiency and comparable model utility for ML models whose inputs reside in the Euclidean space, such as images and texts. However, the input of a GNN is a graph, and data samples, i.e., nodes of the graph, are not independently and identically distributed. Naively applying SISA on GNNs for unlearning, i.e., randomly partitioning the training graph into multiple shards, will destroy the training graph's structure which may result in large model utility loss. One solution is to rely on community detection methods to partition the training graph by the detected communities, which can preserve the graph structure to a large extent. However, directly adopting classical community detection methods may lead to highly unbalanced shards in terms of shard size due to the specific structural properties of real-world graphs [52, 127, 190] (see Section 3.2.1 for more details). In consequence, the efficiency of the unlearning process will be affected. Indeed, a revoked record would be more likely to belong to a large shard whose retraining time would be larger. Therefore, in the context of GNNs, the unlearning algorithm should satisfy the following objectives:

- **G1: Balanced Shards.** Different shards should share a similar size in terms of the number of nodes in each shard. In this way, each shard's retraining time is similar, which improves the efficiency of the whole graph unlearning process. Enforcing this objective can automatically satisfy the general unlearning pursuit of high unlearning efficiency.

- **G2: Comparable Model Utility.** Graph structural information is the major factor that determines the performance of GNN [82, 64, 174]. To achieve comparable model utility, i.e., high prediction accuracy in node classification tasks, each shard should preserve the structural properties of the training graph.

3.1.2 GraphEraser Framework

To address the two challenges of unlearning in GNNs, we propose GraphEraser, which consists of the following three phases: Balanced graph partition, shard model training, and shard model aggregation. The general framework of GraphEraser is illustrated in Figure 3.1. It partitions the original training graph into disjoint shards, parallelly trains a set of shard models $\mathcal{M}_i$, and learns an optimal importance score α_i for each shard model. When a node w needs prediction, GraphEraser sends w to all the shard models and obtains the corresponding posteriors, which are then aggregated using the optimal importance score α_i to make the prediction. When a node u mounts an unlearning

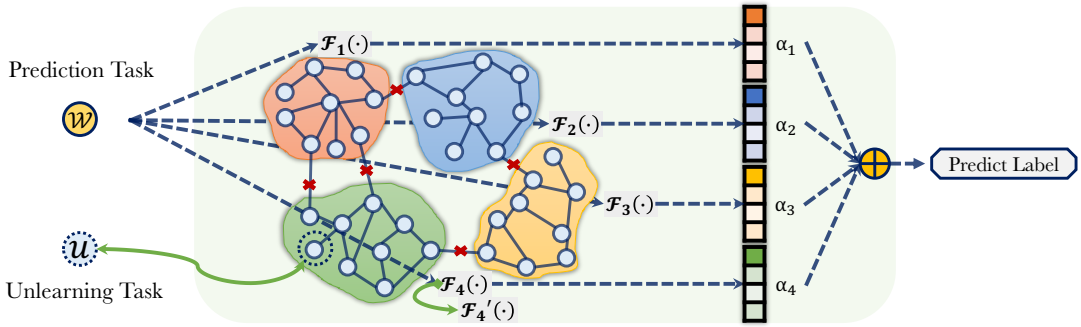


Figure 3.1: A schematic view of the framework of GraphEraser.

request, GraphEraser removes u from the corresponding shard and retrains the shard model.

Balanced Graph Partition. It is a crucial step of GraphEraser to fulfill the two requirements defined in Section 3.1.1. We propose to use balanced graph partition methods to partition the training graph into disjoint shards.

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

Shard Model Aggregation. At the inference phase, for predicting the label of node w, GraphEraser sends the corresponding data (the features of w, the features of its neighbors, and the graph structure among them) to all the shard models simultaneously, and the final prediction is obtained by aggregating the predictions from all the shard models. The most straightforward aggregation strategy, also mainly used in [25] is majority voting, where each shard model predicts a label and w takes the label predicted most often. We refer to this aggregation strategy as MajAggr.

An alternative solution is to gather the posterior vectors of all shard models and average them to obtain aggregated posteriors. The target nodes are predicted as the highest posterior in this aggregation. We name this aggregation strategy as MeanAggr.

Observe that different shard models can contribute differently to the final prediction; thus, allocating the same *importance score* for all shard models during the aggregation phase might not achieve the best prediction accuracy. Therefore, we further propose LBAggr, a learning-based method to find the optimal importance score allocation for different shard models (see details in Section 3.3). This further improves the prediction

accuracy of GraphEraser.

3.2 Balanced Graph Partition

In this section, we introduce the graph partition module.

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

This strategy can perfectly satisfy **G1** (Balanced Shards) in Section 3.1.1, while it cannot satisfy **G2** (Comparable Model Utility) since it can destroy the structural information of the graph. Thus, we treat this strategy as a baseline strategy.

To permit efficient retraining while keeping the structural information of the graph, we propose two graph partition strategies to address **G2**.

- **Strategy 1.** Consider the structural information only and try to preserve it as much as possible. One promising approach to do this is relying on community detection [167, 56, 168].

- **Strategy 2.** Consider both the structural information and the node features. To implement this, we can first represent the node features and graph structure into low-dimensional vectors, namely node embeddings, and then cluster the node embeddings into different shards.

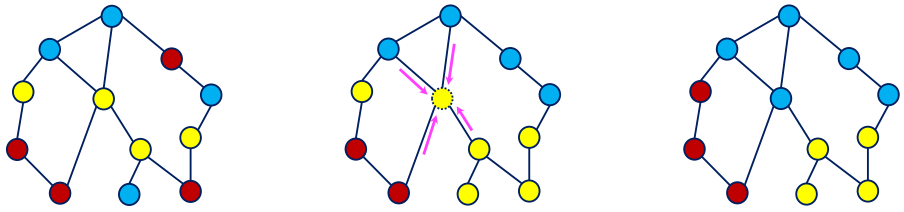
However, directly applying them can result in a highly unbalanced graph partition due to the underlying structural properties of real-world graphs (see the distribution of shard sizes with classical partition methods in Figure 3.4 and Figure 3.3). To address this issue, we propose a general principle for achieving a balanced graph partition (corresponding to **G1**) and apply this principle to design new approaches to achieve a balanced graph partition for both **Strategy 1** and **Strategy 2**. In the following, we elaborate on our balanced graph partition algorithms for **Strategy 1** and **Strategy 2**.

3.2.1 Community Detection Method (Strategy 1)

For **Strategy 1**, we rely on community detection, which aims at dividing the graph into groups of nodes with dense connections internally and sparse connections between groups.

A spectrum of community detection methods has been proposed, such as Louvain [190], Infomap [129], and Label Propagation Algorithm (LPA) [127, 167]. Among them, LPA has the advantage of low computational overhead and superior performance. Thus, we rely on LPA to design our graph partition algorithm. For consistency purposes, we use *shard* to represent the *community*.

Label Propagation Algorithm. Figure 3.2 illustrates the workflow of LPA. At the initial state, each node is assigned a random shard label (Figure 3.2a). During the label propagation phase (Figure 3.2b → Figure 3.2c), each node sends out its own label and updates its label to be the majority of the labels received from its neighbors. For instance, the yellow node with a dashed outline in Figure 3.2b will change its label to blue because the majority of its neighbors (two nodes above it) are labeled blue. The label propagation process iterates through all nodes multiple times until convergence (no nodes change their labels).



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

Figure 3.2: Illustration of LPA's workflow. Different colors represent different shards.

Unbalanced Partition. LPA is intriguing and powerful; however, directly applying the classical LPA results in a highly unbalanced graph partition. For instance, Figure 3.3 shows the distribution of shard size on the Cora dataset [181] (2166 nodes in the training graph). It originally generated 341 shards, and we only show the top 100 shards in terms of their sizes. We observe that the largest shard contains 113 nodes, while the smallest one contains only 2 nodes. Directly adopting the unbalanced shards detected by the classical LPA does not satisfy **G1**, which severely affects the unlearning efficiency. For instance, if the revoked node is in the largest shard, there is not much benefit in terms of unlearning time.

General Principle to Achieve Balanced Partition. To address the above issue, we propose a general recipe to achieve a balanced graph partition. Given the desired shard number k and maximal shard size δ, we define a *preference* for every *node-shard pairs* representing the node assigned to the shard (which is referred to as *destination*

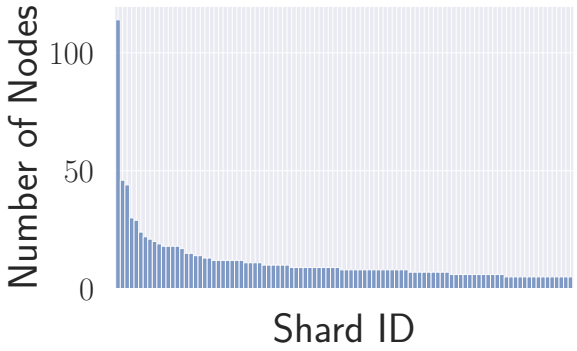


Figure 3.3: Distribution of shard sizes with classical LPA.

shard). This results in $k \times n$ node-shard pairs with different preference values. Then, we sort the node-shard pairs by preference values. For each pair in descending preference order, we assign the node to the destination shard if the current number of nodes in that destination shard does not exceed δ.

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

Algorithm 3.1 gives the workflow of BLPA. The algorithm takes as input the set of nodes $\mathcal{V}$, the adjacency matrix A, the number of desired shards k, the maximum number of nodes in each shard δ, maximum iteration T, and works in four steps as follows:

- **Step 1: Initialization.** We first randomly assign each node to one of the k shards (line 2).
- **Step 2: Reassignment Profiles Calculation.** For each node u, we denote its *reassignment profile* using a tuple $\langle u, \mathbb{C}_{src}, \mathbb{C}_{dst}, \xi \rangle$, where $\mathbb{C}_{src}$ and $\mathbb{C}_{dst}$ are the current and destination shards of node u, ξ is the neighbor counts of the destination shard $\mathbb{C}_{dst}$ (line 5 - line 7). We store all the reassignment profiles in $\mathbb{F}$.
- **Step 3: Reassignment Profiles Sorting.** We rely on the intuition that the reassignment profile with larger neighbor counts should have a higher priority to be fulfilled; thus we sort $\mathbb{F}$ in descending order by ξ and obtain $\mathbb{F}_s$ (line 10).
- **Step 4: Label Propagation.** Finally, we enumerate every instance of $\mathbb{F}_s$. If the size of the destination shard $\mathbb{C}_{dst}$ does not exceed the given threshold δ, we add

the node u to the destination shard and remove it from the current shard (line 11 - line 13). After that, we remove all the remaining tuples containing node u from $\mathbb{F}_s$.

The BLPA algorithm repeats steps 2-4 until the algorithm reaches the maximum iteration T, or the shard does not change (line 16 - line 17).

Algorithm 3.1 BLPA Algorithm

Input: The set of all nodes $\mathcal{V}$, adjacency matrix A, number of shards k, maximum number of nodes in each shard δ; maximum iteration T;
Output: Shards $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \cdots, \mathbb{C}_k\}$;

1 **Initialization:**
2 Randomly allocate all nodes into k shards and obtain $\mathbb{C}^0 = \{\mathbb{C}_1^0, \mathbb{C}_2^0, \cdots, \mathbb{C}_k^0\}$, step $t = 0$;
3 **Label Propagation:**
4 **while** *True* **do**
5 **foreach** *node* u *in* $\mathcal{V}$ **do**
6 **foreach** *shard* $\mathbb{C}_{dst}$ *in* $\{\mathbb{C}_i | v \in \mathcal{N}_u, v \in \mathbb{C}_i\}$ **do**
7 Store tuple $\langle u, \mathbb{C}_{src}, \mathbb{C}_{dst}, \xi \rangle$ in $\mathbb{F}$;
8 **end**
9 **end**
10 Sort $\mathbb{F}$ by ξ in descending order and obtain $\mathbb{F}_s$;
11 **foreach** *tuple in* $\mathbb{F}_s$ **do**
12 **if** $|\mathbb{C}_{dst}^t| < \delta$ **then**
13 $\mathbb{C}_{dst}^t \leftarrow \mathbb{C}_{dst}^{t-1} \cup u$;
 $\mathbb{C}_{src}^t \leftarrow \mathbb{C}_{src}^{t-1} \setminus u$;
 Remove all the remaining tuples containing node u from $\mathbb{F}_s$;
14 **end**
15 **end**
16 **if** $t > T$ *or the shard does not change* **then**
17 break;
18 **end**
19 $t \leftarrow t + 1$;
20 **end**
21 **return** $\mathbb{C}^t$.

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

3.2.2 Embedding Clustering Method (Strategy 2)

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

Embedding Clustering. We can adopt any state-of-the-art GNN models introduced in Section 2.1.2 to project each node into an embedding space. With respect to clustering, we rely on the widely used k-means algorithm[79], which consists of three phases: Initialization, node reassignment, and centroids updating. In the initialization phase, we randomly sample k *centroids*, which represent the "center" of each shard. In the node reassignment phase, each node is assigned to its "nearest" shard in terms of the Euclidean distance from the centroids. In the centroids updating phase, the new centroids are recalculated as the average of all the nodes in their corresponding shard.

Similar to the case of the LPA method, directly using k-means can also produce highly unbalanced shards. In Figure 3.4, we observe that on the Cora dataset, the largest shard contains 10.24% of the nodes, while the smallest one only contains 1.05% of the nodes.

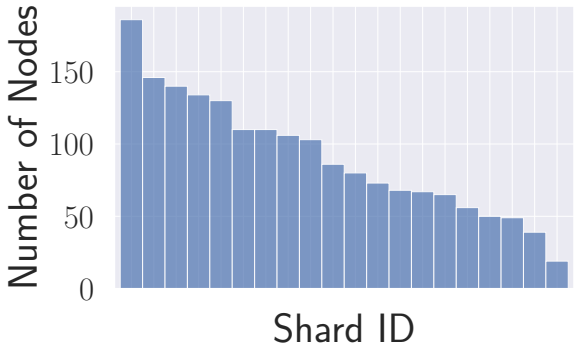


Figure 3.4: Distribution of shard sizes with classical k-means.

Balanced Embedding k-means (BEKM). Following the same principle for achieving a balanced partition, we propose BEKM as shown in Algorithm 3.2. We define the preference as the Euclidean distance between the node embedding and the centroid of the shard for all the node-shard pairs. A shorter distance implies a higher priority. BEKM takes as input the set of all node embeddings $\mathbb{H} = \{H_1, H_2, \cdots, H_n\}$, the number of desired shards k, the maximum number of node embeddings in each shard δ, the maximum number of iterations T, and works in four steps as follows:

- **Step 1: Initialization.** We first randomly select k centroids $C^0 = \{C_1^0, C_2^0, \cdots, C_k^0\}$ (line 2).

- **Step 2: Embedding-Centroid Distance Calculation.** Then, we calculate all the pairwise distances between the node embeddings and the centroids, which results in $n \times k$ embedding-centroid pairs. These pairs are stored in $\mathbb{F}$ (line 5 - line 7).

- **Step 3: Embedding-Centroid Distance Sorting.** We rely on the intuition that the embedding-centroid pairs with closer distance have higher priorities; thus, we sort $\mathbb{F}$ in ascending order and obtain $\mathbb{F}_s$ (line 10).

- **Step 4: Node Reassignment and Centroid Updating.** For each embedding-centroid pair in $\mathbb{F}_s$, if the size of $\mathbb{C}_j$ is smaller than δ, we assign node u to shard $\mathbb{C}_j$ (line 11 - line 18). After that, we remove all the remaining tuples containing node i from $\mathbb{F}_s$. Finally, the new centroids are calculated as the average of all the nodes in their corresponding shards.

The BEKM algorithm repeats steps 2-4 until the algorithm reaches the maximum iteration T, or the centroid does not change (line 20 - line 21).

Algorithm Analysis. Similar to BLPA, the computational complexity of BEKM depends on the size of $\mathbb{F}$. Since there are n nodes and k shards, the computational complexity of BEKM is $\mathcal{O}(k \cdot n)$. We empirically validate the convergence performance of BEKM in Section 3.2.3.

The node embeddings from the pretrained model are only used to define the partition of the graph but are not used in any stage of training and unlearning GNN models for each shard. The goal of unlearning is to unlearn nodes/edges' impact on the target GNN model. Thus, once the partition is defined, we keep it fixed. In this sense, once a node is removed, its influence is completely removed from the target GNN. In Section 3.5.5, we evaluate multiple unlearning iterations with a fixed partition and show the stability of graph partition methods.

Remarks. The choice between BLPA and BEKM depends on the GNN structure. In Section 3.5.2.2, we provide a guideline on which one to choose.

In addition, we emphasize that GraphEraser is a general framework for graph unlearning, and any other balanced graph partition methods can be plugged into it. In Section 3.5.3.3, we empirically compare our proposed BLPA and BEKM with several existing representative balanced graph partition methods and show that our proposed methods are either more computationally efficient or better performing.

Algorithm 3.2 BEKM Algorithm

Input: Node embeddings $\mathbb{H} = \{H_1, H_2, \cdots, H_n\}$, the number of clusters k, maximum number of nodes embedding in each cluster δ; maximum number of iteration T;

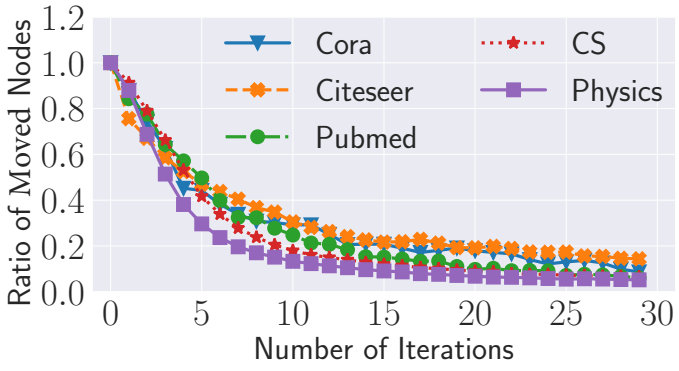
Output: Clusters $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \cdots, \mathbb{C}_k\}$;

Initialization:
Randomly select k centroids $C^0 = \{C_1^0, C_2^0, \cdots, C_k^0\}$, step $t = 0$;
while *True* **do**
  Nodes Reassignment:
  foreach *node embedding* $i \in \mathbb{H}$ **do**
    foreach *centroid* $j \in \mathbb{C}$ **do**
      Store $||H_i - C_j||_2$ in $\mathbb{F}$;
    end
  end
  Sort $\mathbb{F}$ in ascending order and obtain $\mathbb{F}_s$.
  foreach *node* i *and centroid* j *in* $\mathbb{F}_s$ **do**
    if $|\mathbb{C}_j^t| < \delta$ **then**
      $\mathbb{C}_j^t \leftarrow \mathbb{C}_j^t \cup i$;
      Remove all the remaining tuples containing node i from $\mathbb{F}_s$;
    end
  end
  Centroids Updating:
  foreach *cluster* $j \in \mathbb{C}^t$ **do**
    $C_j^t = \frac{\sum_{i \in \mathbb{C}_j^t} H_i}{|\mathbb{C}_j^t|}$;
  end
  if $t > T$ *or the centroid do not change* **then**
    break;
  end
  $t \leftarrow t + 1$;
end
return $\mathbb{C}^t$.

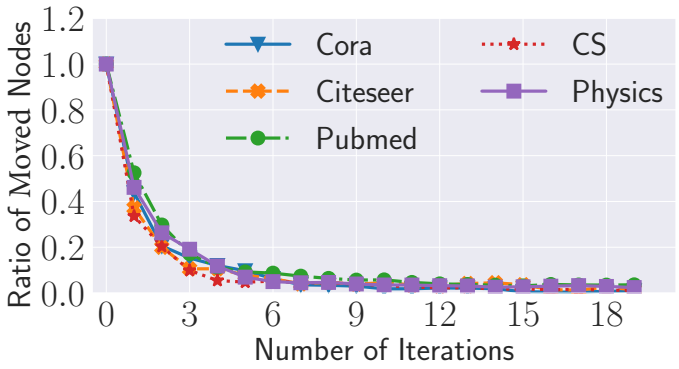
3.2.3 Convergence Analysis

It is difficult to theoretically prove the convergence of both BLPA and BEKM. Instead, we conduct empirical experiments to validate the convergence performance of both algorithms. An algorithm converges when the nodes of different shards between two consecutive iterations do not move. Figure 3.5 illustrates the ratio of moved nodes between different shards in each iteration. The experimental results show that the ratio of moved nodes gradually approximates zero within 30 iterations for both algorithms on all five datasets. Therefore, we set the number of iterations T to 30 for all our

experiments.



(a) BLPA



(b) BEKM

Figure 3.5: Convergence evaluation of GraphEraser-BLPA and GraphEraser-BEKM on five datasets.

3.3 Learning-based Aggregation (LBAggr)

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

$$\min_{\alpha} \mathop{\mathbb{E}}_{w \in \mathcal{G}_o} \left[\mathcal{L} \left(\sum_{i=0}^{m} \alpha_i \cdot \mathcal{M}_i(F_w, \mathcal{N}_w), y \right) \right] + \lambda \sum_{i=0}^{m} ||\alpha_i|| \tag{3.1}$$

where F_w and $\mathcal{N}_w$ are the feature vector and neighborhood of a node w from the training graph, y is the true label of w, $\mathcal{M}_i(\cdot)$ represents shard model i, α_i is the importance score for $\mathcal{M}_i(\cdot)$, and m is the total number of shards. We regulate the summation of

all importance scores to 1. Further, $\mathcal{L}$ represents the loss function and we adopt the standard cross-entropy loss. The regularization term $||\cdot||$ is used to reduce overfitting.

Solving the Optimization Problem. We can run gradient descent to find the optimal α to solve the optimization problem. However, directly running gradient descent can result in negative values in α. To address this issue, after each gradient descent iteration, we map the negative importance score back to 0. The mapping of the negative importance scores to 0 follows the general idea of projected gradient descent (PGD) [16]. In addition, the summation of the importance scores could deviate from 1. We first tried to normalize the importance score using the summation of current scores in each iteration; however, we empirically found that the loss could be extremely unstable across different epochs. Thus, we instead use the SoftMax function for normalization in each iteration.

Importance Scores Unlearning. Note that the nodes that learn the optimal importance scores can also be revoked by their data subjects. Therefore, we need to relearn the shard importance scores if a request-to-unlearn node is used to train the LBAggr, and this learning time is counted as part of the unlearning time. To reduce this relearning time, we propose to use only a small random subset of nodes from the training graph to relearn. We empirically show in Section 3.5.2.3 that using only 10% of the nodes in the training graph can achieve comparable utility as using all nodes. In this sense, relearning the optimal shard importance scores is unnecessary when the unlearned nodes are not used to train the LBAggr.

3.4 Putting Things Together: GraphEraser

Algorithm 3.3 illustrates the overall workflow of GraphEraser. It takes as input the training graph $\mathcal{G}_0$, the GNN model type f, and all necessary parameters for Algorithm 3.1 and Algorithm 3.2 (k, δ, and T). If f is a GCN, we invoke Algorithm 3.1 to partition $\mathcal{G}_0$; otherwise, we use Algorithm 3.2 (line 1 - line 6). We then use the partitioned graph $\mathcal{G}_s$ to train a set of shard models $\mathcal{M}$ (line 8). Finally, we randomly sample a set of nodes $\mathcal{V}_0$ from $\mathcal{G}_0$ to train the importance scores α for each shard model. The shard models and importance scores produced by GraphEraser can be used to predict the label of new samples. When the data owner revokes some nodes or edges, we only need to retrain the corresponding shard model.

Algorithm 3.3 GraphEraser

Input: Training graph $\mathcal{G}_0$, GNN model type f, number of shards k, maximum number of nodes in each shard δ, maximum iteration T;

Output: Shard models $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_k\}$, importance scores $\alpha = \{\alpha_1, \alpha_2, \cdots \alpha_k\}$;

Graph Partition:
if *the GNN model type f is* GCN*:* **then**
  Partitioning $\mathcal{G}_0$ into k shards with Algorithm 3.1 and obtain $\mathcal{G}_s = \{\mathcal{G}_s^1, \mathcal{G}_s^2, \cdots, \mathcal{G}_s^k\}$;
end
else
  Partitioning $\mathcal{G}_0$ into k shards with Algorithm 3.2 and obtain $\mathcal{G}_s = \{\mathcal{G}_s^1, \mathcal{G}_s^2, \cdots, \mathcal{G}_s^k\}$;
end
Shard Model Training:
Using $\mathcal{G}_s$ to train shard models $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_k\}$;
Importance Scores Learning:
Randomly sampling a set of nodes $\mathcal{V}_0$ from $\mathcal{G}_0$;
Replacing $\mathcal{G}_0$ in Equation 3.1 with $\mathcal{V}_0$ and train α;
return $\mathcal{M}$, α.

3.5 Evaluation

In this section, we evaluate the overall performance including unlearning efficiency, model utility, and the superiority of our proposed learning-based aggregation method LBAggr in Section 3.5.2. Second, to gain a better understanding of the uniqueness of graph data and its impacts on unlearning, we investigate the correlation between the properties of the shard models and the importance scores resulting from LBAggr and the impact of graph structure in a more controllable manner. We also examine the state-of-the-art balanced graph partition methods that come from three groups and compare them with our balanced graph partition methods in terms of running time and model utility Section 3.5.3. Third, we conduct ablation studies to show the impact of k and δ on the unlearning efficiency and model utility in Section 3.5.4. Four, we show the robustness of GraphEraser to the number of unlearned nodes/edges in Section 3.5.5. Finally, we launch a membership inference attack to examine the unlearning power of GraphEraser in Section 3.5.6.

3.5.1 Experimental Setup

Datasets. We conduct our experiments on five public graph datasets, including Cora, Citeseer, Pubmed [181], CS [139], and Physics [139]. These datasets are widely used as benchmark datasets for evaluating the performance of GNN models [82, 144, 193]. Table 3.1 summarizes the statistics of all the datasets.

Table 3.1: Dataset statistics for evaluating the performance of GraphEraser.

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

For the datasets in Table 3.1, Cora, Citeseer, and Pubmed are citation datasets, where the nodes represent the publications, and there is an edge between two publications if one cites the other. The node features are binary vectors indicating the presence of the keywords from a dictionary, and the class labels represent the publications' research field. CS and Physics are coauthor datasets, where two authors are connected if they collaborate on at least one paper. The node features represent paper keywords for each author's papers, and the class labels indicate the most active fields of study for each author.

GNN Models. We evaluate the efficiency and utility of GraphEraser on four state-of-the-art GNN models, including SAGE, GCN, GAT, and GIN (discussed in Section 2.1.2). For each GNN model, we stack two layers of GNN modules. All the models are implemented with the PyTorch Geometric[1] library. All the GNN models (including the shard models) considered in this chapter are trained for 100 epochs. We use the Adam optimizer and set the default learning rate to 0.01 with 0.001 weight decay.

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

- **Unlearning Efficiency.** Directly measuring the unlearning time for one unlearning request is inaccurate due to the diversity of shards. Thus, we calculate the *average unlearning time* for 100 independent unlearning requests. Concretely, we randomly

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

sample 100 nodes/edges from the training graph, record the retraining time of their corresponding shard models, and calculate the average retraining time.

- **Model Utility.** We use the *Micro F1 score* to measure the model utility, which is widely used to evaluate the prediction ability of GNN models on multi-class classification [57]. The F1 score is a harmonic mean of *precision* and *recall* and can provide a better measure of the incorrectly classified cases than the accuracy metric.

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

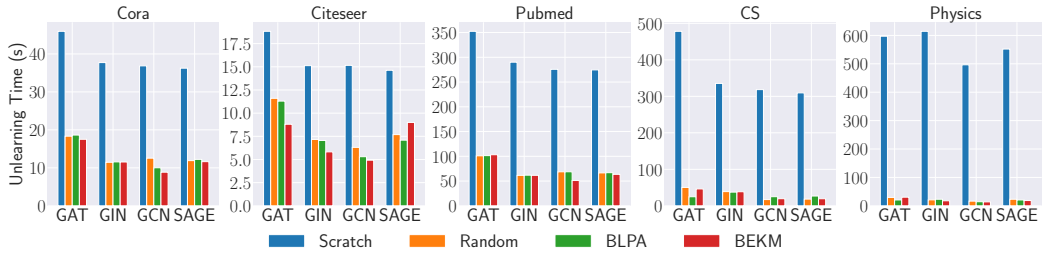
We implement community detection and embedding clustering-based graph partition methods in Section 3.2 for GraphEraser. For presentation purposes, we refer to them as GraphEraser-BLPA and GraphEraser-BEKM, respectively.

3.5.2 Overall Performance

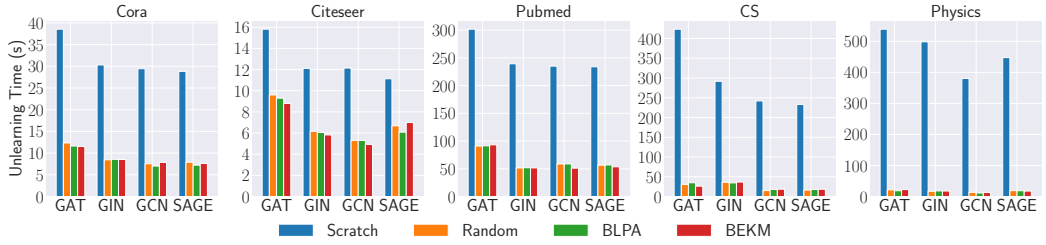
3.5.2.1 Evaluation of Unlearning Efficiency

In this section, we evaluate the unlearning efficiency of different graph unlearning methods on five datasets and four GNN models.

Setup. The default graph partition setting is based on the size of the original datasets, and we ensure each shard is trained on a reasonable number of nodes and edges. Specifically, we partition Cora, Citeseer, Pubmed, CS, and Physics into 20, 20, 50, 50, and 100 shards, respectively. Figure 3.6 illustrates the node/edge unlearning efficiency for different graph unlearning methods. We randomly unlearn 5% of the nodes in the training graph and report the time cost. For the shard-based unlearning methods, i.e., Random, GraphEraser-BLPA, and GraphEraser-BEKM, each unlearning request time cost consists of two parts: Retraining the shard models and relearning the importance scores of LBAggr. As discussed in Section 3.3, we only use a small portion of nodes in the training graph to learn the importance scores. The *average relearning time* of LBAggr on all datasets is shown in the last column of Table 3.2. The results show that the relearning time is less than 30 seconds for most of the datasets, which is negligible compared to retraining the shard models.



(a) Node Unlearning



(b) Edge Unlearning

Figure 3.6: Evaluation of the node/edge unlearning efficiency.

Results. We observe that the shard-based unlearning methods can significantly improve the unlearning efficiency compared to the Scratch method. For all four GNN models, we observe a similar time efficiency improvement level. In addition, the relative efficiency improvement of larger datasets (Pubmed, CS, and Physics) is more significant than that of smaller datasets (Cora and Citeseer). For instance, the unlearning time improvement is 4.16× for the Cora dataset, 3.08× for the Citeseer dataset, 5.40× for the Pubmed dataset, 19.25× for the CS dataset, and 35.9× for the Physics datasets. This is expected. From the Scratch method perspective, training a large graph can cost a large amount of time. From the shard-based methods perspective, we can tolerate more shards for larger graphs while preserving model utility. Comparing different shard-based methods, we observe that GraphEraser-BLPA and GraphEraser-BEKM have similar unlearning time as Random. This is made possible by our approach for achieving a balanced partition with BLPA and BEKM (see Section 3.2).

Additional Cost Analysis. Besides the unlearning cost, there are two additional costs in the GraphEraser framework: Graph partition cost and prediction cost. Table 3.2 illustrates these costs on five datasets. We observe that the graph partition costs of BLPA and BEKM are higher than Random. This is expected since both BLPA and BEKM need to iterate multiple times to preserve the structural information. Once the graph partition is done, we keep it fixed without redoing the graph partition. In this sense, we

Table 3.2: Computational costs of the GraphEraser framework on five datasets. We report the prediction cost and the relearning cost of LBAggr for BEKM.

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

can tolerate this cost since it is only executed once. We further show in Section 3.5.5 that using a fixed partition does not result in noticeable model utility degradation for GraphEraser.

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

3.5.2.2 Evaluation of Model Utility

Next, we evaluate the model utility of different graph unlearning methods. Table 3.3 (the red ground columns) shows the experimental results for node unlearning. For a fair comparison, we also apply LBAggr for Random.

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

To validate whether the graph structural information indeed diversely affects the GNN models' performance among different datasets, we introduce a baseline that uses

Table 3.3: Comparison of F1 scores for unlearning methods and different aggregation methods.

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

a 3-layer MLP (multi-layer perceptron) to train the prediction models for all datasets. Note that we only use the node features to train the MLP model without considering any graph structural information. Table 3.4 depicts the comparison of the F1 scores between the MLP model and four GNN models on five datasets. We observe that for the Cora and Citeseer datasets, the F1 score of the MLP model is significantly lower than that of the GNN models, which means the graph structural information plays a major role in the GNN models. On the other hand, the MLP model can achieve an adequate F1 score compared to the GNN models on Pubmed, CS, and Physics datasets, which means the graph structural information does not contribute much to the GNN models.

Table 3.4: Comparison of F1 scores for MLP and four GNN models. A larger gap in F1 scores for MLP and GNN models means that the graph structural information is more important for the GNN models.

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

In conclusion, the contribution of the graph structural information to the GNN

model can significantly affect the behaviors of different shard-based graph unlearning methods. To better illustrate the correlation between the importance of the graph structure and the utility improvement over Random, we conduct an ablation study in Section 3.5.4.

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

Regarding the choice between the two shard partition methods, i.e., GraphEraser-BLPA and GraphEraser-BEKM, we empirically observe that if the GNN follows the GCN structure, one can choose GraphEraser-BLPA; otherwise, one can adopt GraphEraser-BEKM. We posit this is because the GCN model requires the node degree information for normalization (see Section 2.1.2), and the GraphEraser-BLPA can preserve more local structural information thus better preserve the node degree [168].

Comparison with Scratch. Interestingly, we could observe that GraphEraser-BEKM performs slightly better than Scratch in some cases. For instance, the F1 score of GraphEraser-BEKM is 0.801 on the Cora dataset and the GIN model, while the corresponding F1 score of Scratch is 0.787. There are two possible reasons for this phenomenon. First, sampling often can eliminate some "noise" in the dataset, which is consistent with the observation of prior studies [186, 35]. Second, GraphEraser makes the final prediction by aggregating all submodels' results. In this sense, GraphEraser performs an ensemble which is another way to improve model performance.

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

3.5.2.3 Effectiveness of LBAggr

To validate the effectiveness of the LBAggr method proposed in Section 3.3, we compare with MeanAggr and MajAggr by conducting experiments on five datasets and four GNN models. Table 3.3 illustrates the F1 scores of different aggregation methods for Scratch, GraphEraser-BLPA, and GraphEraser-BEKM. Note that the Scratch method does not

need aggregation. We highlight the Scratch method in the green ground and our proposed methods in the red. For each graph partition strategy, we highlight the best value in **bold**. And for each GNN model, we highlight the best value in **blue bold**. We omit the results of edge unlearning due to similar conclusions.

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

Comparing different GNN models, GCN benefits the most while GIN benefits the least from LBAggr. In terms of model utility, the GraphEraser-BLPA method benefits the most from LBAggr. We conjecture this is because the BLPA partition method can capture the local structural information while losing some of the global structural information of the training graph [143, 168]. Using LBAggr helps better capture the global structural information by assigning different importance scores to shard models.

Table 3.5: Impact of the number of training nodes for learning LBAggr.

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

Impact of the Number of Training Nodes. As discussed in Section 3.3, to further improve the unlearning efficiency, one can use a small portion of nodes in the training graph to learn the importance score. Doing this can effectively reduce the relearning time of LBAggr, as shown in Section 3.5.2.1. Here we evaluate its impact on the model utility.

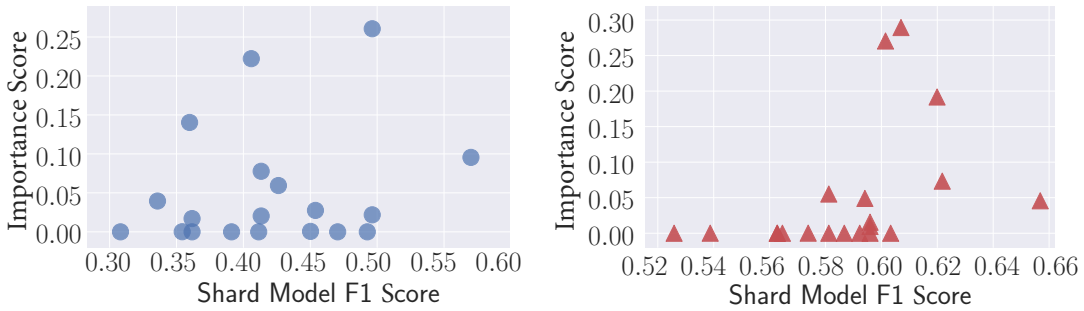
We experiment on three different cases: randomly sample 10% of nodes, randomly sample a fixed number of 1,000 nodes and use all nodes in the training graph.

Table 3.5 illustrates the results on five datasets and four GNN models for GraphEraser-BEKM. We observe that both using 10% of nodes and using a fixed number of 1,000 nodes can achieve comparable model utility as that of using all nodes. In practice, we suggest the model provider adopt the minimum of 10% and $1,000$ to learn the importance scores. In other words, the model provider can use 10% for small graphs and $1,000$ for large graphs. The conclusions are the same for GraphEraser-BLPA.

3.5.3 Gaining a Deeper Understanding

3.5.3.1 Correlation between Importance Scores and Shard Properties

To support the evidence of the effectiveness of LBAggr in Section 3.5.2.3, we next investigate the influence of a shard's properties on its importance score determined by the LBAggr method. Figure 3.7 depicts the correlation between each shard's F1 score



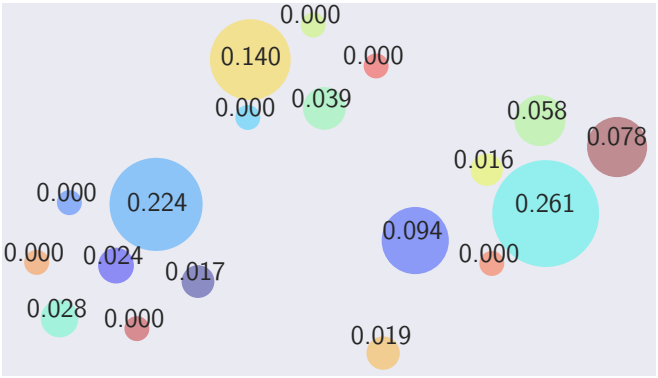
(a) GraphEraser-BLPA (b) GraphEraser-BEKM

Figure 3.7: Correlation between the importance score of a shard model and its F1 score on the Cora dataset.

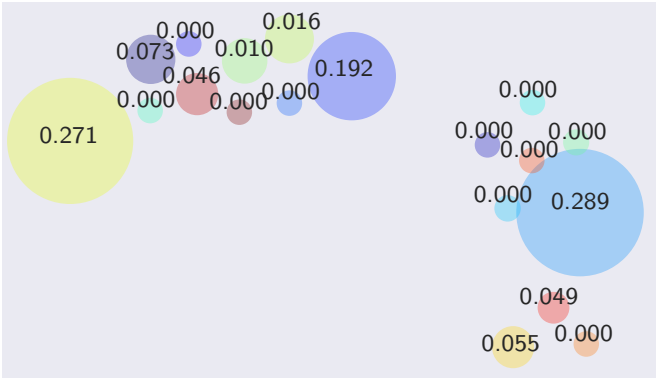
and its importance score. The x-axis stands for the shard model's F1 score, and the y-axis stands for the importance score of that shard. We report the results of the GAT model with GraphEraser-BLPA and GraphEraser-BEKM unlearning methods. Generally, shard models with more accurate predictions are assigned more significant importance scores. This demonstrates that LBAggr guides GraphEraser to choose the shards with the highest prediction capability.

We further investigate whether each shard's graph properties influence its importance score. To this end, we extract each shard's embedding by averaging all its nodes' embeddings obtained from the pretrained GNN model and project the shard embedding

into a two-dimensional space using t-distributed stochastic neighbor embedding (t-SNE) [99]. The results are plotted in Figure 3.8. Each circle represents the mean node



(a) GraphEraser-BLPA



(b) GraphEraser-BEKM

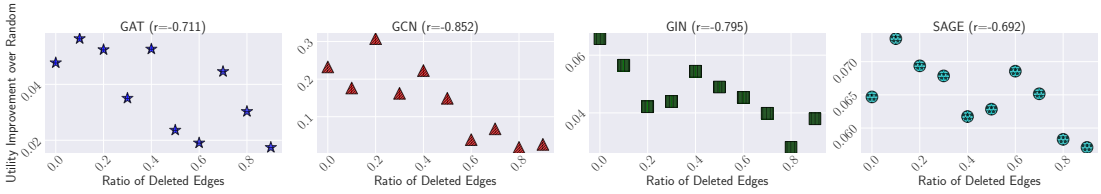
Figure 3.8: The t-SNE plot of shard embeddings for the Cora dataset.

embeddings of a shard, where the circle size is proportional to its importance score in annotations. As we can see, shards with more significant importance scores are typically accompanied by shards with more miniature importance scores. This implies that for shards trained on similar graphs (similar shard embeddings in the two-dimensional space), our learning-based aggregation assigns a higher score to one of them. In another way, it also learns to discard redundant information to improve utility.

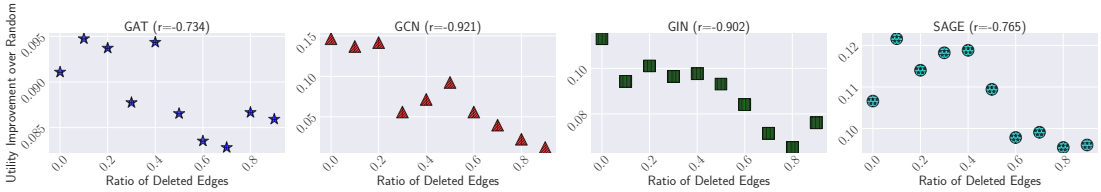
3.5.3.2 Role of Graph Structure

To better illustrate the correlation between the importance of the graph structure and the utility improvement of GraphEraser over Random (SISA), we performed another experiment on Cora and Citeseer. Concretely, we delete different fractions of edges from the training graph to model the significance of the graph structure and then

compare the performance gap between GraphEraser and Random. Figure 3.9 illustrates the experimental results. We vary the ratio of deleted edges (as shown in the x-axis) from 0% to 90% with a step of 10%. A higher deletion ratio reduces graph structure information. And the y-axis represents the utility improvement of GraphEraser over Random. Although there are some outliers, the overall trend (measured by the Pearson correlation score above each sub-figure) shows that when the graph structure is more important, the utility improvement of GraphEraser over Random is more significant in most of the cases.



(a) Cora



(b) Citeseer

Figure 3.9: Correlation between the importance of the graph structure (larger ratio of edge deletion indicates graph structure is less important) and the utility improvement of GraphEraser over Random.

3.5.3.3 Comparison with Existing Balanced Graph Partition Solutions

In this section, we empirically compare GraphEraser with existing solutions for balanced graph partitioning [163, 87, 100] regarding running time and model utility.

Balanced Graph Partitioning. As discussed in Section 3.5.3.3, the existing balanced graph partitioning algorithms can be broadly classified into three categories. The first two categories adopt **Strategy 1** in Section 3.2 that only considers graph structural information. The third category adopts **Strategy 2** in Section 3.2 that considers both graph structural and node feature information.

Recall that in Section 3.2.1, community detection can inherently preserve graph structural information with the cost of unbalanced partitioning. Thus, the first category of previous studies aims to modify existing community detection methods to satisfy

balanced community size constraints. In BLPA-LP [163], the authors formulate the label propagation process as a linear programming problem to satisfy the community size constraints.

On the other hand, previous studies in the second category do not rely on community detection. Instead, they directly partition the graph by optimizing predefined criteria, such as minimizing the graph cut [142, 38] or maximizing the graph modularity [113, 55]. However, these optimization problems are always NP-hard and cannot be solved exactly; thus, the researchers proposed many approximate or intuitive algorithms. Spectral graph partitioning [103, 190] is a widely adopted approach. The general idea is first to calculate the Laplacian matrix of the graph, then calculate the eigenvectors of the Laplacian matrix. Each node is mapped to one of the eigenvalues in the second smallest eigenvector, and the sign of the corresponding eigenvalues defines the graph partition. One can conduct the spectral graph partitioning method hierarchically to partition the graph into multiple shards. The main drawback of the spectral methods is they cannot deal with large-scale graphs. A promising solution for large-scale graph partitioning is utilizing the multilevel graph partitioning methods. The general idea is first to contract edges and obtain smaller graphs, then cut the resulting graph, and finally unfold back to the original graph with some local improvement criterion [14, 16, 72]. Among the multilevel graph partitioning methods, METIS [80, 87] is a family of the most widely known techniques and achieves state-of-the-art performance [17].

The general idea of the third category is first to transform the attributed graph into node embeddings and use balanced clustering methods to cluster the node embeddings. In BEKM-Hungarian [100], the authors modify the reassignment step of the k-means algorithm to achieve balanced clusters. The core idea is to formulate the node reassignment problem as a matching problem which is approximately solved by the Hungarian algorithm. In [93], the authors propose to use linear regression to estimate the class-specific hyperplanes that partition each class of the data point from others. A soft balanced constraint is enforced to achieve balanced clustering. The drawback of this method is that we cannot precisely control the cluster size.

Competitors. Existing balanced graph partition algorithms can be broadly classified into three categories: The first category considers only the graph structural information and relies on community detection as GraphEraser-BLPA. The second category considers the graph structural information without relying on community detection. The third category considers both structural information and node features as GraphEraser-BEKM. For each category, we choose one most representative method as a competitor, and we

list the details as follows.

- **BLPA-LP [163].** Similar to our proposed GraphEraser-BLPA, this method achieves a balanced graph partition by constraining the label propagation process. The general idea is to formulate the label propagation process as a linear programming problem with $2k(k-1)$ variables and $2k^2 + nk(k-1)$ constraints, where n and k are the number of nodes and the number of shards, respectively. When the size of the graph and the number of shards are large, solving the linear programming problem is time-consuming.

- **METIS [87].** The objective of METIS is to obtain a balanced graph partition while cutting the minimum number of edges. The computational complexity of METIS is $\mathcal{O}((n+m)\cdot \log k)$, where m is the number of edges. We implement this method with official METIS 5.1.0[2] and a Python wrapper[3] for METIS library.

- **BEKM-Hungarian [100].** BEKM-Hungarian shares the general idea of our GraphEraser-BEKM. The main difference is that it has a different mechanism in the node reassignment step for achieving balanced k-means. Concretely, BEKM-Hungarian formulates the node reassignment step as a matching problem and is approximately solved by the Hungarian algorithm. The computational complexity of the Hungarian algorithm is $\mathcal{O}(n^3)$.

Results. Table 3.6 and Table 3.7 illustrate the model utility and graph partitioning efficiency for different methods. We apply LBAggr for all the graph partitioning methods for a fair comparison.

In general, graph partitioning methods rely on both graph structural information and node features. i.e., GraphEraser-BEKM and BEKM-Hungarian, achieve the best model utility when the target model is GAT, GIN, and SAGE, which is consistent with the conclusion of Section 3.5.2.2. Comparing GraphEraser-BEKM and BEKM-Hungarian, we observe that they achieve similar model utility; however, the computational complexity of BEKM-Hungarian ($\mathcal{O}(n^3)$) is much higher than that of GraphEraser-BEKM ($\mathcal{O}(k\cdot n)$). From Table 3.7, we also observe that BEKM-Hungarian is not scalable to large graphs.

When the target model is GCN, the community detection-based methods, i.e., GraphEraser-BLPA and BLPA-LP, achieve a better model utility than the minimum-cut-based method (METIS). We suspect this is because the GCN model requires the

[2] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
[3] https://github.com/inducer/pymetis

Table 3.6: Comparison of F1 scores for different graph partition methods.

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

node degree information for normalization, and the community detection-based methods can preserve more local structural information, thus better preserving the node degree. Comparing GraphEraser-BLPA and BLPA-LP, GraphEraser-BLPA is more efficient than BLPA-LP (see Table 3.7) while achieving comparable model utility.

Remarks. GraphEraser is a general framework for GNN unlearning; any balanced graph partitioning method which meets the requirements in Section 3.1.1 can be adopted. Therefore, we encourage the community to develop more efficient and better-performing balanced graph partitioning algorithms for graph unlearning.

We generally observe that the graph partitioning methods rely on both graph structural information and node features. i.e., GraphEraser-BEKM and BEKM-Hungarian, achieve the best model utility when the target model is GAT, GIN, and SAGE, which is consistent with the conclusion of Section 3.5.2.2. Comparing GraphEraser-BEKM and BEKM-Hungarian, we observe that they achieve similar model utility; however, the computational complexity of BEKM-Hungarian ($\mathcal{O}(n^3)$) is much higher than that of GraphEraser-BEKM ($\mathcal{O}(k \cdot n)$). From Table 3.7, we can see that BEKM-Hungarian is not scalable to large graphs.

When the target model is GCN, the community detection-based methods, i.e.,

Table 3.7: Comparison of graph partition efficiency for different balanced graph partition methods.

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

GraphEraser-BLPA and BLPA-LP, achieve better model utility than the minimum-cut-based method (METIS). We suspect this is because the GCN model requires the node degree information for normalization, and the community detection-based methods can preserve more local structural information, thus better preserving the node degree. Comparing GraphEraser-BLPA and BLPA-LP, GraphEraser-BLPA is more computationally efficient than BLPA-LP (see Table 3.7) while achieving comparable model utility.

Remarks. GraphEraser is a general framework for GNN unlearning; any balanced graph partitioning method which meets the requirements in Section 3.1.1 can be considered. Therefore, we encourage the research community to develop more efficient and better-performing balanced graph partitioning algorithms for the graph unlearning application.

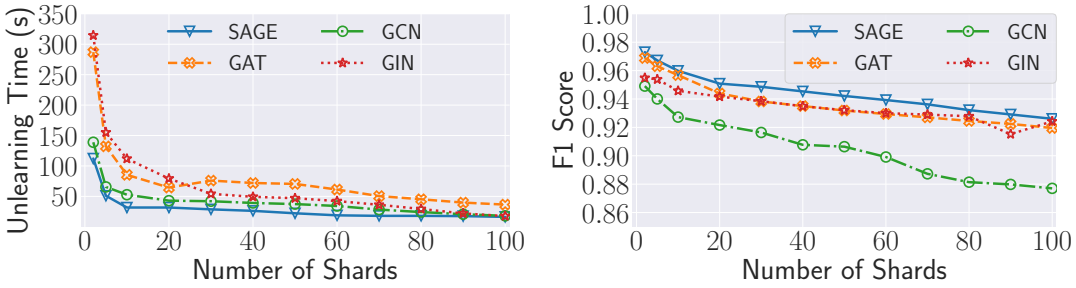
3.5.4 Hyperparameters

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

Number of Shards k. We conduct the experiments on the Physics dataset with four GNN models. We vary the number of shards from 2 to 100. As suggested in Section 3.5.2.2, we apply GraphEraser-BEKM for GIN, GAT, and SAGE, and GraphEraser-BLPA for GCN.

The experimental results in Figure 3.10 show that the average unlearning time cost decreases when the number of shards increases for all the GNN models. This is expected since a larger number of shards means a smaller shard size, leading to higher unlearning efficiency. On the other hand, the F1 score of all four GNN models slightly decreases. Comparing the four GNN models, the utility of GCN model drops the most. We suspect this is because the GCN model requires the node degree information for normalization, which is severely reduced by graph partitioning. The number of shards is an important hyperparameter for GraphEraser. In practice, it should be selected based on the size of

the training graph.



(a) Unlearning Efficiency

(b) Target Model Utility

Figure 3.10: Impact of the number of shards on the unlearning efficiency and model utility on the Physics dataset.

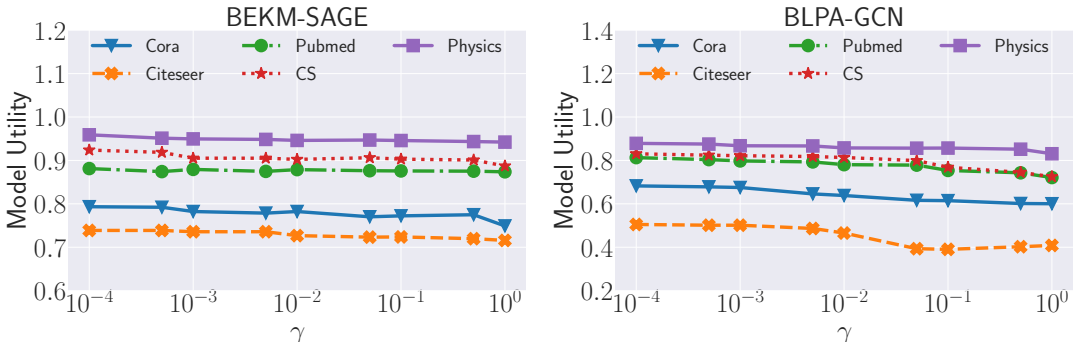
Maximum Number of Nodes in Each Shard δ. It is an important parameter in both GraphEraser-BLPA and GraphEraser-BEKM for controlling the degree of balance of the partitioned graphs. The minimum value of δ is $\lceil \frac{n}{k} \rceil$, in which case the shards are balanced. The maximum value of δ is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

Intuitively, we aim to make δ as close as $\lceil \frac{n}{k} \rceil$ to achieve balanced shards for efficiency. The remaining concern is what is the impact of δ on the model utility. We conduct experiments for both GraphEraser-BLPA and GraphEraser-BEKM on five datasets. To achieve comparable experiments across different datasets, we introduce a scaling parameter γ in the range of $[0, 1]$ to regulate the choice of δ, i.e., $\delta = \lceil \frac{n}{k} \rceil + \gamma \cdot (n - \lceil \frac{n}{k} \rceil)$. When $\gamma = 0$, δ equals to $\lceil \frac{n}{k} \rceil$, which is the lower bound of δ; when $\gamma = 1$, δ equals to n, which is the upper bound of δ. Figure 3.11 illustrates the experimental results. In general, we observe that δ only has a slight impact on the model utility; thus, we set $\delta = \lceil \frac{n}{k} \rceil$ for all of our experiments which leads to the best efficiency.

3.5.5 Robustness of GraphEraser

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

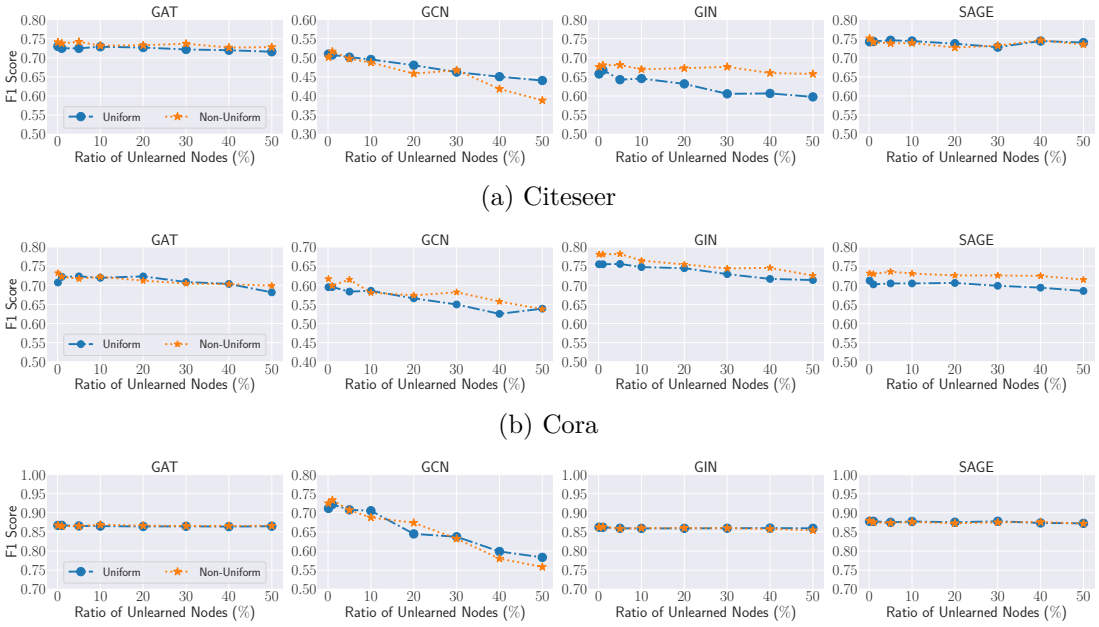
Figure 3.12 illustrates the experimental results on three datasets and we evaluate both uniform and non-uniform unlearning request distribution. We first observe that



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

Figure 3.11: Impact of δ on GraphEraser-BEKM and GraphEraser-BLPA for five datasets.

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



(a) Citeseer

(b) Cora

(c) Pubmed

Figure 3.12: Impact of the ratio of unlearned nodes on the model utility.

Table 3.8: Attack AUC of membership inference against our GraphEraser ($\mathcal{A}_I$) and deterministic unlearning ($\mathcal{A}_{II}$).

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

3.5.6 Unlearning Power of GraphEraser

Since our method is highly empirical, we adopt the state-of-the-art attack against machine unlearning [P2] to quantify the extra information leakage of GraphEraser when the graph is not re-partitioned. In particular, Chen et al. [P2] showed that the attackers, using an enhanced membership inference attack [145], can determine whether a target sample exists in the original model and is revoked from the unlearned model when they have access to both the original and unlearned model. Here we quantify the extra information leakage of GraphEraser as *the attack's performance difference between deterministic unlearning and* GraphEraser *unlearning*. Concretely, we introduce two scenarios of membership inference attacks. We start from the same set of original shard models. In scenario 1, the unlearned models are obtained by directly deleting the revoked nodes from the corresponding shard graph and retraining the corresponding shard models. This is how GraphEraser generates the unlearned models. In scenario 2, we retrain from scratch (re-partition the graph and train a set of new shard models). This type of unlearning deterministically unlearns every component while it is extremely time-consuming. Denoting the two scenarios as $\mathcal{A}_I$ and $\mathcal{A}_{II}$, the extra information leakage is the difference of the attack AUC between $\mathcal{A}_I$ and $\mathcal{A}_{II}$. We use the implementation[4] of [P2] to conduct our experiments. The experimental results in Table 3.8 show that the attack AUC of both $\mathcal{A}_I$ and $\mathcal{A}_{II}$ are close to 0.5 (random guessing), meaning that GraphEraser does not leak much extra information. This is also consistent with the observation of [P2] that the membership inference performs badly on the SISA-based method since the aggregation reduces the influence of a specific sample on its global model.

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

3.6 Discussion

Guarantee to the right-to-be-forgotten. The adversary might recover the deleted data when they can access both the original and the unlearned model. Previous work studied the unintended information leakage in the unlearning setting [21, P2]. However, it is orthogonal to our work since the primary goal of machine unlearning is to comply with "legitimate regulations" such as GDPR. In this sense, as long as the model is trained without the revoked sample, the requirement of the right to be forgotten is satisfied. One may argue that the graph partition depends on the node to be deleted; however, it is unclear how adversaries can exploit this information. Besides, the graph partition of SISA [25] also depends on the node to be deleted since they rely on prior knowledge to put the more likely-to-be-deleted samples in one shard.

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

Adaptive Machine Unlearning. The authors in [61] define the notion of (α, β, γ)-unlearning, which enforces that the output of any unlearning algorithm should be similar to that of retraining from scratch. The authors prove that the SISA method satisfies (α, β, γ)-unlearning in the *non-adaptive* setting (unlearning requests arrive in a non-adaptive way), but it does not satisfy (α, β, γ)-unlearning in the *adaptive* setting. As GraphEraser follows the general idea of SISA, GraphEraser also satisfies (α, β, γ)-unlearning in the non-adaptive setting but does not satisfy the adaptive setting.

3.7 Conclusion

In this chapter, we aim to design an unlearning framework for graph neural networks that can achieve "High Unlearning Efficiency" and "Comparable Model Utility" for retraining from scratch. We propose the first machine unlearning framework GraphEraser in the context of GNNs. Concretely, GraphEraser first partitions the training data into several small groups, trains a series of submodels, then generates the final prediction by aggregating the predictions from all the submodels. Only the affected shard model

will be retrained in randomly unlearning request cases to achieve the high unlearning efficiency goal. To achieve efficient retraining while keeping the structural information of the graph, we propose a general principle for balancing the shards resulting from the graph partitioning and instantiate it with two novel balanced graph partition algorithms, Balanced Embedding K-Means (BEKM) and Balanced Label Propagation Algorithm (BLPA). We further propose a learning-based aggregation method to improve the model's utility.

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

4

Understanding the Privacy Risks in Machine Unlearning

In this chapter, we investigate the unintended information leakage caused by machine unlearning. We first propose a novel membership inference attack in the machine unlearning setting that determines whether the target sample is part of the training set of the original model. Different from classical membership inference attacks [145, 132], which leverage the output (posteriors) of a single target model, our attack leverages outputs of both original and unlearned models. To solve this challenge, we need to answer the following research questions: First, whether our proposed membership inference attack can work, and how bad this privacy risk can be. Second, whether the privacy leakage still exists in different unlearning scenarios, such as Third, whether this privacy risk can be mitigated.

In the following of this chapter, we first present the details of privacy risks of machine unlearning in Section 4.1. Next, we evaluate the attack performance on multiple models, datasets, and practical scenarios in Section 4.2. In Section 4.3, we introduce several possible defense mechanisms and empirically evaluate their effectiveness. Finally, we conclude the chapter in Section 4.4.

4.1 Membership Inference in Machine Unlearning

In this section, we study to what extent data is indelibly imprinted in an ML model by quantifying the additional information leakage caused by machine unlearning.

4.1.1 Problem Statement

Machine unlearning naturally generates two versions of ML models, namely the *original model* and the *unlearned model*, and creates a discrepancy between them due to the target sample's deletion. While originally designed to protect the target sample's privacy, we argue that machine unlearning may leave some imprint of it and thus create unintended privacy risks. Specifically, while the original model may not reveal much private information about the target sample, additional information might be leaked through the unlearned model.

4.1.2 Threat Model

When implementing the right to be forgotten in machine learning models, a common acknowledgment of forgetting is that the revoked sample and its corresponding impacts on the model should be erased, which can also be named "machine unlearning". There

are two lines of work to implementing machine unlearning, approximate unlearning and deterministic unlearning. We focus on deterministic unlearning, which means the model is indeed trained without the revoked samples, which is the original requirement of the right to be forgotten. While originally designed to protect the privacy of the data owner, we argue that machine unlearning may leave some imprint of the data in the ML model and thus create unintended privacy risks.

Adversary's Goal:. Given a target sample x, an original model, and its unlearned model, the adversary aims to infer whether x is unlearned from the original model. In other words, the adversary aims to know that the target sample is in the training dataset of the original model, but it is not in the training dataset of the unlearned model. While the goal of unlearning x is to protect x's privacy, a successful attack considered here can show unlearning instead jeopardizes x's privacy (especially when x's membership leakage risk is not severe on the original model before machine unlearning).

Adversary's Impacts:. Knowing that a specific data sample x was used to train a particular model may lead to potential privacy breaches. For example, knowing that a certain patient's clinical records were used to train a model associated with a disease (e.g., to determine the appropriate drug dosage or to discover the genetic basis of the disease) can reveal that the patient carries the associated disease.

Adversary's Knowledge:. Unlike classical membership inference, which only leverages the output of a target ML model, our adversary can exploit information from both original and unlearned models to perform their attack. We assume that the adversary has black-box access to an original ML model and its unlearned model. This is realistic as the target black-box model can be queried at any time, such as in the setting of MLaaS, and all of the query results can be stored locally by the adversary. As such, when there are no changes in the target sample's outputs from two consecutive queries, the unlearning does not happen, and the adversary does not need to launch the attack. On the other hand, when the adversary observes changes in the target sample's outputs, they know that the target model has been updated.

4.1.3 Attack Pipeline

The general attack pipeline comprises three phases: posteriors generation, feature construction, and (membership) inference. **Posteriors Generation:** The adversary has access to two versions of the target ML model, the original model $\mathcal{M}_o$ and the unlearned model $\mathcal{M}_u$. Given a target sample x, the adversary queries $\mathcal{M}_o$ and $\mathcal{M}_u$,

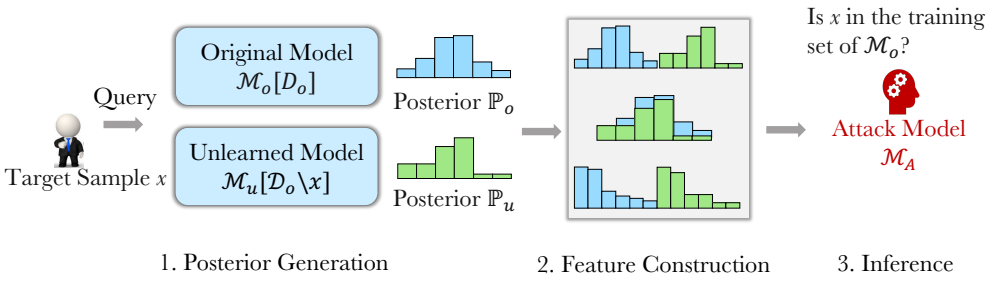


Figure 4.1: A schematic view of the general attack pipeline. The membership status of the target sample x is leaked due to the two versions of the model.

and obtains the corresponding posteriors, i.e., $\mathbb{P}_o$ and $\mathbb{P}_u$. **Feature Construction:** Given the two posteriors $\mathbb{P}_o$ and $\mathbb{P}_u$, the adversary aggregates them to construct the feature vector $\mathcal{F}$. We discuss five representative methods to construct the feature vector, including direct concatenate, sorted concatenate, direct difference, sorted difference, and Euclidean distance. **Inference:** Finally, the adversary sends the obtained $\mathcal{F}$ to the attack model, which is a binary classifier, to determine whether the target sample x is in the training set of the original model.

4.1.4 Attack Model Training

We assume the adversary has a local dataset, which we call the *shadow dataset* $\mathcal{D}^s$. The shadow dataset can come from a different distribution than the one used to train the target model. To infer whether the target sample x is in the original model or not, our core idea is to train an attack model $\mathcal{M}_A$ that captures the difference between the two posteriors. The intuition is that if the target sample x is deleted, the two models $\mathcal{M}_o$ and $\mathcal{M}_u$ will behave differently. Figure 4.2 illustrates the training process of the attack model, and the detailed training procedure is presented as follows. The shadow dataset $\mathcal{D}^s$ is split into disjoint shadow positive dataset $\mathcal{D}^s_p$ and shadow negative dataset $\mathcal{D}^s_n$. The shadow positive dataset $\mathcal{D}^s_p$ is used to train the shadow original model $\mathcal{M}^s_o$. The shadow unlearned model $\mathcal{M}^{s,i}_u$ is trained on $\mathcal{D}^s_p \setminus x^i_p$, where $x^i_p \in \mathcal{D}^s_p$. In the inference phase, the adversary first uses target sample x^i_p to query the original and unlearned models simultaneously to generate the positive features. Then they use a random sample $x^i_n \in \mathcal{D}^s_n$ to query the corresponding models to generate the negative features. Finally, they use the positive and negative features to train the attack model $\mathcal{M}_A$.

Training Shadow Models. To mimic the behavior of the target model, the adversary

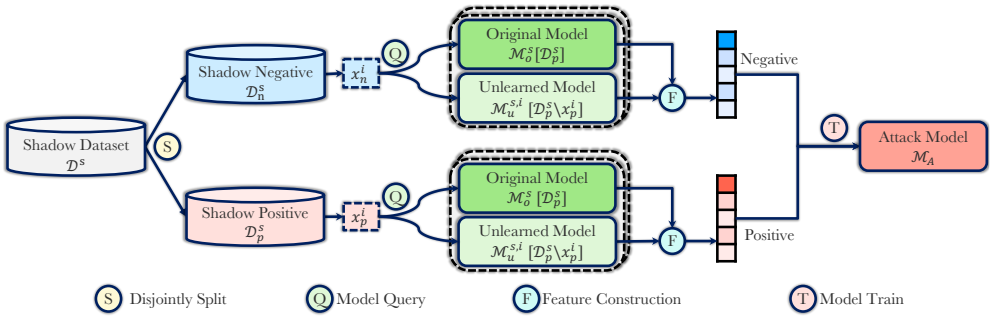


Figure 4.2: Training process of the attack model.

needs to train a shadow original model and a set of shadow unlearned models. To do this, the adversary first partitions $\mathcal{D}^s$ into two disjoint parts, the shadow negative set $\mathcal{D}_n^s$ and the shadow positive set $\mathcal{D}_p^s$. The shadow positive set $\mathcal{D}_p^s$ is used to train the shadow original model $\mathcal{M}_o^s$. The shadow unlearned model $\mathcal{M}_u^s$ is trained by deleting samples from $\mathcal{D}_p^s$. For ease of presentation, we assume the shadow unlearned model $\mathcal{M}_u^s$ is obtained by deleting exactly one sample. We will show that our attack is still effective for group deletion in Section 4.2.7.2. The adversary randomly generates a set of deletion requests (target samples) $\mathcal{R}_p = \{x_p^1, x_p^2, \cdots, x_p^m\}$ and train a set of shadow unlearned models $\mathcal{M}_u^{s,1}, \mathcal{M}_u^{s,2}, \cdots, \mathcal{M}_u^{s,m}$, where the shadow unlearned model $\mathcal{M}_u^{s,i}$ is trained on dataset $\mathcal{D}_p^s \setminus x_p^i$.

Obtaining Posteriors. At the posteriors generation phase, the adversary feeds each target sample $x_p^i \in \mathcal{R}_p$ to the shadow original model $\mathcal{M}_o^s$ and its corresponding shadow unlearned model $\mathcal{M}_u^{s,i}$, and gets two posteriors $\mathbb{P}_o^i$ and $\mathbb{P}_u^i$.

Constructing Features. The adversary then uses the feature construction methods discussed in Section 4.1.5 to construct training cases for the attack model. In classical membership inference, posteriors of $x_p^i \in \mathcal{R}_p$ serve as member cases of the attack model. But in the machine unlearning setting, $x_p^i \in \mathcal{R}_p$ is a member of the shadow original model $\mathcal{M}_o^s$ and non-member of the shadow unlearned model $\mathcal{M}_u^s$. To avoid confusion, we call the samples related to $x_p^i \in \mathcal{R}_p$ *positive* cases instead of member cases for the attack model.

To train the attack model, the adversary also needs a set of negative cases. This can be done by sampling a set of negative query samples $\mathcal{R}_n$ from the shadow negative dataset $\mathcal{D}_n^s$ and query the shadow original model and unlearned model. To get a good attack model generalization performance, the adversary needs to ensure that the number of positive cases and the number of negative cases of the attack model are balanced,

i.e., $|\mathcal{R}_p| = |\mathcal{R}_n|$, where $|\cdot|$ is the cardinality of the sample set.

4.1.5 Feature Construction

Given the two posteriors, a straightforward approach to aggregate the information is to concatenate them, i.e., $\mathbb{P}_o||\mathbb{P}_u$, where $||$ is the concatenation operation. This preserves the full information. However, it is possible that the concatenation contains redundancy. To reduce redundancy, we can instead rely on the difference between $\mathbb{P}_o$ and $\mathbb{P}_u$ to capture the discrepancy left by the deletion of the target sample. In particular, we make use of the element-wise difference $\mathbb{P}_o - \mathbb{P}_u$ and the Euclidean distance $\|\mathbb{P}_o - \mathbb{P}_u\|_2$.

To better capture the level of confidence of the model, one may also sort the posteriors before the difference or concatenation operations [49]. Specifically, we sort the original posteriors $\mathbb{P}_o$ in descending order and get the sorted original posteriors $\mathbb{P}_o^s$. We then rearrange the order of the unlearned posteriors $\mathbb{P}_u$ to align its elements with $\mathbb{P}_o$ and get the sorted unlearned posteriors $\mathbb{P}_u^s$.

To summarize, we adopt the following five methods to construct the features for the attack model:

- Direct concatenate (DirectConcat), i.e., $\mathbb{P}_o||\mathbb{P}_u$
- Sorted concatenate (SortedConcat), i.e., $\mathbb{P}_o^s||\mathbb{P}_u^s$
- Direct difference (DirectDiff), i.e., $\mathbb{P}_o - \mathbb{P}_u$.
- Sorted difference (SortedDiff), i.e., $\mathbb{P}_o^s - \mathbb{P}_u^s$.
- Euclidean distance (EucDist), i.e., $\|\mathbb{P}_o - \mathbb{P}_u\|_2$

In Section 4.2.3.2, we conduct empirical experiments to evaluate the performance of the above methods and provide a high-level summary of the best features to use depending on the behavior of the underlying ML model.

4.2 Privacy Degradation Measurement

In this section, we conduct extensive experiments to evaluate the unintended privacy risks of machine unlearning. In Section 4.2.3.1, we first conduct an end-to-end experiment to validate the effectiveness of our attack on multiple datasets using the most straightforward unlearning method, i.e., retraining from scratch. Second, we compare different feature construction methods proposed in Section 4.2.3.2 and summarize the

most appropriate choice depending on the context. We evaluate the impact of overfitting and different hyperparameters in Section 4.2.3.3. We conduct experiments to evaluate the dataset and model transferability between the shadow model and the target model in Section 4.2.5. We also show the effectiveness of our attack against the SISA unlearning method in Section 4.2.6. Third, we evaluate our attack in multiple practical scenarios to Section 4.2.7. Finally, we evaluate the attack performance of our attack under four defense mechanisms in Section 4.3.

4.2.1 Experimental Setup

We evaluate our proposed membership inference attack in seven datasets and seven models. The experimental results show that our attack in multiple cases outperforms the classical membership inference attack, which indicates that machine unlearning can have counterproductive effects on privacy.

Target Models. In our experiments, we evaluate the vulnerability of both simple machine learning models, including logistic regression (LR), decision tree (DT), random forest (RF), and 5-layer multi-layer perceptron (MLP), and the state-of-the-art convolutional neural networks, including SimpleCNN (implemented by us), DenseNet [71], and ResNet50 [66]. All the convolutional networks are trained for 100 epochs.

Datasets. We run experiments on two different types of datasets: categorical datasets and image datasets. The categorical datasets are used to evaluate the vulnerability of simple machine learning models, while the image datasets are used to evaluate the vulnerability of the convolutional neural networks.

- **UCI Adult [7].** This is a widely used categorical dataset for classification. It is a census dataset that contains around $50,000$ samples with 14 features. The classification task is to predict whether the income of a person is over $\$50k$, which is a binary classification task.

- **US Accident [8].** This is a countrywide traffic accident dataset, which covers the 49 states of the United States. This dataset contains around 3M samples. We filter out attributes with too many missing values and obtain 30 valid features. The valid features include temperature, humidity, pressure, etc. The classification task is to predict the accident severity level, which contains 3 classes.

- **Insta-NY [18].** This dataset contains a collection of Instagram users' location check-in data in New York. Each check-in contains a location and a timestamp, and

each location belongs to a category. We use the number of check-ins that happened at each location in each hour on a weekly basis as the location feature vector. The classification task is to predict each location's category among 9 different categories. After filtering out locations with less than 50 check-ins, we get 19,215 locations for the Insta-NY dataset. Later in the section, we also use check-ins in Los Angeles, namely Insta-LA [18], for evaluating the data transferring attack. This dataset includes 16,472 locations.

- **MNIST [9].** MNIST is an image dataset widely used for classification. It is a 10-class handwritten digits dataset that contains 42,000 samples, each being formatted into a 28×28-pixel image.

- **CIFAR10 [10].** CIFAR10 is the benchmark dataset used to evaluate image recognition algorithms. This dataset contains 60,000 colored images of size 32×32, which are equally distributed on the following 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. There are 50,000 training images and 10,000 testing images.

- **STL10 [36].** STL10 is a 10-class image dataset, with each class containing 1,300 images. Classes include airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck. In our experiments, we use STL10 only for data transferring attacks.

Experimental Settings. We evenly split each dataset $\mathcal{D}$ into disjoint target dataset $\mathcal{D}^t$ and shadow dataset $\mathcal{D}^s$. In Section 4.2.5, we will show that the shadow dataset can come from a different distribution than the target dataset. The shadow dataset $\mathcal{D}^s$ is further split into shadow positive dataset $\mathcal{D}^s_p$ and shadow negative dataset $\mathcal{D}^s_n$ (80% for $\mathcal{D}^s_p$ and 20% for $\mathcal{D}^s_n$). We randomly sample S_o subsets of samples from $\mathcal{D}^s_p$, each containing S_r samples, to train S_o shadow original models. For each shadow original model $\mathcal{M}^{s,i}_o$, we train S_u shadow unlearned models on $\mathcal{D}^{s,i}_o \setminus x$. We split the target dataset $\mathcal{D}^t$ in a similar way as the shadow dataset $\mathcal{D}^s$.

By default, we set the hyperparameters of the shadow models to $S_o = 20, S_r = 5000, S_u = 100$, and the corresponding hyperparameters of the target models to $T_o = 20, T_r = 5000, T_u = 100$. These hyperparameters have been shown to achieve a good balance between computational overhead and attack performance in Section 4.2.4.

4.2.2 Privacy Degradation Metrics

We propose two privacy degradation metrics that measure the difference in the confidence levels of our attack and classical membership inference when predicting the correct membership status of the target sample. Given n target samples x^1 to x^n, define p_u^i as the confidence of our attack in classifying x^i as a member, and p_m^i as the confidence of classical membership inference. Let b^i be the true status of x^i, i.e., $b^i = 1$ if x^i is a member, and $b^i = 0$ otherwise.

In addition to the two privacy degradation metrics, we rely on the traditional AUC metric to measure the absolute performance of our attack and classical membership inference. To summarize, we have the following three metrics:

- **DegCount.** DegCount stands for Degradation Count. It calculates the proportion of target samples whose true membership status is predicted with higher confidence by our attack than by classical membership inference. Formally, DegCount is defined as

$$DegCount = \frac{1}{n}\sum_{i}^{n}\left[b^i \mathbb{1}_{p_u^i > p_m^i} + (1-b^i)\mathbb{1}_{p_u^i < p_m^i}\right]$$

  where $\mathbb{1}_P$ is the indicator function which equals 1 if P is true, and 0 otherwise. Higher DegCount means higher privacy degradation.

- **DegRate.** DegRate stands for Degradation Rate. It calculates the average confidence improvement rate of our attack predicting the true membership status compared to classical membership inference. DegRate can be formally defined as

$$DegRate = \frac{1}{n}\sum_{i}^{n}\left[b^i(p_u^i - p_m^i) + (1-b^i)(p_m^i - p_u^i)\right]$$

  Higher DegRate means higher privacy degradation.

- **AUC.** It is a widely used metric to measure the performance of binary classification in a range of thresholds. An AUC value of 1 shows a maximum performance, while an AUC value of 0.5 shows a performance equivalent to random guessing.

4.2.3 Overall Performance

4.2.3.1 Retrain from Scratch

In this subsection, we conduct end-to-end experiments to evaluate our attack against the most straightforward approach of retraining the ML model from scratch.

Setup. We start by considering the scenario where only one sample is deleted for each unlearned model. The scenario where multiple samples are deleted before the ML model is retrained will be evaluated in Section 4.2.7.2. We conduct the experiments on both categorical datasets and image datasets with three evaluation metrics, namely AUC, DegCount, DegRate, and report the results with the optimal features as explained in Section 4.2.3.2.

Results for Categorical Datasets. Figure 4.3 depicts the attack performance of categorical datasets. In each subfigure, the groups on the x-axis represent different target models, and the legends (in different colors) represent different attack models. For the AUC metric, the right bars (transparent ones) stand for the AUC value of classical membership inference. We generally observe that our attack performs consistently better than classical membership inference on all datasets, target models, attack models, and metrics. Compared to classical membership inference, our attack achieves up to 0.48 improvement of the AUC. The best DegCount and DegRate values are 0.94 and 0.40, respectively. This indicates that our attack indeed degrades the membership privacy of the target sample in the machine unlearning setting. Comparing the performance of different target models, we observe that the decision tree is the most vulnerable ML model. We posit this is because the decision tree forms a tree structure, and deleting one sample could explicitly change its structure; thus, the posterior difference between the decision tree's original model and the unlearned model is more significant, leading to a better attack performance.

Results for Image Datasets. Figure 4.4 illustrates the performance for the image datasets and complex convolutional neural networks. We keep the same attack models as categorical datasets and use the SimpleCNN model for MNIST, use the ResNet50 and DenseNet models for CIFAR10. In general, we also observe that our attack outperforms classic membership inference attacks in all settings. Besides, CIFAR10 trained with DenseNet shows the highest privacy degradation, while the MNIST dataset trained with SimpleCNN shows the lowest. The reason behind this is that the overfitting level of CIFAR10 trained with DenseNet is the largest. To further confirm this, we list the

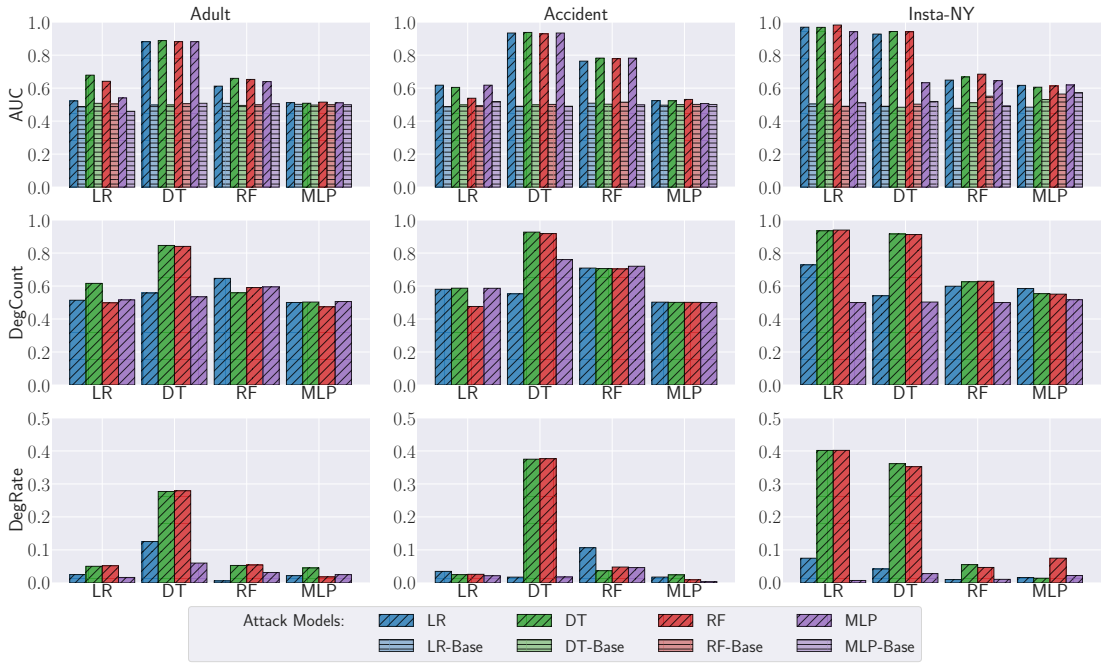


Figure 4.3: Privacy degradation level on the Scratch method for three categorical datasets.

overfitting level of different models in Table 4.1. We observe that the overfitting level of CIFAR10 trained with DenseNet is 0.439, while the MNIST dataset trained with SimpleCNN has an overfitting level smaller than 0.05.

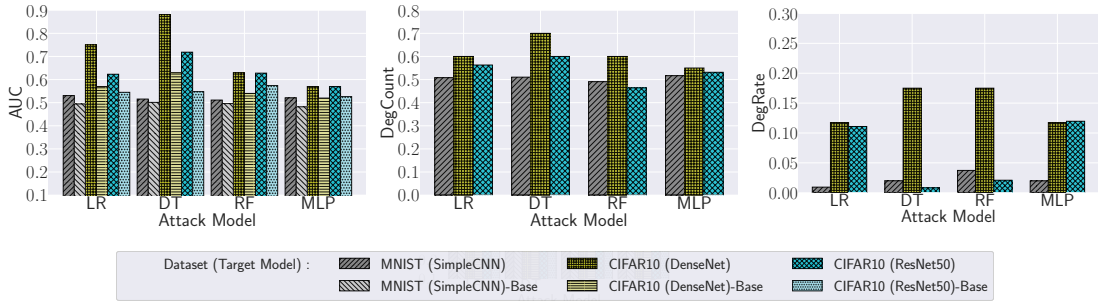
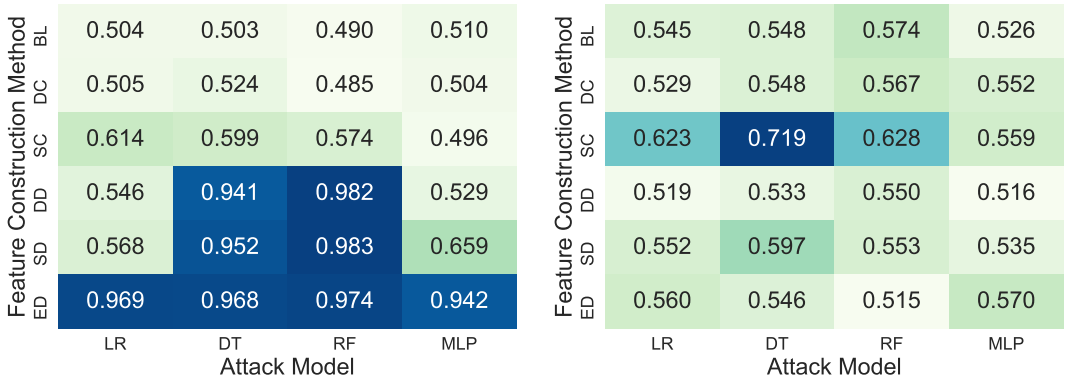


Figure 4.4: Privacy degradation level on the Scratch method for image datasets.

4.2.3.2 Finding Optimal Features

Figure 4.5 illustrates the attack AUC of different feature construction methods. We compare two different types of target models: (a) the well-generalized model logistic regression (trained on the Insta-NY dataset), and (b) the overfitted model ResNet50

(trained on the CIFAR10 dataset). We then apply the 5 different feature construction methods proposed in Section 4.1.5 to 4 different attack models, resulting in 20 combinations. For comparison, we also include the classical membership inference as a baseline. DC, SC, DD, SD, ED stand for DirectConcat, SortedConcat, DirectDiff, SortedDiff, EucDist, respectively. BL stands for the baseline, i.e., classical membership inference.



(a) LR + Insta-NY

(b) ResNet50 + CIFAR10

Figure 4.5: Attack AUC for different feature construction methods.

Concatenation vs. Difference. Concatenation-based methods (DirectConcat, SortedConcat) directly concatenate the two posteriors to preserve the full information, while difference-based methods capture the discrepancy between two versions of posteriors. We use two approaches to capture this discrepancy: element-wise difference (DirectDiff, SortedDiff) and Euclidean distance (EucDist).

Overall, Figure 4.5 shows that, on the one hand, concatenation-based methods perform better on the overfitted model, i.e., ResNet50. On the other hand, the difference-based methods perform better on the well-generalized model, i.e., logistic regression. We suspect this is due to the fact that the concatenation-based methods rely on plain posterior information, which can provide a strong signal for membership inference on the overfitted target model. This is consistent with the conclusion of previous studies [145, 132] that classical membership inference (which uses plain posterior information) performs well on overfitted target models. While we can also exploit the difference-based methods to mount the attack on the overfitted target models, the attack signal is not as strong as that of the concatenation-based methods, as shown in Figure 4.5b. For the well-generalized target models, exploiting the plain posterior information has shown to perform poorly in terms of membership inference [145, 132].

In this case, the discrepancy information between the two versions of the posteriors captured by the difference-based methods is more informative than the concatenation-based feature construction methods.

Sorted vs. Unsorted. Comparing DirectConcat to SortedConcat and DirectDiff to SortedDiff in Figure 4.5, we observe that the attack AUC of both the concatenation-based method and difference-based method are clearly better after sorting. These results confirm our conjecture that sorting could improve the confidence level of the adversary.

Feature Selection Summary. Our empirical comparison provides us with the following rules for the feature construction methods: (1) use concatenation-based methods on overfitted models; (2) use difference-based methods on well-generalized models; (3) sort the posteriors before the concatenation and difference operations.

4.2.3.3 Impact of Overfitting

Overfitting measures the difference in accuracy between training and testing data. Previous studies [145, 182, 98] have shown that overfitted models are more susceptible to classical membership inference attacks, while well-generalized models are almost immune to them. In this subsection, we want to revisit the impact of overfitting on our attack.

Table 4.1 depicts the attack AUC for different overfitting levels. We use the random forest as the attack model and use SortedDiff and SortedConcat as feature construction methods for well-generalized and overfitted target models, respectively. In general, our attack consistently outperforms the classical membership inference on both well-generalized and overfitted models. On the overfitted models, i.e., CIFAR10 datasets with ResNet50 and DenseNet as target models, we can observe that the classical membership inference also works. However, our attack can achieve much better performance. On the other hand, the experimental results show that **our attack can still correctly infer the membership status of the target sample in well-generalized models**. For example, when the target model is a decision tree, the overfitting level in the Adult (Income) dataset is 0.019; thus, the decision tree can be regarded as a well-generalized model. While the performance of classical membership inference on this model is equivalent to random guessing (AUC = 0.497), our attack achieves a good performance, with an AUC of 0.882. In summary, our attack performance is relatively independent of the overfitting level.

Table 4.1: Attack AUC in different overfitting levels.

Dataset	$\mathcal{M}_o$	Train / Test Acc.	Overfitting	AUC / Base-AUC
Adult	LR	0.795 / 0.782	0.013	0.600 / 0.505
	DT	0.853 / 0.834	0.019	0.882 / 0.497
	RF	0.852 / 0.843	0.009	0.659 / 0.459
	MLP	0.767 / 0.763	0.004	0.506 / 0.503
Accident	LR	0.702 / 0.698	0.002	0.538 / 0.494
	DT	0.722 / 0.701	0.021	0.929 / 0.501
	RF	0.730 / 0.709	0.021	0.78 / 0.499
	MLP	0.670 / 0.644	0.026	0.513 / 0.493
Insta-NY	LR	0.508 / 0.439	0.069	0.983 / 0.490
	DT	0.404 / 0.373	0.031	0.941 / 0.503
	RF	0.523 / 0.442	0.081	0.685 / 0.551
	MLP	0.738 / 0.483	0.255	0.619 / 0.553
MNIST	SimCNN	0.954 / 0.951	0.003	0.511 / 0.496
CIFAR10	DenseNet	0.942 / 0.477	0.465	0.881 / 0.630
	ResNet50	0.975 / 0.592	0.383	0.719 / 0.548

4.2.4 Hyperparameters

We now evaluate the impact of the hyperparameters on the performance of our attack. Specifically, we focus on the number of shadow original models S_o, the number of samples S_r per shadow original model, and the number of unlearned models S_u per shadow original model. The corresponding hyperparameters of the target models are fixed (as defined at the end of Section 4.2.1) since only the hyperparameters of the shadow models can be tuned to launch the attack.

We conduct the experiments on the Adult (Income) dataset with a decision tree as the target model. Following our findings in Section 4.2.3.2, we evaluate the attack AUC of different combinations of attack models, i.e., decision tree, random forest, logistic regression, and difference-based feature construction methods, i.e., DirectDiff, SortedDiff, EucDist.

Number of Shadow Original Models S_o. Figure 4.6a depicts the impact of S_o, which varies from 1 to 100. The figure shows that the attack AUC sharply increases when S_o increases from 1 to 5, but remains quite stable for greater values of S_o. This indicates that setting $S_o = 5$ is enough for the diversity of the shadow original models.

Number of Samples S_r per Model. Figure 4.6b illustrates the impact of the number

of samples per model S_r, the values are in the range of $\in \{500, 1000, 2000, 5000, 10000\}$. When S_r increases from 500 to 1000, the attack AUC with SortedDiff increases from 0.67 to 0.83, while the attack AUC with EucDist increases from 0.73 to 0.86, except for logistic regression. However, adding more than 1000 samples does not help improve the attack performance further.

Number of Unlearned Models S_u per Shadow Original Model. Figure 4.6c illustrates the impact of S_u, which varies from 1 to 100 and the legends are 5 combinations of attack models and feature construction methods guided by Section 4.2.3.2. We observe that S_u has negligible impact on the attack AUC. This indicates that using a few unlearned models is sufficient to achieve a high attack performance.

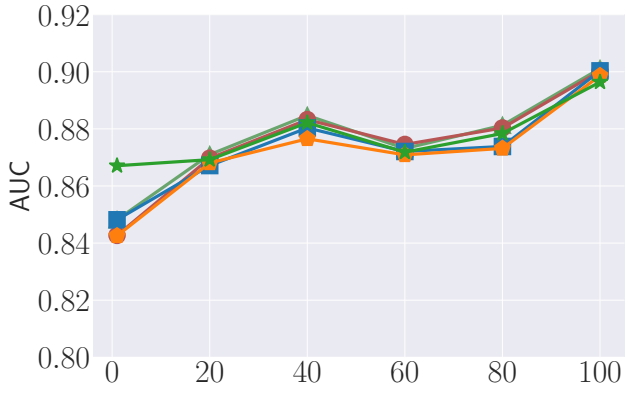
4.2.5 Attack Transferability

In practice, the adversary might not be able to get the same distribution dataset or the same model structure to train the shadow models. We next validate the dataset and model transferability between the shadow model and the target model. That is, we evaluate whether the adversary can use a different dataset and model architecture than the target model to train the shadow models. We evaluate categorical datasets with simple model structures and image datasets with complex model structures in Table 4.2. Where columns stand for dataset transfer, and rows stand for model transfer. Names on the left of the arrows stand for configurations of shadow models, values in the parentheses stand for the attack AUC of the classical membership inference.

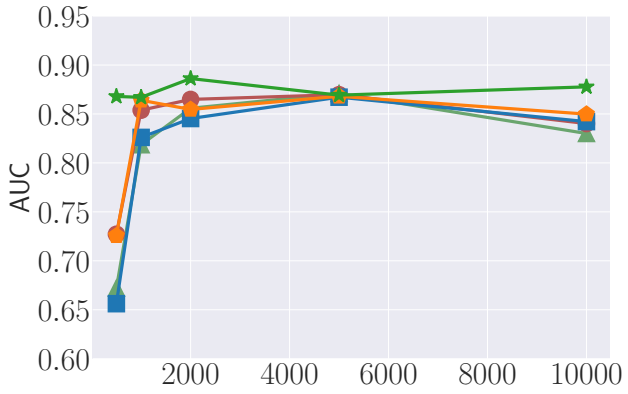
Table 4.2: Attack AUC for dataset and model transfer.

Shadow→Target	**Insta-NY→Insta-NY**	Insta-NY→Insta-LA
DT→DT	**0.944 (0.491)**	0.931 (0.503)
DT→LR	0.964 (0.494)	0.974 (0.513)
LR→LR	**0.986 (0.505)**	0.982 (0.511)
LR→DT	0.927 (0.502)	0.926 (0.508)
Shadow→Target	**CIFAR10→CIFAR10**	CIFAR10→STL10
DenseNet →DenseNet	**0.881 (0.630)**	0.813 (0.621)
DenseNet→ResNet50	0.847 (0.624)	0.805 (0.632)
ResNet50→ResNet50	**0.719 (0.548)**	0.687 (0.550)
ResNet50→DenseNet	0.721 (0.523)	0.675 (0.542)

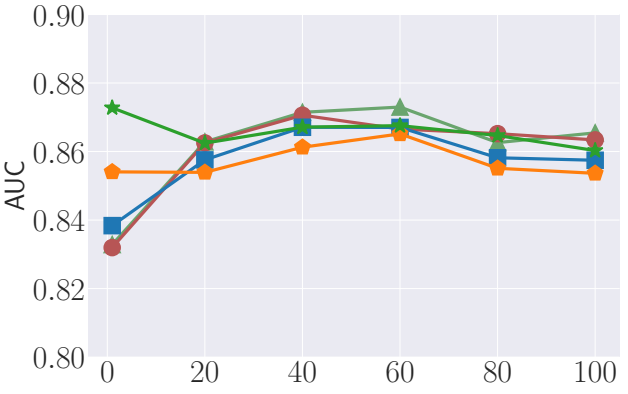
Dataset Transferability. Comparing the AUC values of the transfer setting with that of the non-transfer setting, i.e., bold rows in column Insta-NY→Insta-LA and CIFAR10→STL10, we only observe a small performance drop for all target models. For



(a) Number of $\mathcal{M}_o^s$.



(b) Number of Training Samples in $\mathcal{M}_o^s$.



(c) Number of $\mathcal{M}_u^s$ per $\mathcal{M}_o^s$.

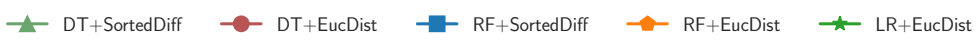


Figure 4.6: Attack AUC sensitivity to different hyperparameters on the Adult (income) dataset with decision tree as target model.

instance, when the target model is a decision tree, the attack AUC of the transfer and

non-transfer settings are 0.944 and 0.931, respectively. The attack AUC only drops by 1%.

Model Transferability. For model transferring attacks, we evaluate the pairwise transferability among the decision tree and logistic regression. In Table 4.2, unbold rows in column Insta-NY→Insta-NY and CIFAR10→CIFAR10 illustrate the performance of model transfer. The experimental results show that model transfer only slightly degrades the attack performance of our attack. For example, when the shadow and target models are LR, the attack AUC equals 0.986. When we change the target model to a decision tree, the attack AUC is still 0.927.

Dataset and Model Transferability. Unbold rows of unbold columns show the attack AUC when we transfer both the dataset and the model simultaneously. Even in this setting, our attack can achieve pretty good performance. This result shows the robust transferability of our attack when the adversary does not have access to the same distribution data and same model architectures.

4.2.6 Evaluation of the SISA Method

The unlearning algorithm we have focused on so far is retraining from scratch, which can become computationally prohibitive for large datasets and complex models. Several *approximate* unlearning algorithms have been proposed to accelerate the training process. In this subsection, we evaluate the performance of our attack against the most general approximate unlearning algorithm, SISA [25].

Setup. We remind the readers that the main idea of SISA is to split the original dataset into k disjoint shards and train k sub-models. In the inference phase, the model owner aggregates the prediction of each sub-model to produce the global prediction using some aggregation algorithm. In this experiment, we set $k = 5$ and use the posterior average as the aggregation algorithm. Figure 4.7 illustrates the attack AUC on the Insta-NY dataset. We report the experimental results of four different target models and four different attack models.

Results. The experimental results show that our attack performance drops compared to the Scratch algorithm. We posit this is because the aggregation algorithm of SISA reduces the influence of a specific sample on its global model. This observation further motivates deploying unlearning methods such as SISA in real-world applications.

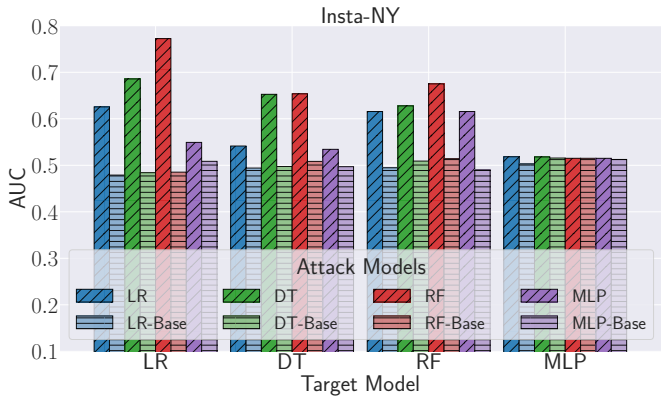


Figure 4.7: Attack AUC for the SISA method on the Insta-NY dataset.

4.2.7 Attack Under Different Scenarios

Next, we evaluate the effectiveness of our attack in different scenarios that might exist in practice. We first focus on the case when there exist multiple intermediate versions of unlearned models. Second, we consider when a group of samples is deleted. Third, we investigate the online learning setting when multiple samples are deleted and added simultaneously. Finally, we evaluate the impact of unlearning on the remaining samples' membership privacy.

4.2.7.1 Multiple Intermediate Unlearned Models

As discussed in the threat model (Section 4.1.2), the adversary gains access to the original and unlearned models by continuously querying the black-box target model. In practice, the adversary obtains access to two consecutive versions of models by two consecutive queries. However, the model owner would produce an unlearned model whenever it receives deletion requests; thus, there might be multiple unlearned models between these two consecutive queries that are unknown to the adversary. We call these models *intermediate models*. Here, we evaluate the effectiveness of our attack when multiple intermediate unlearned models exist.

Setup. Due to space limitations, we concentrate on the Insta-NY dataset with three different target models, while the conclusions are consistent for other datasets. We use LR as the attack model and select the best features following the principles described in Section 4.2.3.2. Figure 4.8 depicts the results. The x-axis represents the number of intermediate unlearned models we studied, i.e., $\{1, 10, 50, 100, 250\}$.

Results. The experimental results show that our attack consistently degrades the

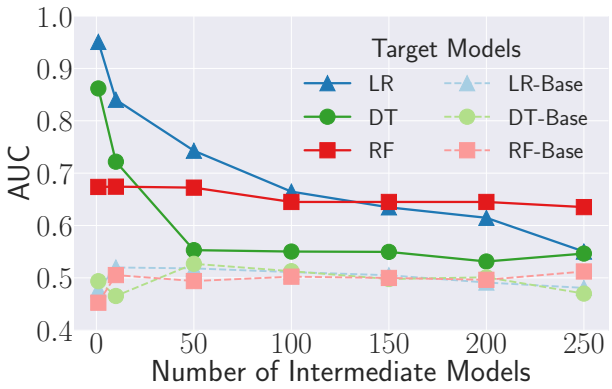


Figure 4.8: Multiple Intermediate Unlearned Models.

privacy of the target sample compared with the classical membership inference. In addition, the attack AUC drops when the number of intermediate models increases. This is expected since previously deleted samples mask the impact of the target sample. If multiple intermediate models exist, the discrepancy information between the original model and the unlearned model is contributed by both the target sample and other deleted samples corresponding to the intermediate models. In other words, the impact of the target sample and other deleted samples is entangled with each other, making the inference of the membership status of the target sample more difficult.

Note that the data samples are unlikely to be revoked frequently in practice, and the number of intermediate models is unlikely to be very large, so our attack is still effective in real-world settings. For instance, our attack AUC can achieve at least 0.84 when the number of intermediate models is less than 10 when the target model is LR.

4.2.7.2 Group Deletion

In practice, cases could exist where a group of samples is deleted at once before generating the unlearned model. This can happen when multiple data owners request the deletion simultaneously or when the model owner caches the deletion requests and updates the model only when he has received numerous requests to save computational resources.

Setup. We conduct experiments on our attack in the group deletion scenario. We randomly delete a group of data samples from each original model to generate the unlearned model. The ratio of samples in each group takes value from $\{0.02\%, 0.2\%, 1\%, 2\%, 5\%\}$. We delete at most 5% of the data samples since it is unlikely that more than 5% of users revoke their data in practice. We evaluate our attack on the Insta-NY dataset with three target models. Notice that the unlearned model of the group deletion is the same

as in Section 4.2.7.1 when the group size equals the number of intermediate unlearned models. The difference is that in group deletion, we consider all samples in the group as target samples.

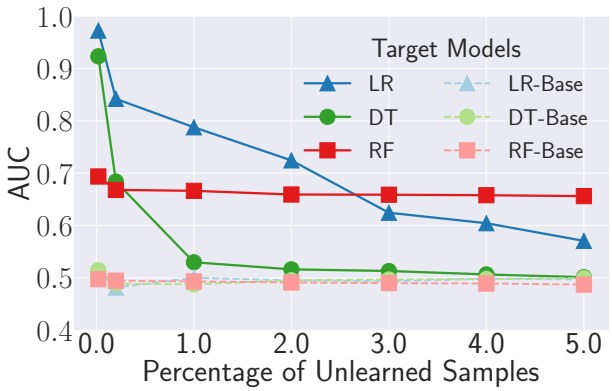


Figure 4.9: Group Deletion.

Results. Figure 4.9 shows that our consistently outperforms classical membership inference attack, demonstrating extra information leakage in group deletion. However, the attack performance of group deletion is slightly worse than single sample deletion, even though our attack is still effective when the group size is smaller than 0.2%. For example, when the target model is LR, the attack AUC of single deletion and group deletion (0.2% target samples) are 0.972 and 0.842, respectively. The reason is that a single sample could be hidden among the deleted sample group, thereby preserving its membership information. This result reveals that conducting group deletion could mitigate, to some extent, the impact of our attack.

In practice, we believe 0.2% might already be too large for unlearning. The results of [22] show that 3.2 million requests for removing URLs have been issued to Google for 5 years, which certainly constitutes less than 0.2% of the total URLs Google indexes.

4.2.7.3 Online Learning

In real-world deployments, ML models are often updated with new samples, which is known as online learning or incremental learning. Next, we evaluate the performance of our attack in online learning settings where multiple samples are deleted and added simultaneously.

Setup. To set up the experiment, we delete a group of target samples from the original dataset and add the same number of new samples; then, we retrain the model from

scratch to obtain the unlearned model. We conduct experiments on Insta-NY with different target models and use LR as the attack model.

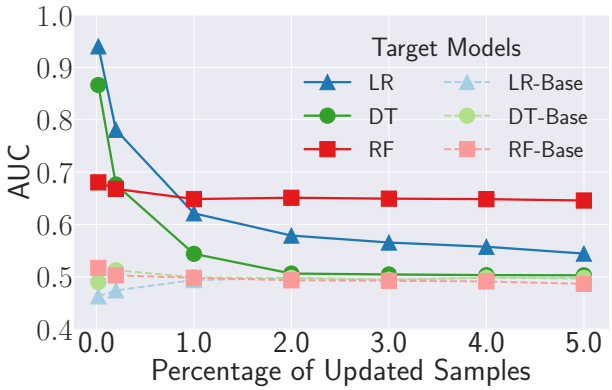


Figure 4.10: Online Learning.

Results. Figure 4.10 show that adding samples to the target model in the unlearning process has a slight impact on our attack. For example, compared with pure deletion, the attack AUC only slightly drops from 0.972 to 0.940 when the target model is LR, and the number of unlearned samples equals 10 (0.2%).

4.2.7.4 Impact on Remaining Samples

Finally, we evaluate whether deleting the target sample can influence the privacy of other remaining samples.

Setup. We use the same attack pipeline described in Section 4.1.3 to mount the attack. Concretely, we use data samples that reside in both the original and unlearned models as positive cases and use shadow/target negative datasets as negative cases. We concentrate on the Insta-NY dataset with four target models and four attack models where one data sample is deleted.

Results. Figure 4.11 shows that the attack AUC of our attack is higher than that of the classical membership inference, which only exploits information of the original model, indicating deleting the target sample also degrades privacy, to some extent, of the remaining samples. However, the attack AUC of all target models are less than 0.6, meaning the remaining samples are less sensitive to our attack. This is expected because the remaining samples are members of both the original model and the unlearned model. Deleting other data samples has some but limited impact on their posteriors in the unlearned model.

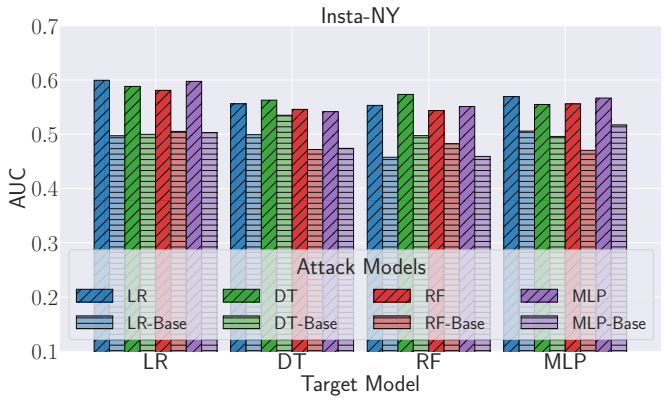


Figure 4.11: Attack AUC of the remaining samples on the Insta-NY dataset.

4.3 Mitigating the Unintended Privacy Risk

We explore four possible defense mechanisms and empirically evaluate their effectiveness. The former two mechanisms reduce the information accessible to the adversary [145], and the latter two eliminate the impact of a single sample on the output of the ML models. In Table 4.3, we list the attack performance of no defense mechanisms (ND), publishing the Top-k confidence values (Top-1, Top-2, Top-3), label-only defense (Label), temperature scaling (TS) based defense, and differential privacy (DP) based defense. $\mathcal{D}_o$, $\mathcal{M}_T$, and $\mathcal{M}_A$ stand for the original dataset, target model, and attack model, respectively.

4.3.1 Publishing the Topk Confidence Values

This defense reduces the attacker's knowledge by only publishing the top k confidence values of the posteriors returned by both original and unlearned models. Formally, we denote the posterior vector as $\mathbb{P} = [p_1, p_2, \cdots, p_\ell]$, where ℓ is the number of classes of the target model and p_i is the confidence value of class i. When the target model receives a query, the model owner calculates posteriors $\mathbb{P}$ and sorts them in descending order, resulting in $\mathbb{P}^s = [p_1^s, p_2^s, \cdots, p_\ell^s]$. The model owner then publishes the first k values in $\mathbb{P}^s$, i.e., $[p_1^s, p_2^s, \cdots, p_k^s]$.

In the machine unlearning setting, the top k confidence values of the original model and the unlearned model may not correspond to the same set of classes. To launch our attack, the adversary constructs a *pseudo-complete posterior vector* for both the original model and the unlearned model. The pseudo-complete posteriors take the published

confidence values for their corresponding classes and evenly distribute the remaining confidence value to other classes, i.e., for $j \in \{k+1, \ldots, \ell\}$, $p_j^s = \frac{1-(p_1^s+p_2^s+\cdots+p_k^s)}{\ell-k}$. The adversary can then launch our attack using the pseudo-complete posteriors.

Table 4.3 shows the experimental results of Top-1, Top-2 and Top-3 defenses on Insta-NY and Adult. For the Adult dataset, we report the results of the decision tree as the target model; for the Insta-NY dataset, we report the results of the decision tree and logistic regression as the target model. We report the performance of 4 different attack models, each selecting the best feature following the principle described in Section 4.2.3.2. The results show that publishing top k confidence value *cannot* effectively mitigate our attack.

Table 4.3: Attack AUC of the defense mechanisms.

$\mathcal{D}_o$ ($\mathcal{M}_T$)	$\mathcal{M}_A$	**ND**	**Top-1**	**Top-2**	**Top-3**	**Label**	**TS**	**DP**[ϵ_1]	**DP**[ϵ_2]
Adult (DT)	RF	0.916	0.899	0.906	0.911	0.501	-	-	-
	DT	0.918	0.903	0.906	0.910	0.506	-	-	-
	LR	0.918	0.904	0.907	0.911	0.506	-	-	-
	MLP	0.918	0.904	0.909	0.907	0.493	-	-	-
Insta-NY (DT)	RF	0.937	0.930	0.931	0.942	0.506	-	-	-
	DT	0.938	0.932	0.932	0.943	0.502	-	-	-
	LR	0.928	0.923	0.927	0.926	0.502	-	-	-
	MLP	0.928	0.923	0.927	0.929	0.505	-	-	-
Insta-NY (LR)	RF	0.976	0.947	0.965	0.965	0.546	0.635	0.519	0.477
	DT	0.972	0.946	0.961	0.961	0.546	0.654	0.524	0.500
	LR	0.969	0.948	0.960	0.962	0.546	0.610	0.519	0.500
	MLP	0.970	0.948	0.960	0.966	0.453	0.653	0.506	0.504

4.3.2 Publishing the Label Only

This defense further reduces the information accessible to the adversary by only publishing the predicted label instead of confidence values (posteriors). To launch our attack, the adversary also needs to construct the pseudo-complete posteriors for both the original model and the unlearned model. The main idea is to set the confidence value of the predicted class as 1 and set the confidence value of other classes as 0. Table 4.3 illustrates the performance of the "label only" defense. The experimental setting is similar to the Top-k defense. The experimental results show that the "label only" defense can effectively mitigate our attack in all cases. The reason is that deleting one sample is unlikely to change the output label of a specific target sample.

It is worth noting that recent studies have shown that an adversary can recover, to

a large extent, the posteriors from the label with the so-called sampling attack [128, 92]. In this case, our membership inference attack is still effective. We leave the investigation of the improved attack in the presence of label-only publishing defense as future work.

4.3.3 Temperature Scaling

Temperature scaling divides the logits vector by a learned scaling parameter, which is a simple yet effective approach to eliminate the over-confident problem of the output posteriors of neural networks [60]. This defense reduces the impact of a single sample on the output posteriors.

Table 4.3 illustrates the performance of the "temperature scaling" defense. We report the performance of 4 different attack models, each selecting the best feature following the principle described in Section 4.2.3.2. The experimental results show that temperature scaling is an effective defense mechanism. However, this method only applies to neural networks whose last layer is SoftMax. Logistic regression in our experiment can be regarded as a neural network with one input layer and one softmax layer.

4.3.4 Differential Privacy (DP)

DP [89, 42, 122, 24, 111] guarantees that any single data sample in a dataset has a limited impact on the output. Previous studies have shown DP can effectively prevent classical membership inference attacks [76, 96]. To validate whether DP can prevent our membership inference attack in the machine unlearning setting, we train both the original model and unlearned model in a differentially private manner.

We experiment with Differentially-Private Stochastic Gradient Descent (DP-SGD) [12], the most representative DP mechanism for protecting machine learning models. The core idea of DP-SGD is to add Gaussian noise to the gradient g during the model training process, i.e., $\tilde{g} = g + \mathcal{N}\left(0, \Delta_f^2 \sigma^2 \mathbf{I}\right)$. We use the Opacus library[1] developed by Facebook to conduct our experiments. Note that since DP-SGD can only be applied to the ML models that encounter gradient updating in the training process, we only report the results for logistic regression. We set the privacy budget parameters as $\delta = 10^{-5}, \epsilon_1 = 4.64, \epsilon_2 = 0.7$. The last two columns of Table 4.3 illustrate the effectiveness of the DP defense. The experimental results show that DP can effectively prevent our membership inference attack. It is worth noting that DP can inevitably degrade

[1] https://github.com/pytorch/opacus

the target model's accuracy. We need carefully tune the privacy budget parameters to strike a trade-off between privacy and model utility in practice.

We leave the in-depth exploration of more effective defense mechanisms against our attack as future work.

4.4 Conclusion

In summary, we have made several important observations:

- Our attack consistently degrades the membership privacy of the Scratch unlearning method compared to classical membership inference. The attack performance drops for the SISA unlearning method, which motivates deploying unlearning methods such as SISA in real-world applications.
- We obtain the following rules for selecting the feature construction methods: (1) use concatenation-based methods on overfitted models; (2) use difference-based methods on well-generalized models; (3) sort the posteriors before the concatenation and difference operations.
- Our transferring attacks show that our attack is still effective when the shadow model is trained on different-distributed datasets and different architecture from the target model.
- When the number of unlearned/updating samples represents less than 0.2% of the training dataset, our attack is still effective in the scenarios of multiple intermediate unlearned models, group deletion, and online learning.
- Deleting the target sample also degrades the privacy of the remaining samples to some extent; however, the remaining samples are less sensitive to our attack.

In our evaluation, privacy degradation is especially significant for well-generalized ML models where classical membership inference does not perform well. To avoid the potential negative impacts of our attack, we investigate four mechanisms to mitigate the newly discovered privacy risks and show that releasing the predicted label only, temperature scaling, and differential privacy are effective. We believe our results can help improve privacy protection in practical implementations of machine unlearning.

5

Assessing the Privacy Risks in Graph Embedding Sharing Systems

Recently, a new family of deep learning models known as graph neural networks (GNNs) has been proposed to obtain graph embedding and has achieved state-of-the-art performance. The core idea of GNNs is to train a deep neural network that aggregates the feature information from neighborhood nodes to obtain *node embedding*. They can be further aggregated to obtain the *graph embedding* for graph classification. Such graph embedding is empirically considered sanitized since the whole graph is compressed to a single vector. In turn, it has been shared with third parties to conduct downstream graph analysis tasks. For example, the graph data owner can generate the graph embeddings locally and upload them to the Embedding Projector service[1] provided by Google to explore the properties of the graph embeddings visually. Despite that sharing graph embeddings for downstream graph analysis tasks is intriguing and practical, the associated security and privacy implications remain unanswered.

In this chapter, we initiate a systematic investigation of the privacy issue of graph embedding by exploring three inference attacks. The first attack is *property inference* attack, which aims to infer the basic properties of the target graph given the graph embedding, such as the number of nodes, the number of edges, the graph density, etc. We then investigate the *subgraph inference* attack. That is, given the graph embedding and a subgraph of interest, the adversary aims to determine whether the subgraph is contained in the target graph. For instance, an adversary can infer whether a specific chemical compound structure is contained in a molecular graph if gaining access to its graph embedding, posing a direct threat to the intellectual property of the data owner. The challenge of the subgraph inference attack is that the formats of the graph embedding (i.e., a vector) and the subgraph of interest (i.e., a graph) are different and not directly comparable. Finally, we aim to reconstruct a graph that shares similar structural properties (e.g., degree distribution, local clustering coefficient, etc.) with the target graph. We call this attack *graph reconstruction* attack. For instance, if the target graph is a social network, the reconstructed graph would allow an adversary to gain direct knowledge of sensitive social relationships.

We introduce the threat model and attack taxonomy in Section 5.1. We depict the design of three inference attacks in Section 5.2, Section 5.3 and Section 5.4, respectively. In Section 5.5, we conduct extensive experiments on multiple real-world graph datasets to illustrate the effectiveness of our proposed attacks. We propose and evaluate an effective defense mechanism in Section 5.6. We summarize some future work and conclude the chapter in Section 5.7.

[1] https://projector.tensorflow.org/

5.1 Threat Model and Attack Taxonomy

5.1.1 Attack Scenario

We focus on the whole graph embedding $H_{\mathcal{G}}$, which is oftentimes computed on a sensitive graph (e.g., biomedical molecular network and social network). Such graph embedding $H_{\mathcal{G}}$ is empirically considered sanitized since the whole graph is compressed to a single vector. In practice, it has been shared with third parties to conduct downstream graph analysis tasks. For example, the graph data owner can calculate the graph embeddings locally and upload them to the Embedding Projector service provided by Google to explore the properties of the graph embeddings visually. Another example is that some companies release their graph embedding systems, together with which they publish some pretrained graph embeddings to facilitate the downstream tasks. These systems including the PyTorch BigGraph[2] system developed by Facebook, DGL-KE[3] system developed by Amazon, and GROVER developed by Tencent[4]. Besides, the graph embeddings can also be shared in the well-known model partitioning paradigm [86, 78]. This paradigm can effectively improve the scalability of inference by allowing the graph data owner to calculate the graph embeddings locally and upload them to the cloud for further inference or analysis.

Despite sharing graph embeddings for downstream graph analysis tasks being intriguing and promising, the associated security and privacy implications remain unanswered. For instance, Song et al. [152, 149] demonstrated that the embeddings could leak sensitive information about image and text data in Euclidean space. Recall that the goal of graph embedding $H_{\mathcal{G}}$ is to preserve graph-level similarity. A natural question is: would the graph embedding $H_{\mathcal{G}}$ leak sensitive structural information of its corresponding graph $\mathcal{G}$?

5.1.2 Threat Model

We consider the scenario where the adversary obtains a whole graph embedding (which is referred to as *target graph embedding* $H_{\mathcal{G}_T}$) from the victim, either from Embedding Projector, pretrained graph embeddings, or model partitioning paradigm. The goal of the adversary is to infer the sensitive information of the graph that is used to generate this graph embedding. We call this graph *target graph* $\mathcal{G}_T$ and the GNN model that

[2] https://github.com/facebookresearch/PyTorch-BigGraph
[3] https://github.com/awslabs/dgl-ke
[4] https://github.com/tencent-ailab/grover

is used to generate the target graph embedding *target embedding model* $\mathcal{F}_T$. Note that inferring the sensitive information of the target graph with "graph embedding" is more challenging than that with "node embeddings" in previous study [40]. From the attacker's perspective, it represents the most difficult setting since the whole graph is compressed to a single vector by the aforementioned pooling methods in Section 2.1.2. To train the attack model $\mathcal{F}_A$, we assume the adversary has an auxiliary dataset $\mathcal{D}_{aux}$ that comes from the same distribution of the target graph. This is plausible in practice. For instance, if the target graph embedding is generated from a social network, the adversary can collect social network graphs by themselves through public data API.[5] For molecular networks, the adversary can use public datasets online.[6] We also show that our attacks are still effective when $\mathcal{D}_{aux}$ comes from a different distribution than the target graphs in Section 5.5. We further assume the adversary only has black-box access to the target embedding model [152, 149], which is the most difficult setting for the adversary [145, 74, 150, 104, 150]. This assumption is plausible when the target embedding model is accessible via public API or freely available online.[7]

5.1.3 Attack Taxonomy

We formalize three inference attacks that can reveal sensitive information about the target graph given the threat model. An overview of the attack taxonomy is shown in Figure 5.1. Concretely, the adversary obtains the whole graph embedding $H_{\mathcal{G}_T}$ of a sensitive target graph $\mathcal{G}_T$, which is primarily shared with third parties for downstream tasks, and aims to infer sensitive information about $\mathcal{G}_T$: (1) Infer the basic properties of $\mathcal{G}_T$, such as the number of nodes, the number of edges, and graph density ($\mathcal{F}_{AP}$); (2) given a subgraph of interest $\mathcal{G}_S$, infer whether $\mathcal{G}_S$ is contained in $\mathcal{G}_T$ ($\mathcal{F}_{AS}$); (3) reconstruct a graph $\mathcal{G}_R$ that is similar with $\mathcal{G}_T$ ($\mathcal{F}_{AR}$).

Property Inference Attack ($\mathcal{F}_{AP}$). Given the target graph embedding $H_{\mathcal{G}_T}$, the attack goal is to infer the basic properties of $\mathcal{G}_T$, such as the number of nodes, the number of edges, the density, etc. Note that the primary goal of GNN is learning information from graphs for downstream tasks, e.g., protein toxicity prediction. Many graph properties, such as node numbers, are not related to the downstream tasks, and successful property inference attacks imply such properties are overlearned [152, 149] by GNNs. These properties can be proprietary when the graph contains valuable

[5] https://developer.twitter.com/en/docs/twitter-api
[6] https://chrsmrrs.github.io/datasets
[7] http://snap.stanford.edu/gnn-pretrain/

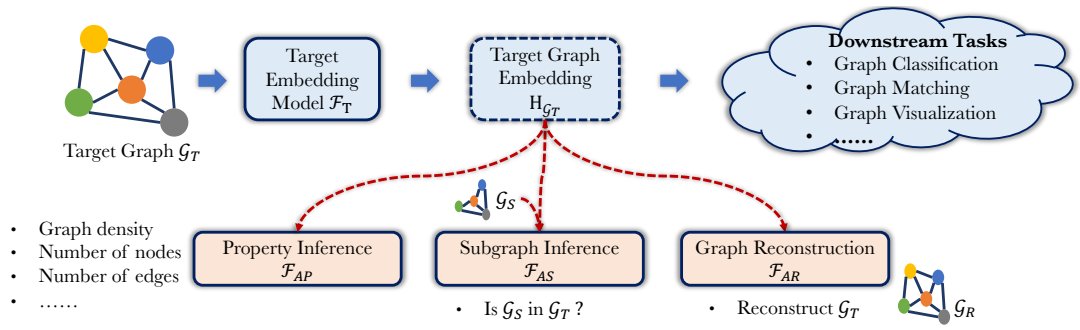


Figure 5.1: Attack taxonomy of the graph embedding.

information, such as molecules. Inferring such properties can directly violate the intellectual property (IP) of the data owner.

Subgraph Inference Attack ($\mathcal{F}_{AS}$). Given the target graph embedding $H_{\mathcal{G}_T}$ and a subgraph of interest $\mathcal{G}_S$, the attack goal is to infer whether $\mathcal{G}_S$ is contained in $\mathcal{G}_T$. For instance, an attacker can infer whether a specific chemical compound structure ($\mathcal{G}_S$) is contained in a molecular graph ($\mathcal{G}_T$) if gaining access to its graph embedding ($H_{\mathcal{G}_T}$).

Note that we consider the scenario where the subgraph constitutes a major part of the target graph. Small graphs, such as triangles or stars, are universal for almost all graphs, hence not taking part in our subgraph inference attack.

Graph Reconstruction Attack ($\mathcal{F}_{AR}$). Given the graph embedding $H_{\mathcal{G}_T}$, the attack goal is to reconstruct a graph $\mathcal{G}_R$ that shares similar graph structural statistics, such as degree distribution and local clustering coefficient, with $\mathcal{G}_T$. Concretely, we aim to reconstruct an adjacency matrix A of $\mathcal{G}_T$. Knowing the high-level structural quantities of the molecular graphs may lead to IP loss for the companies creating them. For instance, the adversary can develop generic drugs at a much lower cost than the famous pharmaceutical companies by exploiting the high-level structural quantities of the reconstructed molecular graphs to narrow down the search space.

5.2 Property Inference Attack

5.2.1 Attack Overview

Given the target graph embedding $H_{\mathcal{G}_T}$, the goal of the property inference attack is to infer the basic properties of the target graph $\mathcal{G}_T$, such as the number of nodes, the number of edges, and density. Figure 5.2 illustrates the general attack pipeline of the

property inference attack. Our attack model $\mathcal{F}_{AP}$ takes as input the target graph embedding $H_{\mathcal{G}_T}$ and outputs all the interested graph properties of $\mathcal{G}_T$ simultaneously. The attack model $\mathcal{F}_{AP}$ is a multi-task classifier, which consists of multiple output layers, each predicting one graph property.

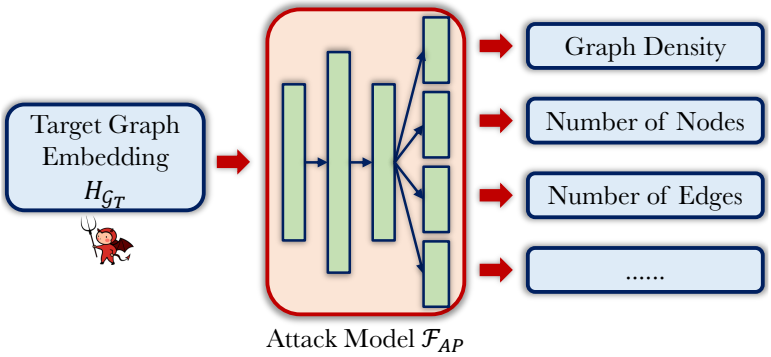


Figure 5.2: Attack pipeline of the property inference attack.

5.2.2 Attack Model $\mathcal{F}_{AP}$

Model Definition. Formally, the property inference attack $\mathcal{F}_{AP}$ is defined as

$$\mathcal{F}_{AP} : H_{\mathcal{G}_T} \rightarrow \{\text{graph properties}\}$$

Concretely, the attack model consists of a feature extractor $\mathcal{E}$ (multiple sequential linear layers), and multiple parallel prediction layers $\mathcal{M}$, each responsible for predicting one property. We outline the technical details of building $\mathcal{F}_{AP}$ below.

Training Data. To train the attack model, we need a set of graph embeddings $H_{\mathcal{G}}$ and a set of properties of interest $\mathbb{P}$. As discussed in Section 5.1, the adversaries have access to an auxiliary dataset $\mathcal{D}_{aux}$ that comes from the same distribution of $\mathcal{G}_T$. The adversaries can obtain the auxiliary graph embedding $H_{\mathcal{G}_{aux}}$ of the auxiliary graph $\mathcal{G}_{aux} \in \mathcal{D}_{aux}$ by querying the target embedding model. Finally, we use the graph properties of $\mathcal{G}_{aux}$ to label $H_{\mathcal{G}_{aux}}$. We further bucketize the domain of the property values into k bins. For instance, if the density of a graph is in the range of $[0, 1]$ and $k = 5$, we bucketize the graph density into 5 bins, which results in 5 classes in the classification. Note that modeling the inference of continuous value into a multi-class classification is commonly used, such as demographic properties prediction in social networks [101] and dropout rate prediction [115].

Training Attack Model. Recall that the attack model $\mathcal{F}_{AP}$ combines a feature extractor $\mathcal{E}$ and multiple prediction layers $\mathcal{M}$. We can train the attack model by optimizing the following optimization problem:

$$\min \underset{\mathcal{G}_{aux} \in \mathcal{D}_{aux}}{\mathbb{E}} \left[\sum_{p \in \mathbb{P}} \mathcal{L}\left[\mathcal{M}^p(\mathcal{E}(H_{\mathcal{G}_{aux}})), p\right] \right]$$

, where $\mathbb{P}$ is the set of properties that the attackers interested, p is a property in $\mathbb{P}$, $\mathcal{L}$ is the cross-entropy loss. Notice that all properties share the same parameters for $\mathcal{E}$, and use different parameters for $\mathcal{M}^p$.

5.3 Subgraph Inference Attack

5.3.1 Attack Overview

Given the target graph embedding $H_{\mathcal{G}_T}$ and a subgraph of interest $\mathcal{G}_S$, the attack goal is to infer whether $\mathcal{G}_S$ is contained in $\mathcal{G}_T$. Here, we assume that $\mathcal{G}_S$ constitutes a major part of the target graph $\mathcal{G}_T$.[8] That is, we do not focus on small subgraphs, such as triangles or stars, as they appear in almost all the graphs, thus not worth the adversary's efforts. The general attack pipeline of the subgraph inference attack is illustrated in Figure 5.3. The attack model $\mathcal{F}_{AS}$ has two inputs with different formats, namely target graph embedding and subgraph. The subgraph is transformed into a subgraph embedding by an embedding extractor integrated into the attack model, aggregated with the target embedding, and sent to a binary classifier for prediction.

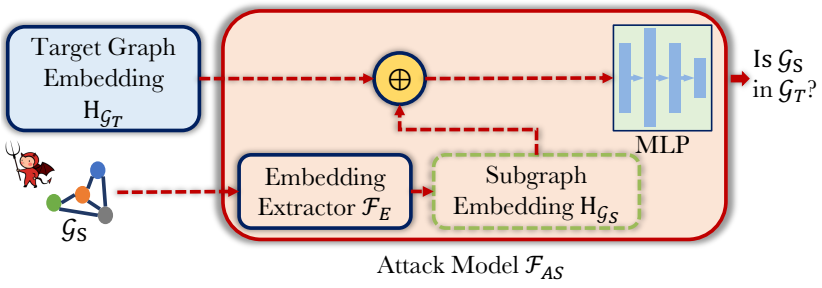


Figure 5.3: Attack pipeline of the subgraph inference attack.

Note that the subgraph inference attack is more challenging than the property

[8] We experiment with subgraphs containing from 20% to 80% of the target graph's nodes (see Section 5.5.3).

inference attack $\mathcal{F}_{AP}$. First, subgraph isomorphism is known to be NP-complete [50]. Second, the attack model $\mathcal{F}_{AS}$ has two inputs with different formats, namely the embedding ($H_{\mathcal{G}_T}$) and the graph ($\mathcal{G}_S$), and cannot be directly compared. To make the two inputs comparable, we integrate a *graph embedding extractor* $\mathcal{M}_E$ in the attack model to transform the subgraph $\mathcal{G}_S$ to a subgraph embedding $H_{\mathcal{G}_S}$. The architecture of $\mathcal{M}_E$ can be either the same with (when the target embedding model is known) or different from (when the target embedding model is unknown) the target embedding model $\mathcal{F}_T$. Finally, the target graph embedding $H_{\mathcal{G}_T}$ and the subgraph embedding $H_{\mathcal{G}_S}$ are aggregated, using the approaches introduced in Section 5.3.2, and sent to a binary classifier for prediction.

5.3.2 Attack Model $\mathcal{F}_{AS}$

Attack Definition. Formally, the subgraph inference attack is defined as

$$\mathcal{F}_{AS} : \langle H_{\mathcal{G}_T}, \mathcal{G}_S \rangle \rightarrow \{\mathcal{G}_S \in \mathcal{G}_T, \mathcal{G}_S \notin \mathcal{G}_T\}$$

Concretely, the attack model $\mathcal{F}_{AS}$ is a binary classifier to determine if a given subgraph $\mathcal{G}_S$ is contained in the target graph $\mathcal{G}_T$. We outline the technical details of building $\mathcal{F}_{AS}$ below.

Generating Positive and Negative Samples. Similar to the property inference attack, we use the auxiliary dataset $\mathcal{D}_{aux}$ to obtain the training data for the attack model $\mathcal{F}_{AS}$. To generate ground truth for $\mathcal{F}_{AS}$, given an auxiliary graph $\mathcal{G}_{aux} \in \mathcal{D}_{aux}$, we generate a *positive subgraph* $\mathcal{G}_S \in \mathcal{G}_{aux}$ and a *negative subgraph* $\bar{\mathcal{G}_S} \notin \mathcal{G}_{aux}$. The positive subgraph $\mathcal{G}_S$ is generated by sampling a subgraph from the auxiliary graph $\mathcal{G}_{aux}$ using the graph sampling method, such as *random walk*. To generate the negative subgraph $\bar{\mathcal{G}_S}$, we use the same sampling method to sample a subgraph from another auxiliary graph $\mathcal{G}'_{aux} \in \mathcal{D}_{aux}$ and $\mathcal{G}'_{aux} \neq \mathcal{G}_{aux}$. As aforementioned, the subgraph of interest constitutes a major part of the target graph, and the sampled negative subgraph $\bar{\mathcal{G}_S}$ is unlikely to be contained in $\mathcal{G}_{aux}$.

For each auxiliary graph $\mathcal{G}_{aux}$, we have one positive subgraph $\mathcal{G}_S$ and one negative subgraph $\bar{\mathcal{G}_S}$. The adversary first obtains the auxiliary graph embedding $H_{\mathcal{G}_{aux}}$ by querying the target embedding model. They then have a *positive sample* $\langle H_{\mathcal{G}_{aux}}, \mathcal{G}_S \rangle$, which is labeled as 1, and a negative sample $\langle H_{\mathcal{G}_{aux}}, \bar{\mathcal{G}_S} \rangle$, which is labeled as 0, for the attack model.

Constructing Features. The attack model first uses a graph embedding extractor to

transform the subgraph $\mathcal{G}_S$ into a subgraph embedding $H_{\mathcal{G}_S}$ to make the two inputs comparable. The attack model then aggregates the target graph embedding $H_{\mathcal{G}_T}$ and the subgraph embedding $H_{\mathcal{G}_S}$ to generate an *attack feature vector* χ. We propose the following three aggregation strategies:

- **Concatenation.** A commonly used approach is to concatenate the two graph embeddings, i.e., $\chi = H_{\mathcal{G}_T}||H_{\mathcal{G}_S}$, where $||$ is the concatenation operation.
- **Element-wise Difference.** An alternative is to calculate the element-wise difference between two graph embeddings, i.e., $\chi = H_{\mathcal{G}_T} - H_{\mathcal{G}_S}$.
- **Euclidean Distance.** Another approach is to calculate the Euclidean distance between two graph embeddings, i.e., $\chi = ||H_{\mathcal{G}_T} - H_{\mathcal{G}_S}||_2$.

We empirically evaluate the effectiveness of these three strategies in Section 5.5.3.

Training Attack Model. The final step of the attack is to send the attack feature vector χ to a binary classifier, which is modeled as an MLP (multi-layer perceptron), to determine whether $\mathcal{G}_S$ is contained in $\mathcal{G}_T$. We use the cross entropy loss and gradient descent algorithm to train the attack model. Note that the binary classifier and the graph embedding extractor in the attack model $\mathcal{F}_{AS}$ are trained simultaneously.

5.4 Graph Reconstruction Attack

5.4.1 Attack Overview

Given the target graph embedding $H_{\mathcal{G}_T}$, the attack goal is to reconstruct a graph $\mathcal{G}_R$ that has similar graph statistics, such as degree distribution and local clustering coefficient, with the target graph $\mathcal{G}_T$. Figure 5.4 shows the overall attack pipeline of the graph reconstruction attack. The attack model $\mathcal{F}_{AR}$ is a decoder that can transform the embedding into a graph. The decoder can be obtained from the graph auto-encoder paradigm. The graph reconstruction attack is the most challenging task because we are rebuilding the whole graph from a single vector H_G. To this end, the attack model $\mathcal{F}_{AR}$ leverages a tailored graph auto-encoder [146] and puts its decoder into service to transform the graph embedding into a graph. Once trained, the adversary feeds $H_{\mathcal{G}_T}$ to the decoder, and the decoder outputs reconstructed graph $\mathcal{G}_R$ that has similar graph statistics with the target graph $\mathcal{G}_T$.

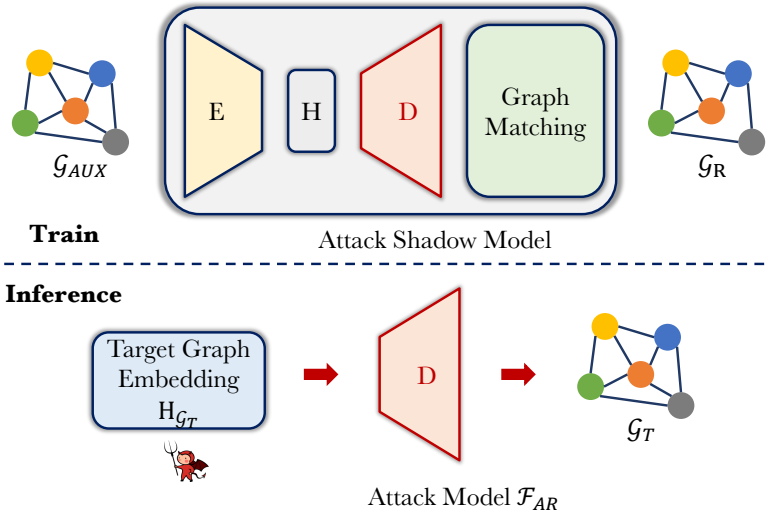


Figure 5.4: Attack pipeline of the graph reconstruction attack.

5.4.1.1 Attack Model $\mathcal{F}_{AR}$

Attack Definition. Formally, the graph reconstruction attack is defined as

$$\mathcal{F}_{AR} : H_{\mathcal{G}_T} \rightarrow \mathcal{G}_R$$

Essentially, the graph reconstruction attack $\mathcal{F}_{AR}$ is the decoder of a customized graph auto-encoder. We outline the technical details of building $\mathcal{F}_{AR}$ below.

Graph Auto-encoder Design. We use the graph auto-encoder paradigm to train the attack model. The architecture is shown in the training phase of Figure 5.4. We use an auxiliary dataset $\mathcal{D}_{aux}$ to train the graph auto-encoder. Different from the auto-encoder in the image domain, the graph auto-encoder has an additional component named *graph matching* except for the encoder and decoder. The reason for introducing the graph matching component is that neither the auxiliary graph $\mathcal{G}_{aux} \in \mathcal{D}_{aux}$ nor the reconstructed graph $\mathcal{G}_R$ imposes node orderings (i.e., graph isomorphism), making the calculation of loss between $\mathcal{G}_{aux}$ and $\mathcal{G}_R$ inaccurate. For instance, an auxiliary graph $\mathcal{G}_{aux}$ and a reconstructed graph $\mathcal{G}_R$ with the same structure and completely different node orderings can have different adjacency matrices, such that the loss between $\mathcal{G}_{aux}$ and $\mathcal{G}_R$ is large while it is expected to be zero. Besides, the encoder in the graph auto-encoder can transform a graph to graph embedding, which can be modeled as a GNN model. The decoder can transform the graph embedding back to the graph in the form of an adjacency matrix, which can be modeled as a multi-layer perceptron.

Graph Matching. Following the same strategy as in [146], we adopt the maximum pooling matching method in our implementation. The main idea is to find a transformation matrix $Y \in \{0,1\}^{n \times n}$ between $\mathcal{G}_T$ and $\mathcal{G}_R$, where $Y_{a,i} = 1$ if node $v_a \in \mathcal{G}_T$ is assigned to $v_i \in \mathcal{G}_R$, and $Y_{a,i} = 0$ otherwise. Due to space limitation, we refer the readers to [146] for the detailed calculation of Y.

Training Attack Model. To train the graph auto-encoder, we use the cross entropy to calculate the loss between $\mathcal{G}_{aux}$ and $\mathcal{G}_R$, which calculates the cross entropy between each pair of elements in $\mathcal{G}_{aux}$ and $\mathcal{G}_R$. Formally, denote the adjacency matrix of $\mathcal{G}_{aux}$ and $\mathcal{G}_R$ as $A_{\mathcal{G}_{aux}}$ and $A_{\mathcal{G}_R}$ respectively. For each training sample, we first conduct the graph matching to obtain Y, then use the cross entropy between $A_{\mathcal{G}_{aux}}$ and $Y A_{\mathcal{G}_R} Y^T$ to update the graph auto-encoder.

Fine-tuning Decoder. Note that the structure or the parameters of the encoder can be different from the target embedding model; thus, the decoder may not perfectly capture the correlation between the auxiliary graph $\mathcal{G}_{aux}$ and its graph embedding $H_{\mathcal{G}_{aux}}$ generated by the target embedding model. To address this issue, we use the auxiliary graph $\mathcal{G}_{aux}$ to query the target embedding model and obtain the corresponding graph embedding $H_{\mathcal{G}_{aux}}$. Then, the graph-embedding pairs $\langle \mathcal{G}_{aux}, H_{\mathcal{G}_{aux}} \rangle$ obtained from the target embedding model are used to fine-tune the decoder using the same procedure of graph matching and loss function as aforementioned [131].

Discussion. Both the space and time complexity of the graph matching algorithm is $O(n^4)$; thus, our attack can only be applied to graphs with tens of nodes. This is enough in many real-world datasets, such as bioinformatics and molecular graphs. In the future, we plan to investigate more advanced methods to extend our attacks to larger graphs. Besides, our current attack can only restore the graph structure of the target graph. We plan to reconstruct the node features and the graph structure simultaneously in the future.

5.5 Evaluation

5.5.1 Experimental Setup

Datasets. We conduct our experiments on five public graph datasets from TUDataset [107], including DD, ENZYMES, AIDS, NCI1, and OVCAR-8H. These datasets are widely used as benchmark datasets for evaluating the performance of GNN models [177, 34, 44, 41]. DD and ENZYMES are bioinformatics graphs, where the nodes

Table 5.1: Dataset statistics.

Dataset	Type	#. Graphs	Avg. Nodes	Avg. Edges	#. Feats	#. Classes
DD	Bioinformatics	1,178	284.32	715.66	89	2
ENZYMES	Bioinformatics	600	32.63	62.14	21	6
AIDS	Molecules	2,000	15.69	16.20	42	2
NCI1	Molecules	4110	29.87	32.30	37	2
OVCAR-8H	Molecules	4052	46.67	48.70	65	2
PC3*	Molecules	2751	26.36	28.49	37	2
MOLT-4H*	Molecules	3977	46.70	48.74	65	2

represent the secondary structure elements, and an edge connects two nodes if they are neighbors along the amino acid sequence or one of the three nearest neighbors in space. The node features consist of the amino acid type, i.e., helix, sheet, or turn, as well as several physical and chemical information. AIDS, NCI1, and OVCAR-8H are molecule graphs, where nodes and edges represent atoms and chemical bonds, respectively. The node features typically consist of one-hot encoding of the atom type, e.g., hydrogen, oxygen, carbon, etc. Each dataset has multiple independent graphs with a different number of nodes and edges, and each graph is associated with a label. For instance, the label of the molecule datasets indicates the toxicity or biological activity determined in drug discovery projects. Table 5.1 summarizes the statistics of all the datasets.

Graph Embedding Models. As discussed in Section 2.1.2, the graph embedding models typically consist of node embedding modules and graph pooling modules (see Section 2.1.2). In our experiments, we use a 3-layer SAGE [64] module to implement node embedding. For graph pooling, we consider the following three methods.

- **MeanPool [63].** Given all the node embeddings $H_u, \forall u \in \mathcal{G}$, MeanPool directly average all the node embeddings to obtain the graph embedding, i.e., $H_{\mathcal{G}} = \frac{1}{|\mathcal{G}|} \sum_{u \in \mathcal{G}} H_u$, where $|\mathcal{G}|$ is the number of nodes in $\mathcal{G}$.
- **DiffPool [185].** This is a hierarchical pooling method, which relies on multiple layers of graph pooling operations to obtain the graph embedding $H_{\mathcal{G}}$. Concretely, we use three layers of graph pooling operations in our implementation. The first and second graph pooling layers narrow the number of nodes to $0.25 \cdot |\mathcal{G}|$ and $0.25^2 \cdot |\mathcal{G}|$, respectively, using DiffPool operation. In the last layer of graph pooling, we use the mean pooling operation to generate the final graph embedding $H_{\mathcal{G}}$.
- **MinCutPool [23].** This is also a hierarchical graph pooling method. Similar to DiffPool, we use three layers of graph pooling operations. The first two graph pooling

layers narrow the number of nodes to $0.5 \cdot |\mathcal{G}|$ and $0.5^2 \cdot |\mathcal{G}|$, respectively, using MinCutPool operation, and the last layer uses the mean pooling operation.

For presentation purposes, we use the name of graph pooling methods, namely MeanPool, DiffPool, and MinCutPool, to represent the graph embedding models in this section.

Experimental Settings. For each dataset $\mathcal{D}$, we split it into three disjoint parts, target dataset $\mathcal{D}_T$, attack training dataset $\mathcal{D}_A^{train}$, and attack testing dataset $\mathcal{D}_A^{test}$. The target dataset $\mathcal{D}_T$ (40%) is used to train the target embedding model $\mathcal{F}_T$, which is shared by all three inference attacks. The attack training dataset $\mathcal{D}_A^{train}$ (30%) corresponds to the auxiliary dataset $\mathcal{D}_{aux}$, which is used to generate the training data for the attack model. The attack testing dataset $\mathcal{D}_A^{test}$ (30%) corresponds to the target graph $\mathcal{G}_T$ in the attack phase. By default, we set the graph embedding dimension d_H as 192, which is the default setting of PyTorch Geometric.

5.5.2 Property Inference Attack

Evaluation Metrics. As the attack goal of the property inference attack is to infer the basic graph properties of the target graph $\mathcal{G}_T$, a commonly used metric to measure the attack performance is the *attack accuracy*, which calculates the proportion of graphs being correctly inferred.

Attack Setup. We conduct extensive experiments on five real-world graph datasets and three state-of-the-art GNN-based graph embedding models. In our experiments, we consider five different graph properties: Number of nodes, number of edges, graph density, graph diameter, and graph radius. For each graph property, we bucketize its domain into k bins, which transforms the attack into a multi-class classification problem. Concretely, for the number of nodes (edges) and the graph diameter (radius), the property domain is from 1 to the maximum number of nodes (edges) and the maximum graph diameter (radius) in the auxiliary dataset $\mathcal{D}_{aux}$. For the graph density, the property domain is $[0.0, 1.0]$. In our experiments, we consider four different *bucketization schemes*, i.e., $k \in \{2, 4, 6, 8\}$.

Competitors. We have two competitors for evaluating the effectiveness of our proposed attack.

- **Random Guessing (Random).** The most straightforward baseline is random guessing, which varies for different bucketization schemes. For instance, the attack

accuracy of random guessing for $k = 2$ and $k = 8$ are 0.5 and 0.125.

- **Directly Summarizing Auxiliary Dataset (Baseline).** Another baseline attack directly summarizes the properties from the auxiliary dataset $\mathcal{D}_{aux}$ instead of training a classifier. Concretely, we calculate the average property values from $\mathcal{D}_{aux}$ and use them for predicting the properties of the target graphs.

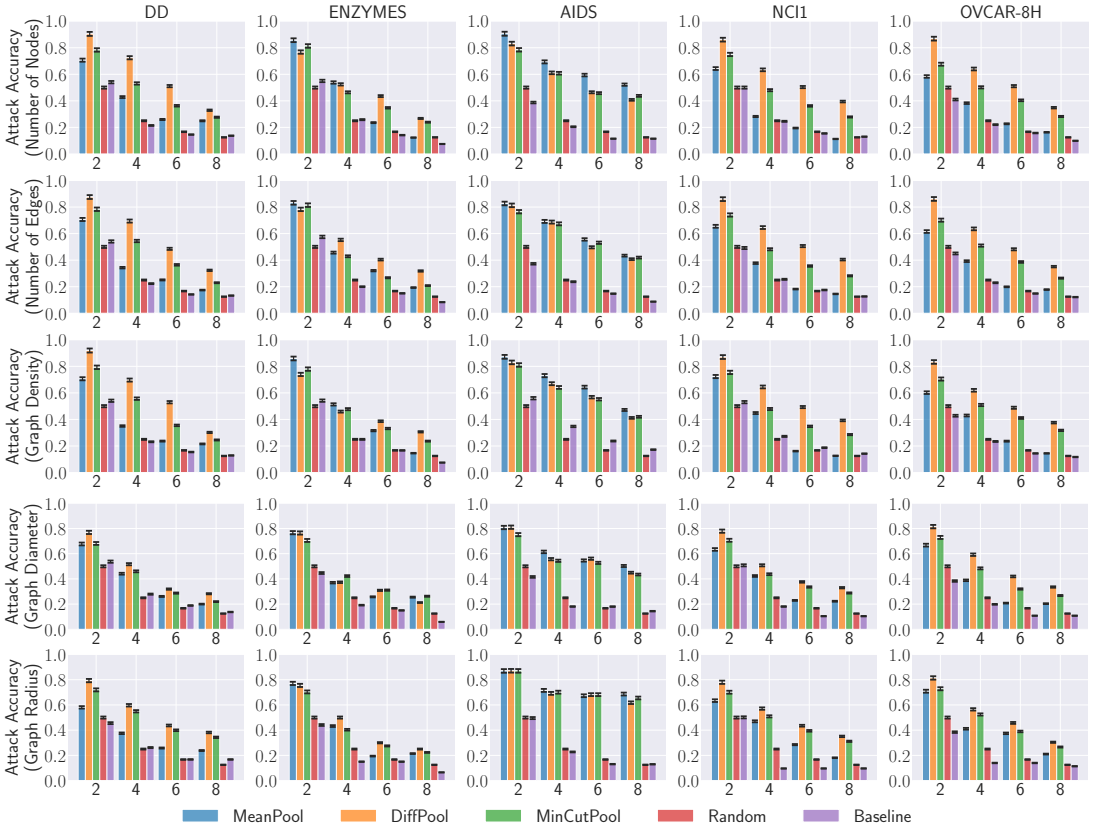


Figure 5.5: Attack accuracy for property inference.

Experimental Results. Figure 5.5 illustrates the attack performance, where different rows represent different graph properties and different columns represent different datasets. In each figure, different legends stand for different graph embedding models, and different groups stand for different bucketization schemes. The Random and Baseline methods represent the random guessing and summarizing auxiliary dataset baseline, respectively.

In general, the experimental results show that our attack outperforms two baseline attacks in most of the settings. For instance, when the bucketization scheme $k = 2$,

on the number of nodes property, we can achieve an attack accuracy of 0.904 on the DD dataset for the DiffPool model, while the attack accuracy of random guessing and summarizing auxiliary dataset baseline is 0.500 and 0.541, respectively. We further observe that a larger bucketization scheme k leads to worse attack accuracy. This is expected because larger k requires higher granularity of graph structural information and is more difficult for the classifier to distinguish. In addition, we note that, in most of the cases, the attack accuracy on the MeanPool model is worse than that of the other two graph embedding models, and sometimes even close to that of the random guessing baseline. This can be explained by the fact that the MeanPool model directly averages all the node embeddings, which might lose some graph structural information.

Datasets Transferability. In previous experiments, we assume the auxiliary dataset $\mathcal{D}_{aux}$ comes from the same distribution as the target graphs. To relax this assumption, we conduct additional experiments when $\mathcal{D}_{aux}$ comes from a different distribution than the target graphs. We evaluate the transferability between OVCAR-8H (OVC) and MOLT-4H (MOL), as well as between NCI1 and PC3 on MinCutPool with $k = 2$. The experimental results in Figure 5.6 show that our property inference attack is still effective when $\mathcal{D}_{aux}$ and the target graphs come from different distributions.

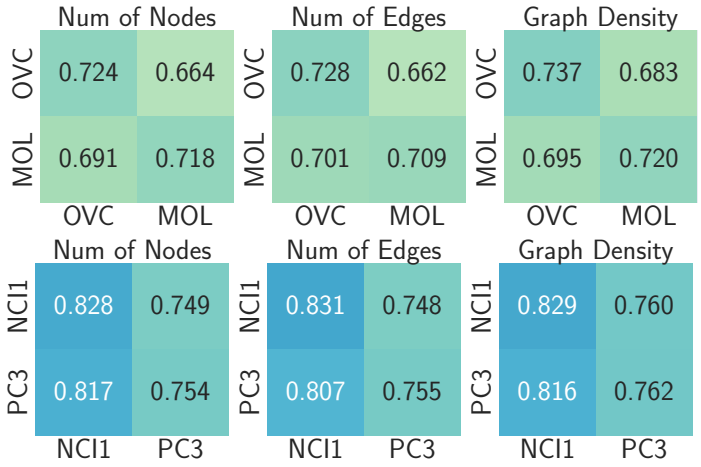


Figure 5.6: Datasets transferability for property inference attack.

5.5.3 Subgraph Inference Attack

Evaluation Metrics. Recall that the subgraph inference attack is a binary classification task; thus, we use the *AUC* metric to measure the attack performance, which is widely used to measure the performance of binary classification in a range of thresholds [48, 18,

124, 77, 191, P2]. The higher AUC value implies better attack performance. An AUC value of 1 implies maximum performance (true-positive rate of 1 with a false-positive rate of 0), while an AUC value of 0.5 means performance equivalent to random guessing.

Attack Setup. We conduct extensive experiments on five graph datasets and three graph embedding models to evaluate the effectiveness of our proposed attack. To obtain the subgraph, we rely on three graph sampling methods: Random walk sampling, snowball sampling, and forest fire sampling.

- **Random Walk Sampling.** The main idea of RandomWalk is to randomly pick a starting node and then simulate a random walk on the graph until we obtain the desired number of nodes.

- **Snowball Sampling.** The main idea of Snowball is to randomly select a set of seed nodes, and then iteratively select a set of neighboring nodes of the selected nodes until we obtain the desired number of nodes.

- **Forest Fire Sampling.** The main idea of FireForest is to randomly select a seed node, and begin "burning" outgoing edges and the corresponding nodes. Here, a node "burns" its outgoing edges and the corresponding nodes means these edges and nodes are sampled. If an edge gets burned, the node at the other endpoint gets a chance to burn its own edges, and so on recursively until we obtain the desired number of nodes.

For each sampling method, we consider four *sampling ratios*, i.e., $\{0.2, 0.4, 0.6, 0.8\}$, which determines how many nodes are contained in the subgraph. In practice, the sampling ratio is determined by the size of the subgraph of interest.

We use the element-wise difference to generate the feature vector χ. We generate the same number of positive samples and negative samples in both training and testing datasets to learn a balanced model.

Competitor. Recall that we integrate a graph embedding extractor in the attack model to transform the subgraph into subgraph embedding in Section 5.3. The embedding extractor is jointly trained with the binary classifier in the attack model. An alternative for subgraph inference is to generate the subgraph embedding from the target model together with the target graph embedding, and then train an isolated binary classifier as the attack model. To validate the necessity of integrating an embedding extractor

in the attack model, we compare it with the baseline attack that obtains subgraph embeddings from the target model.

Experimental Results. Figure 5.7 illustrates the attack performance, where different rows represent different datasets, and different columns represent different sampling methods. In each figure, different legends and groups stand for different graph embedding models and different sampling ratios. We use the element-wise difference method to generate the feature vector χ.

The experimental results show that our attack is effective in most of the settings, especially when the sampling ratio is 0.8. For instance, we can achieve 0.982 attack AUC on the DD dataset and MeanPool model with FireForest sampling method. Besides, we observe that when the sampling ratio decreases, the attack AUC decreases for most settings. This is expected as the positive samples, and the negative samples tend to be more similar to each other on smaller subgraphs, making the attack model more difficult to distinguish between them. Despite this, our attack can still achieve 0.859 attack AUC on ENSYMES and MeanPool with Snowball when the sampling ratio is 0.2.

Comparing different graph embedding models, we further observe that the subgraph inference attack performs the best on the MeanPool model in most settings, which is opposite to the property inference attack. We suspect this is because DiffPool and MinCutPool decompose the graph structure during their pooling process; thus, the subgraph as a whole might never be seen by the target model. This makes it harder for graph embedding matching to be effective.

Necessity of Embedding Extractor. Comparing with the baseline, we observe that our subgraph inference attack consistently outperforms the baseline attack in most cases, especially when the sampling ratio is small. For instance, on the DD dataset, when the sampling ratio is 0.2, our attack achieves 0.821 AUC on MeanPool model and FireForest sampling method, while the baseline attack achieves an AUC of 0.515. We further observe that when the sampling ratio increases, the baseline attack can gradually achieve a comparable attack AUC as our attack. This is expected as distinguishing between the positive subgraph, and negative subgraph is much easier when the sampling ratio is large.

Comparison of Feature Construction Methods. We propose three strategies to aggregate the graph embeddings of the target graph and the subgraph of interest in the attack model $\mathcal{F}_{AS}$, namely concatenation, element-wise difference, and Euclidean distance, in Section 5.3. We now compare the performance of different strategies.

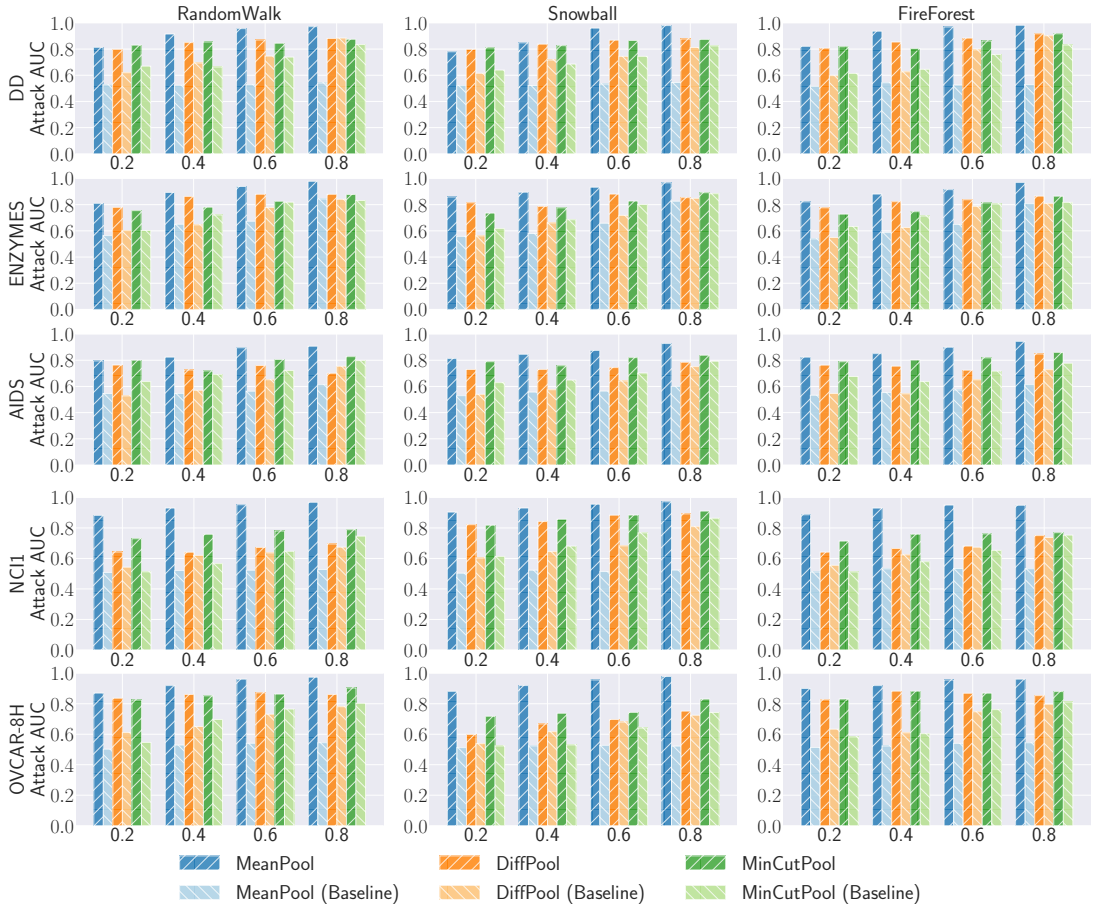


Figure 5.7: Attack AUC for subgraph inference attack.

Table 5.2: Attack AUC for different feature construction methods in the subgraph inference attack.

	0.8			0.6			0.4			0.2		
Dataset	Concat	EDist	EDiff	Concat	EDist	EDiff	Concat	EDist	EDiff	Concat	EDist	EDiff
DD	0.53 ± 0.01	0.81 ± 0.06	**0.88 ± 0.01**	0.51 ± 0.01	0.79 ± 0.04	**0.87 ± 0.01**	0.52 ± 0.01	0.79 ± 0.02	**0.85 ± 0.01**	0.50 ± 0.02	0.71 ± 0.08	**0.80 ± 0.00**
ENZYMES	0.49 ± 0.02	0.63 ± 0.10	**0.88 ± 0.03**	0.52 ± 0.03	0.71 ± 0.10	**0.88 ± 0.03**	0.54 ± 0.02	0.56 ± 0.07	**0.86 ± 0.01**	0.48 ± 0.02	0.53 ± 0.03	**0.78 ± 0.01**
AIDS	0.51 ± 0.01	0.53 ± 0.04	**0.78 ± 0.04**	0.55 ± 0.01	0.51 ± 0.02	**0.76 ± 0.05**	0.54 ± 0.01	0.51 ± 0.03	**0.73 ± 0.06**	0.56 ± 0.02	0.50 ± 0.00	**0.76 ± 0.05**
NCI1	0.51 ± 0.00	0.51 ± 0.02	**0.70 ± 0.06**	0.49 ± 0.02	0.52 ± 0.01	**0.67 ± 0.06**	0.50 ± 0.01	0.51 ± 0.01	**0.64 ± 0.03**	0.49 ± 0.01	0.51 ± 0.01	**0.64 ± 0.00**
OVCAR-8H	0.54 ± 0.01	0.63 ± 0.12	**0.89 ± 0.02**	0.50 ± 0.04	0.69 ± 0.09	**0.88 ± 0.02**	0.51 ± 0.03	0.74 ± 0.02	**0.84 ± 0.01**	0.54 ± 0.01	0.60 ± 0.13	**0.82 ± 0.02**

Table 5.2 shows the experimental results on five datasets when the graph embedding model is DiffPool, and the graph sampling method is RandomWalk. Due to space limitation, we use Concat, EDist, and EDiff to represent Concatenation, Euclidean Distance, and Element-wise Difference, respectively.

We observe that the element-wise difference method achieves the best performance, while the concatenation method has an attack AUC close to random guessing. This

indicates that the discrepancy information between two graph embeddings (element-wise difference method) is more informative than the plain graph embeddings (concatenation method) in terms of subgraph inference attack. Note that the Euclidean distance also implicitly captures the discrepancy information of two graph embeddings, while it relies on one scalar value and loses other rich discrepancy information.

Sampling Methods Transferability. So far, our experiments use the same sampling method for the auxiliary graph to train the attack model and the target graph to test the attack model. We conduct additional experiments to show whether our attack still works when the sampling methods are different. Figure 5.8 illustrates the experimental results on DD and ENZYMES datasets. RW, SB, and FF are abbreviations for RandomWalk, Snowball, and FireForest, respectively. We use DiffPool as the graph embedding model and adopt a sampling ratio of 0.8. As we can see, in most cases, the sampling methods do not have a significant impact on the attack performance.

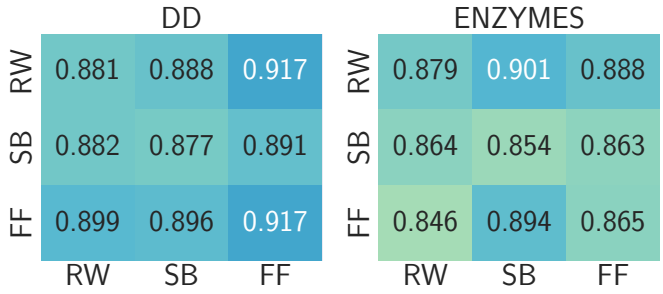


Figure 5.8: Sampling methods transferability for subgraph inference attack.

Embedding Models Transferability. In previous experiments, the architecture of the graph embedding extractor in the attack model is the same as the target embedding model. In practice, the model architecture of the target embedding model might be unknown to the adversaries. To understand whether our attack still works when the architectures are different, we conduct experiments on the DD and ENZYMES datasets. Figure 5.9 illustrates the experimental results of RandomWalk sampling method with a sampling ratio of 0.8. MP, DP, and MCP are abbreviations for MeanPool, DiffPool, and MinCutPool, respectively. We observe that the attack performance drops slightly when the model architectures are different. Despite this, we can still achieve a 0.773 attack AUC in the worse case.

Datasets Transferability. Similar to the property inference attack, to relax the assumption that $\mathcal{D}_{aux}$ comes from the same distribution of the target graphs, we conduct additional experiments when $\mathcal{D}_{aux}$ and the target graphs come from different

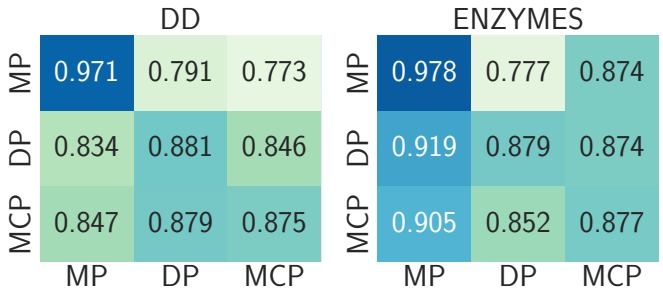


Figure 5.9: Embedding models transferability for subgraph inference attack.

distributions. We experiment on the RandomWalk method with a sampling ratio of 0.8. The experimental results in Figure 5.10 show that our subgraph inference attack is still effective for dataset transfer.

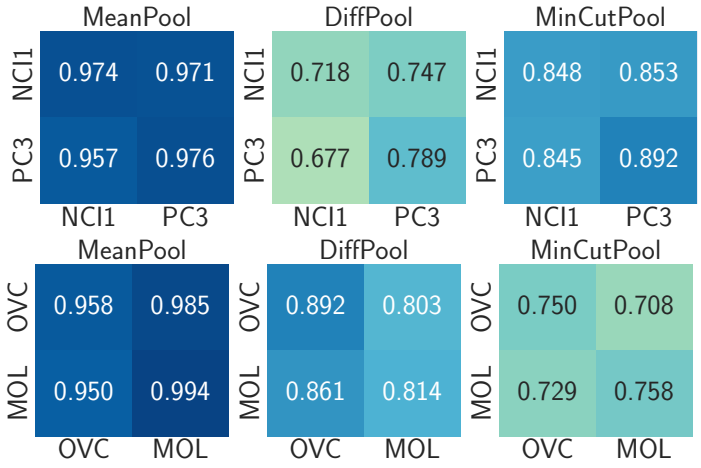


Figure 5.10: Dataset transferability for subgraph inference.

5.5.4 Graph Reconstruction Attack

Evaluation Metrics. We evaluate the performance of graph reconstruction from two perspectives:

1. **Graph Isomorphism**. The graph isomorphism compares the structure of the reconstructed graph $\mathcal{G}_R$ with the target graph $\mathcal{G}_T$ and determines their similarity. The graph isomorphism problem is well-known to be intractable in polynomial time; thus, approximate algorithms such as *Weisfeiler-Lehman (WL) algorithm* are widely used for addressing it [141, 177, 108]. The general idea of the WL algorithm is to calculate the WL graph kernel of two graphs iteratively. We

normalize the WL graph kernel in the range of [0.0, 1.0], and a WL graph kernel of 1.0 means two graphs perfectly match. We adopt the DGL implementation of the WL algorithm in our experiments.[9]

2. **Macro-level Graph Statistics**. Recall that the objective of the graph reconstruction attack is to generate a graph $\mathcal{G}_R$ that has similar graph statistics with the target graph $\mathcal{G}_T$. In practice, there are a plethora of graph structural statistics to analyze a graph, we adopt four of them: Degree distribution, local clustering coefficient (LCC), betweenness centrality (BC), and closeness centrality (CC).

   - **Degree Distribution.** The degree distribution $P(k)$ of a graph is defined to be the fraction of nodes in the graph with degree k. It is the most widely used graph statistic to quantify a graph.
   - **Local Clustering Coefficient (LCC).** The LCC of a node quantifies how close its neighbors are being into a cluster. It is primarily introduced to determine whether a graph is a small-world network.
   - **Betweenness Centrality (BC).** The betweenness centrality is a measure of centrality in a graph based on the shortest paths. For every pair of nodes in a graph, there exists at least one shortest path between the nodes such that either the number of edges that the path passes through is minimized. The betweenness centrality for each node is the number of these shortest paths that pass through the node.
   - **Closeness Centrality (CC).** The CC of a node is a measure of centrality in a graph, which is calculated as the reciprocal of the sum of the length of the shortest paths between the node and all other nodes in the graph. Intuitively, the more central a node is, the closer it is to all other nodes.

Note that the number of nodes in $\mathcal{G}_R$ might be different from the target graph $\mathcal{G}_T$ due to the graph auto-encoder architecture, and there are no node orderings imposed for $\mathcal{G}_R$ and $\mathcal{G}_T$; thus, we cannot directly compare the node-level graph statistics including LCC, CC, and BC. To address this issue, we bucketize the statistic domain into 10 bins and measure their distributions. For each graph statistic, we use three metrics to measure the distribution similarity between the target graph $\mathcal{G}_T$ and the reconstructed graph $\mathcal{G}_R$: *Cosine similarity*, *Wasserstein distance*, and *Jensen-Shannon (JS) divergence*.

[9] https://github.com/InkToYou/WL-Kernel-DGL

Intuitively, higher cosine similarity and lower Wasserstein distance/JS divergence mean better attack performance. The ranges of cosine similarity, Wasserstein distance, and JS divergence are $[-1.0, 1.0]$, $[0.0, 1.0]$, and $[0.0, 1.0]$, respectively.

Attack Setup. Recall that both the space and time complexity of the graph matching algorithm is $O(n^4)$, we conduct our experiments on three small datasets in Table 5.1, i.e., AIDS, ENZYMES, and NCI1, and three graph embedding models. We run all the experiments five times with the mean and standard deviation reported.

Table 5.3: Attack performance of graph reconstruction measured by graph isomorphism.

Dataset	DiffPool	MeanPool	MinCutPool
AIDS	0.875 ± 0.003	0.794 ± 0.003	0.869 ± 0.002
ENZYMES	0.670 ± 0.019	0.653 ± 0.022	0.704 ± 0.012
NCI1	0.752 ± 0.005	0.771 ± 0.010	0.693 ± 0.007

Experimental Results. Table 5.3 and Table 5.4 illustrate the attack performance in terms of graph isomorphism and macro-level graph statistics (measured by cosine similarity), respectively. In general, our attack achieves strong performance. For instance, the WL graph kernel on AIDS and DiffPool achieves 0.875. Besides, the cosine similarity of the betweenness centrality distribution is larger than 0.85 for all the settings. We can also achieve 0.99 cosine similarity for local clustering coefficient distribution for the AIDS and NCI1 datasets. For degree distribution and closeness centrality distribution, the attack performance is slightly worse; however, we can still achieve cosine similarity larger than or close to 0.5.

Table 5.4: Macro-level graph statistics on the attack performance of graph reconstruction, the similarity of which is measured by cosine similarity.

Dataset	Target Model	Degree Dist.	LCC Dist.	BC Dist.	CC Dist.
AIDS	MeanPool	0.651 ± 0.001	0.999 ± 0.001	0.987 ± 0.001	0.876 ± 0.002
	DiffPool	0.894 ± 0.001	0.999 ± 0.001	0.983 ± 0.001	0.787 ± 0.002
	MinCutPool	0.888 ± 0.003	0.999 ± 0.001	0.983 ± 0.001	0.785 ± 0.006
ENZYMES	MeanPool	0.450 ± 0.070	0.646 ± 0.005	0.959 ± 0.001	0.516 ± 0.037
	DiffPool	0.519 ± 0.007	0.661 ± 0.008	0.958 ± 0.001	0.504 ± 0.005
	MinCutPool	0.467 ± 0.019	0.490 ± 0.009	0.916 ± 0.001	0.414 ± 0.009
NCI1	MeanPool	0.736 ± 0.003	0.999 ± 0.001	0.877 ± 0.001	0.402 ± 0.001
	DiffPool	0.633 ± 0.002	0.999 ± 0.001	0.877 ± 0.001	0.495 ± 0.002
	MinCutPool	0.570 ± 0.002	0.999 ± 0.001	0.877 ± 0.001	0.496 ± 0.001

Impact of Graph Auto-encoder. To investigate the impact of the quality of the graph auto-encoder on the attack performance, we conduct additional experiments on the graph auto-encoders trained with different epochs. Figure 5.11 shows the experimental results. We observe that as the number of epochs increases, our attack performance increases, indicating the quality of the graph auto-encoder has a positive impact on our attack. When the number of epochs exceeds 10, the attack performance remains unchanged for most of the settings. Thus, we train the graph auto-encoder for 10 epochs in our experiments.

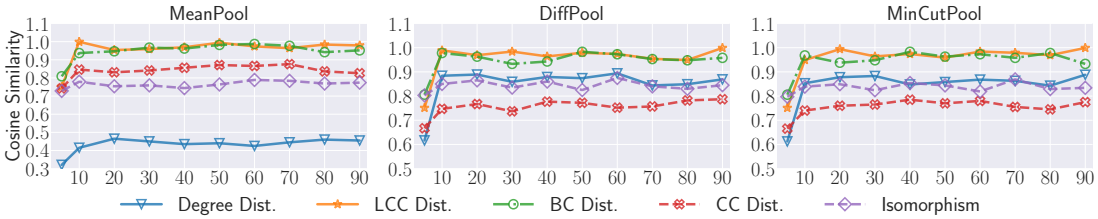


Figure 5.11: Impact of the quality of graph auto-encoder on the AIDS dataset.

Visualization. To better illustrate the effectiveness of our graph reconstruction attack on preserving the macro-level graph statistics, we provide a distribution visualization of the AIDS dataset in Figure 5.12. We experiment on the MinCutPool model. The visualization results show that our graph reconstruction attack can effectively preserve the macro-level graph statistics.

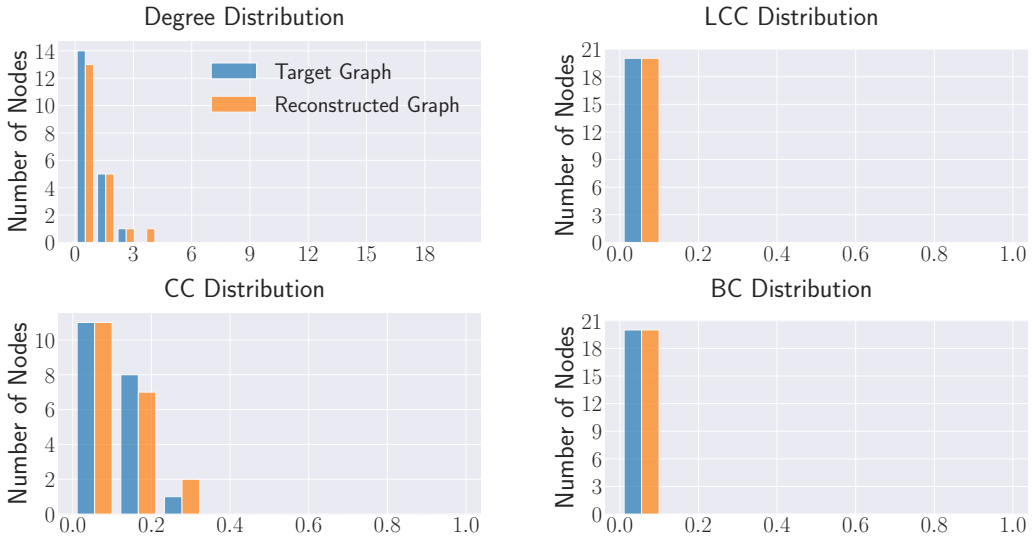


Figure 5.12: Visualization of macro-level graph statistic distribution for graph reconstruction attack on the AIDS dataset.

5.6 Defenses

Graph Embedding Perturbation. A commonly used defense mechanism for inference attacks is adding perturbation to the output of the model [192]. We propose to add perturbations to the target graph embedding $H_{\mathcal{G}_T}$ to defend our proposed inference attacks. Formally, given the target graph embedding $H_{\mathcal{G}_T}$, the data owner only shares a noisy version of graph embedding $\tilde{H_{\mathcal{G}_T}} = H_{\mathcal{G}_T} + \mathsf{Lap}(\beta)$ to the third party, where $\mathsf{Lap}(\beta)$ denotes a random variable sampled from the Laplace distribution with scale parameter β; that is, $\mathsf{Pr}[\mathsf{Lap}(\beta) = x] = \frac{1}{2\beta} e^{-|x|/\beta}$. Notice that adding noise to the graph embedding vector may destroy the graph structural information, thus affecting normal tasks such as graph classification. Therefore, we need to choose a moderate level of noise to trade off the defense effectiveness and the performance of the normal tasks.

Defense Evaluation Setup. We conduct experiments to validate the effectiveness of our proposed defense against all the inference attacks, as well as the impact on the normal graph classification task. For the property inference attack, we evaluate the performance of graph density with bucketization scheme $k = 2$. For the subgraph inference attack, we consider the RandomWalk sampling method with a sampling ratio of 0.8. We conduct experiments on three graph embedding models.

Defense against Inference Attacks. Figure 5.13 illustrates the experimental results, where the first and second column represents the attack performance of the property inference attack and subgraph inference attack, respectively, and the last column represents the accuracy of the normal graph classification task. For two attacks, higher means better attack performance. For the original task, higher means better model utility. In each figure, the x-axis stands for the scaling parameter β of Laplace noise, where larger β means larger noise. The y-axis stands for the attack performance/normal graph classification accuracy. We observe that when the noise level increases, the attack performance for both property inference and subgraph inference attacks decreases. This is expected since more noise will hide more structural information contained in the graph embedding. On the other hand, the accuracy of the graph classification tasks will also decrease when the noise level increase. To defend against inference attacks while preserving the utility for normal tasks, one needs to choose the noise level carefully. For instance, when we set the standard deviation of Laplace noise to 2, the performance of the subgraph inference attack significantly drops while the graph classification accuracy only slightly decreases.

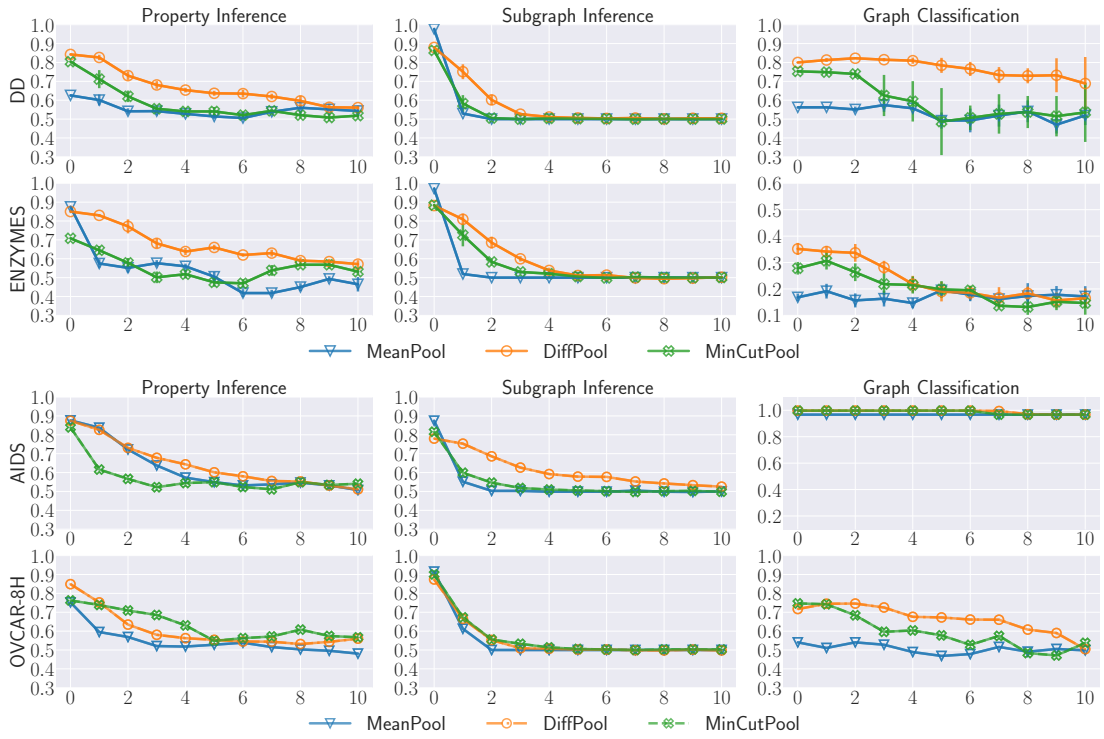


Figure 5.13: Perturbation defense against two inference attacks.

Defense against Graph Reconstruction Attack. Figure 5.14 illustrates the graph embedding perturbation defense performance for graph reconstruction attack. In each figure, the x-axis stands for the scaling parameter β for Laplace noise, where larger β means a higher noise level. The y-axis stands for the cosine similarity of degree distribution and graph isomorphism, respectively. The experimental results show that our defense mechanism is still effective for graph reconstruction attacks.

Connection to Differential Privacy. Note that adding Laplace noise in our graph embedding perturbation defense is closely related to the Laplace mechanism of differential privacy (DP) [42, 89], which provides a rigorous mathematical guarantee for data privacy. Therefore, it will be interesting to connect the graph embedding perturbation defense to DP. Formally, the graph embedding perturbation satisfies ϵ-DP if $\beta = \frac{\Delta_{\mathcal{F}_T}}{\epsilon}$, where $\Delta_{\mathcal{F}_T}$ is the ℓ_1 sensitivity of $\mathcal{F}_T$, which measures the maximum ℓ_1 distance of any two neighboring graphs (two graphs that differ in one edge). However, $\Delta_{\mathcal{F}_T}$ is difficult to compute since $\mathcal{F}_T$ is a complex non-linear function, and analyzing the impact of one edge of the target graph on the graph embedding is difficult. We leave the investigation of $\Delta_{\mathcal{F}_T}$ as our future work.

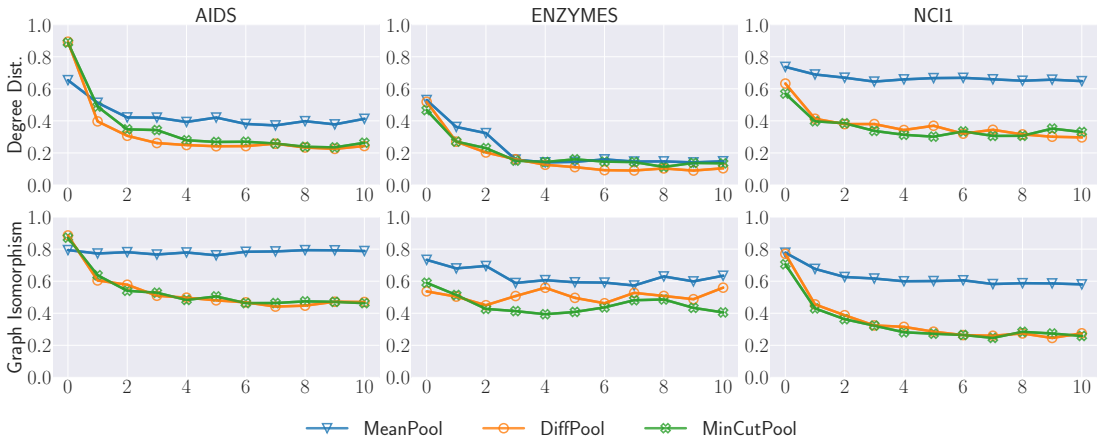


Figure 5.14: Perturbation defense against the graph reconstruction attack.

5.7 Conclusion

When dealing with the graph data, sharing graph embedding is empirically considered sanitized since the whole graph is compressed to a single vector. In turn, it has been shared with third parties to conduct downstream graph analysis tasks. For example, the graph data owner can generate the graph embeddings locally and upload them to the Embedding Projector service provided by Google to explore the properties of the graph embeddings visually.

In this chapter, we systematically investigate the information leakage of the graph embedding by mounting three inference attacks. We consider the scenario where the adversary obtains a whole graph embedding from the victim, either from Embedding Projector, pretrained graph embeddings, or model partitioning paradigm. The goal of the adversary is to infer the sensitive information of the graph that is used to generate this graph embedding. First, we can successfully infer basic graph properties, such as the number of nodes, the number of edges, and graph density, of the target graph with up to 0.89 accuracy. Second, given a subgraph of interest and the graph embedding, we can determine with high confidence whether the subgraph is contained in the target graph. For instance, we achieve 0.98 AUC on the DD dataset. Third, we propose a novel graph reconstruction attack that can reconstruct a graph with similar graph structural statistics to the target graph. We further propose an effective defense mechanism based on graph embedding perturbation to mitigate the inference attacks without noticeable performance degradation for graph classification tasks.

All three successful inference attacks prove that the graph information could be

leaked even if only sharing a compressed graph embedding vector. We need to be more cautious about sharing our data, even if it is in a representation form.

6

Assessing the Privacy Risks in Few-shot-based Facial Recognition Systems

With the power of facial recognition systems, entities with moderate resources can canvas the Internet for face images and build well-performed facial recognition models without people's awareness and consent. For example, `clearview.ai` reveals that a private company has collected 3 billion online face images and trained a powerful model capable of recognizing millions of citizens. Such kinds of misuse of facial recognition systems are potentially disastrous [106] and infringe the privacy laws such as European Union's General Data Protection Regulation (GDPR). GDPR states that *the personal data must only be processed if the individual has given explicit consent* (Article 6(1)(a)), and *the processing of personal data must be lawful, fair, and transparent* (Article 5(1)(a)) [1]. This means that if the third parties want to use the data owner's face images, they need to obtain consent from the data owner and inform the data owner how their face images are processed. Sharing personal data online typically implies that the data owners are willing to share their data with the public for social or promotion purposes. However, this does not grant others the right to misuse the data for unconsent purposes, particularly in commercial activities.

To prevent face images from being misused, one straightforward method is to modify the raw face images before uploading them to the Internet, such as distorting the face images [90], producing adversarial patches [158], or adding imperceptible pixel-level cloaks [137]. However, these approaches inevitably destroy the semantic information of the face images and also increase the difficulty of retroactivity. Also, researchers have argued that such defenses can be bypassed by newer technologies [126], which leads to an endless arms race between the attacker and defender.

In this chapter, we take a different angle by advocating a responsible *auditing* approach that enables normal users to detect whether their private face images are being used to train a facial recognition system. This approach provides users with evidence in claiming proprietary of their face images. Furthermore, it complies with data privacy protection regulations such as GDPR, which gives users the right to know how their data is processed. If data owners do not want any entity to use their face images to train the facial recognition system, they can use Face-Auditor for auditing if their face images are being used. If they find their face images were used without their consent, the data owners can take legal action against the model developer in accordance with GDPR regulations.

In Section 6.1, we first formulate the auditing process as user-level membership inference and depicts the design details of Face-Auditor. We conduct extensive experiments in Section 6.2 to illustrate the effectiveness of Face-Auditor. In Section 6.3, we investigate

the robustness of Face-Auditor when different defense mechanisms are introduced to protect the training images or the target models. We discuss the practical impacts and the future work and conclude the chapter in Section 6.4.

6.1 Auditing Methodology

Auditing Goal. We aim to determine whether any of the *target user* u's face images were used to train a target facial recognition model $\mathcal{M}_T$ (*target model* for short). We formulate this auditing process as a *user-level* membership inference problem. Formally, assume the target user u has a set of face images $\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$, the user-level membership inference aims to distinguish between $\mathcal{U} \cap \mathcal{D}^T_{train} \neq \varnothing$ (member user) and $\mathcal{U} \cap \mathcal{D}^T_{train} = \varnothing$ (non-member user), where $\mathcal{D}^T_{train}$ is the training dataset of $\mathcal{M}_T$. This is different from the classic *sample-level* membership inference that aims to determine whether a specific face image was used to train the target model, i.e., $u_i \in \mathcal{D}^T_{train}$ (member sample) or $u_i \notin \mathcal{D}^T_{train}$ (non-member sample).

Auditing Scenario. Facial recognition systems are often trained by computer vision companies and sold to individual users or other companies for deployment. The model developer of the facial recognition system might collect face images from the Internet and misuse these face images without the data owners' consent. Users who want to audit potential misuse of their face images could use Face-Auditor as a privacy-auditing tool. Note that Face-Auditor is unnecessary to be trained by individuals. Alternatively, a third party with legal access to facial images (such as a qualified Auditing-ML-as-a-Service company, law enforcement, or government agency) can purchase the well-known facial recognition systems in the market and provides (free or charged) auditing services to individuals. By doing so, the third-party entity can ensure auditing accuracy and efficiency, making it more convenient for users who want to audit their face images. Individuals can quickly check if their face images are being used without their consent and take appropriate actions, such as reporting to the authorities or suing the model developer per the protection of privacy regulations [1, 2, 11].

Auditor's Capabilities. The auditor has a basic knowledge of the facial recognition model, such as metric scores, input format, etc. To mimic the real-world application, we consider the most challenging setting where the auditor only has *black-box* access to the target model. We assume the auditor can obtain an auxiliary face image dataset. Note that the auxiliary dataset does not need to contain face images from the same set of users or the same distribution as the target model; thus, the auditor can utilize some

online public datasets to build Face-Auditor, which is practical in real-world applications. In the auditing phase, the auditor does not need access to the specific face images used to train the target model; instead, it only needs to take a few available face images of the target user. Furthermore, the auditor can design their support set (legitimate users) and query set to audit the target model.

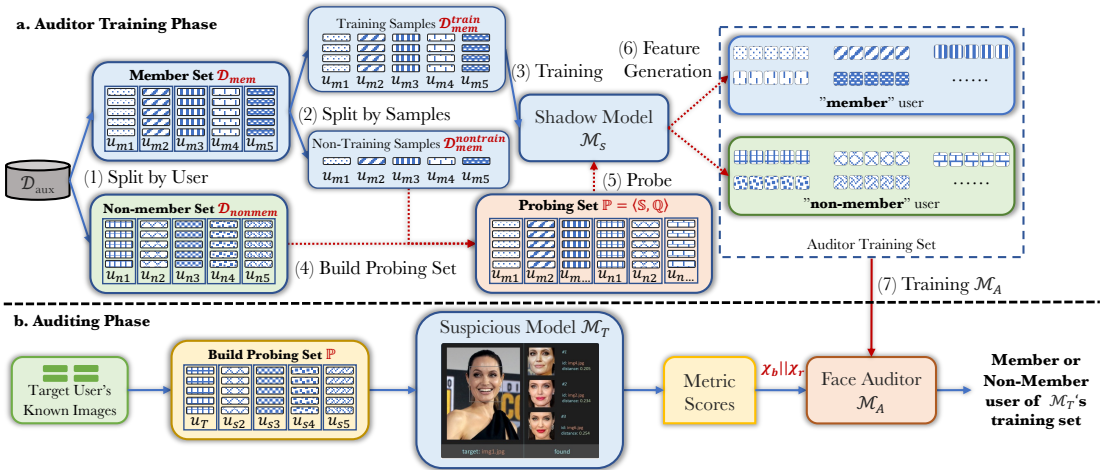


Figure 6.1: Overview of Face-Auditor.

6.1.1 Face-Auditor

There are two phases of Face-Auditor: training and auditing. The *training phase* is composed of seven steps: (1) The auditor splits its auxiliary dataset into (disjoint) training set and testing set by class (here, a class represents a user). (2) The training set is further split into training and non-training samples. (3) The training samples are used to train a shadow model. (4) The non-training samples and testing set form a probing set. (5) We use the probing set to query the shadow model and collect the model outputs. (6) The outputs are labeled by member or non-member, depending on whether the input is from the non-training samples or the testing set. (7) We train a supervised binary classifier as our auditing model. In *auditing phase*, the auditor builds a probing set with known images from the target users and then queries a target facial recognition model $\mathcal{M}_T$ and collects the corresponding outputs (i.e., distance scores) as the auditing feature. Feeding these feature vectors into Face-Auditor, the auditor gives a prediction of member or non-member user.

Figure 6.1 illustrates the overall workflow of Face-Auditor. Generally, there are two

phases, *auditor training* and *target user auditing*. The auditor training phase aims to train a binary classifier that can distinguish between member users and non-member users. The general idea is to use the auxiliary dataset $\mathcal{D}_{aux}$ to train a *shadow model* that mimics the behavior of the target model. We then design a *probing set* (consisting of support set and query set) to query the shadow model and generate a set of *similarity scores* (between support set and query set), which serves as features to train the auditor model $\mathcal{M}_A$. While most of the existing studies on membership inference are based on the shadow model paradigm [145, 132, 151, 65, 105], the main challenge lies in constructing the attack/audit features for the attack model. For the sample-level membership inference against classification models, the attack features are constructed by feeding the target samples to the target model independently and using the output posteriors as attack features. On the other hand, in the user-level few-shot setting, the auditor *does not have the exact images* that are used to train the target model. Thus, we need to carefully design a probing set to query the target model and combine the similarity scores as audit features.

In the auditing phase, the auditor collects a set of new face images from the target user and builds a probing set to query the target model. The auditor then collects the similarity scores returned by the target model as the auditing features and feeds them to $\mathcal{M}_A$, which gives a prediction of the member or non-member user.

6.1.2 Auditor Training Phase

Training the Shadow Model. Assume the auxiliary dataset $\mathcal{D}_{aux}$ contains face images of U users, and each user has I face images. We first split $\mathcal{D}_{aux}$ into two disjoint datasets by users, namely member dataset $\mathcal{D}_{mem}$ and nonmember dataset $\mathcal{D}_{nonmem}$. Recall that, for member users, Face-Auditor does not need to have access to the specific images used to train the auditor model; thus, for the member dataset $\mathcal{D}_{mem}$, we further split it (by sample) into two disjoint parts, $\mathcal{D}_{mem}^{train}$ and $\mathcal{D}_{mem}^{nontrain}$. We use $\mathcal{D}_{mem}^{train}$ to train the shadow model and use $\mathcal{D}_{mem}^{nontrain}$ and $\mathcal{D}_{nonmem}$to construct the probing set. To be more clear, all users in $\mathcal{D}_{mem}$ are the member users, while images in $\mathcal{D}_{mem}^{train}$ are member samples, and images in $\mathcal{D}_{mem}^{nontrain}$ are non-member samples. We follow the procedure described in Section 2.4.2 to construct the support and query set to train the shadow model $\mathcal{M}_s$.

Constructing the Probing Dataset. Unlike classical classification models that take a single image as input and output posteriors, few-shot learning models require a support

set $\mathbb{S}$ and a query set $\mathbb{Q}$ as input and output a sequence of similarity scores (as described in Section 2.4.2). Consequently, generating an auditing feature for few-shot learning is more complex than traditional membership inference attacks against classification models. To improve auditing performance, the auditor must carefully design the support set $\mathbb{S}$ and query set $\mathbb{Q}$, rather than directly feeding the training and testing datasets to the shadow model $\mathcal{M}_s$ to obtain posteriors. For ease of presentation, we call the combination of the support set and query set as probing set $\mathbb{P} = \langle \mathbb{S}, \mathbb{Q} \rangle$.

Since the architecture of SiameseNet is slightly different from ProtoNet and RelationNet, we need to design different probing sets for them.

- **SiameseNet.** As discussed in Section 2.4.2, the SiameseNet model processes the support set separately, which leads to its probing set consisting of a 1-way-ℓ-shot support set and multiple query images. For each probe, we set both the support set and the query images from the same target user (the user to be audited, who may be a member from $\mathcal{D}_{mem}^{nontrain}$ or a non-member from $\mathcal{D}_{nonmem}$).
- **ProtoNet & RelationNet.** Unlike SiameseNet, both ProtoNet and RelationNet takes the support set and query images together as input, forming a k-way-ℓ-shot support set. We assign the first class of the support set to the target user and select the query images from that user. The remaining classes in the support set can be selected from any user, as the similarity scores between these classes and the query images are not used to generate the auditing features.

Generating the Auditing Feature. We use the *similarity scores* between the query image and the support set returned by the shadow model as the *basic auditing feature* χ_b. We use q images in the query set $\mathbb{Q}$, resulting in an auditing feature vector of length q.

To further improve the auditing performance, we consider using the image-level similarity between the query image and the support set as additional reference information, referred to as *reference auditing feature* χ_r. In summary, the auditing feature χ is a concatenation of the basic auditing feature and the reference auditing feature, i.e., $\chi = \chi_b || \chi_r$. We consider three types of image-level similarity metrics: Directly compare the similarity between image pixels (MSE and CosSim), compare the structural similarity between images (SSIM), and use a deep neural network to compare (LPIPS). Denote the pixel matrix of two images as X and Y, and four metrics can be described as follows.

- **MSE (Mean Square Error).** We first represent the image pair as two pixel vectors X and Y, the MSE of these two images is calculated as $\frac{1}{N}\sum_{i=1}^{N}(X_i - Y_i)^2$, where N is the total number of pixels. A smaller MSE indicates higher similarity.
- **CosSim (Cosine Similarity).** For two pixel vectors X and Y, the CosSim is calculated as $\frac{X \cdot Y}{||X||||Y||}$, where $\cdot$ represents an inner product of two vectors and $||\cdot||$ presents the cardinality of a vector. The values of CosSim are in the range of [-1, 1]. A larger CosSim indicates higher similarity.
- **SSIM (Structural Similarity Index Measure) [171].** It compares two images by considering luminance, contrast, and structure. Formally, $SSIM(X,Y) = \ell(X,Y)^{\alpha} \cdot c(X,Y)^{\beta} \cdot s(X,Y)^{\gamma}$, where $\ell(\cdot)$, $c(\cdot)$, $s(\cdot)$ represent luminance, contrast, structure respectively, and α, β, γ are weight parameters. A larger SSIM value indicates higher similarity.
- **LPIPS (Learned Perceptual Image Patch Similarity) [189].** The general idea is to use a pretrained convolutional model to transform the two images X and Y into embeddings, normalize the activations in the channel dimension, and take the ℓ_2 distance. We then average across spatial dimensions and across all layers. We use VGG16 as the default backbone convolutional model of the LPIPS model. A larger LPIPS indicates higher similarity.

We conduct empirical experiments in Section 6.2.3 to show that the reference information can effectively improve the auditing performance, and cosine similarity achieves relatively better performance in most settings.

Training the Auditing Model. For all the few-shot learning models, we use $\mathcal{D}_{train}^{nonmem}$ and $\mathcal{D}_{test}$ to construct the probing set for member users and non-member users, respectively. We use a three-layer multi-layer perceptron (MLP) with 100 hidden neurons as the auditing model.

6.1.3 Auditing Phase

To determine whether a target user's face images are used to train the target model, the auditor only needs to take multiple face images from the target user. Note that these face images are not necessarily used to train the target model. The auditor then uses the same strategy as the training phase to construct the probing set $\mathbb{P}$ and generate the auditing feature χ. Finally, the auditing feature is fed to the auditing model to determine the membership status of the target user.

6.2 Evaluation

In this section, we first describe the experimental setup in Section 6.2.1 and evaluate the overall auditing performance in Section 6.2.2. We then validate the effectiveness of the reference information and investigate the effectiveness of different image-level similarity metrics in Section 6.2.3. Third, we evaluate the impact of different hyperparameters on the auditing performance in Section 6.2.4. We show the transferability of Face-Auditor in Section 6.2.5.

6.2.1 Evaluation Settings

Face Datasets. We perform experiments on four widely used real-world face image datasets: UMDFaces [19], WebFace [183], VGGFace2 [27], and CelebA [97]. Since the number of face images for all users is highly unbalanced, to make the experimental results comparable, we filter out the users with a number of images less than 100 (except for CelebA). For the users having more than 100 images, we randomly sample 100 images for them. We resize all images to 96×96 and evaluate the performance of both the target model and the auditing model.

Under the setting of Figure 6.1, which indicates half of the images from 40% users are used to train the shadow/target models, we randomly select 10% images from the 40% to generate member labels and 10% testing images for generating non-member labels, they use the data to train and evaluate the performance of Face-Auditor. We summarize the dataset split in Table 6.1. The dataset was split into two halves for the shadow model and target model, with users being divided equally between them. We allocated 80% of the users for $\mathcal{D}_{mem}$ and the remaining 20% for $\mathcal{D}_{nonmem}$. Within the training set, each user's images were split into two equal parts. One part (50% as $\mathcal{D}_{mem}^{train}$) was used to train the shadow/target model, while the other part (50% as $\mathcal{D}_{mem}^{nontrain}$) was used to generate the member labels. This split ensured sufficient training data for a well-performed shadow/target model. We keep member and nonmember labels balanced for a fair and accurate evaluation of the performance of Face-Auditor.

Table 6.1: Dataset split in detail.

Dataset after Prepossessing			**Target/Shadow Model**		**Auditing Model**	
Dataset ($\mathcal{D}$)	**#. Users** (U)	**#. Images per User** (I)	**#. Training Images** $(40\% * U) * (50\% * I)$	**#. Testing Images** $(10\% * U) * (50\% * I)$	**#. Training Images** $(10\% * U) * (50\% * I)$	**#. Testing Images** $(10\% * U) * (50\% * I)$
UMDFaces	200	100	4,000	1,000	1,000	1,000
Webface	827	100	16,520	4,130	4,130	4,130
VggFace2	5,257	100	105,140	26,285	26,285	26,285
CelebA	6,348	20	25,392	6,348	6,348	6,348

Target Models. We experiment on three facial recognition system architectures as introduced in Section 2.4.2, all with the default configurations.

- **SiameseNet.** Following the setting of [83], we implement the SiameseNet with a four-convolution-layer feature extractor with a ReLU and Max-Pooling for each convolution layer to learn complex patterns in the data. The target model is trained with BCE loss and Adam optimizer.
- **ProtoNet.** Following the setting of [147], we implement the ProtoNet with a four-convolution-layer feature extractor with batch normalization and ReLU activation function for each convolution layer. The target model is trained with cross-entropy loss and an SGD optimizer with a step scheduler.
- **RelationNet.** Following the setting of [153], we implement the RelationNet with a four-convolution-layer feature extractor and a two-convolution-layer relation network. The feature extractor and the RelationNet are trained with Adam optimizer with a step scheduler. We use MSE loss to train the metric parameters of RelationNet.

Metrics. We use four metrics to evaluate the performance of Face-Auditor.

- **Accuracy.** We use accuracy to measure the auditing success rate. Concretely, accuracy measures the correctly predicted probing sets to the total probing sets. Higher accuracy means better performance.
- **AUC.** For a binary classification model (our attack model), AUC (the Area Under the Curve) is the measure of the ability of a classifier to distinguish between classes when the decision threshold varies. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. AUC equals 1 indicates perfect prediction, while 0.5 indicates random guessing.
- **F1 Score.** F1 Score is a harmonic mean of *precision* (the proportion of true positive cases to the member classes) and *recall* (the proportion of true positive cases to all correctly predicted classes), which can provide a better measure of the incorrectly classified cases than the accuracy metric. A higher F1 Score indicates better auditing performance.
- **False Positive Rate (FPR).** The false Positive Rate evaluates the proportion of incorrect ownership claims to the total cases. In practice, a higher false positive rate may degrade the credibility of Face-Auditor and cause unnecessary lawsuits. In our case, a lower FPR indicates better auditing performance.

Experimental Settings. Following the classical setting of shadow model-based membership inference [145, 132, 151, 65, 105], we equally split each dataset by users into two disjoint parts, the target set $\mathcal{D}_T$ and auxiliary set $\mathcal{D}_{aux}$. We then split both the target set $\mathcal{D}_T$ and the auxiliary set $\mathcal{D}_{aux}$ as in Section 6.1.2. We train the auditing model on $\mathcal{D}_{aux}$ and evaluate the auditing model on $\mathcal{D}_T$. We evaluate 5-way-5-shot with 5 queries by default and explore the impacts of different parameters in Section 6.2.4.

Table 6.2: Target model performance.

Dataset	**UmdFaces**			**WebFace**			**VGGFace2**			**CelebA**		
$\mathcal{M}_{Target}$	$\mathcal{M}_S$	$\mathcal{M}_P$	$\mathcal{M}_R$	$\mathcal{M}_S$	$\mathcal{M}_P$	$\mathcal{M}_R$	$\mathcal{M}_S$	$\mathcal{M}_P$	$\mathcal{M}_R$	$\mathcal{M}_S$	$\mathcal{M}_P$	$\mathcal{M}_R$
Train Acc.	0.775	0.960	1.000	0.650	0.748	0.800	0.818	0.951	1.000	0.647	0.818	0.940
Test Acc.	0.500	0.794	**0.852**	0.460	0.670	**0.757**	0.767	0.868	**0.943**	0.603	0.802	**0.867**
Overfitting	**0.275**	0.166	0.148	**0.190**	0.078	0.043	0.051	**0.083**	0.057	0.044	0.016	**0.073**

6.2.2 Overall Performance

Target Model Performance. We first investigate the performance of the target models. Table 6.2 illustrates the training accuracy, the testing accuracy, and the overfitting (accuracy gap between training and testing datasets) of three target models trained on four face image datasets. Higher test accuracy means better representation power and higher overfitting indicates a worse generalization ability of the target model. In the table, $\mathcal{M}_S$ represents SiameseNet, $\mathcal{M}_P$ represents ProtoNet, while $\mathcal{M}_R$ represents RelationNet. We first observe that the overfitting level varies across different models but keeps low in most settings. Besides, the RelationNet achieves the best testing accuracy, indicating RelationNet has the best representation power.

Auditing Performance. We then evaluate the overall auditing performance of Face-Auditor. We conduct experiments on three target models trained on four face image datasets and report the auditing performance with four metrics in Figure 6.2. The three model architectures are grouped by dataset and the auditing performance over four different evaluation metrics in each subfigure.

In general, we observe that Face-Auditor achieves good auditing performance for all the target models and datasets. For instance, SiameseNet, ProtoNet, and RelationNet trained on the UMDFaces dataset achieve up to 1.0, 0.80, and 0.85 auditing accuracy, respectively. We further observe that the auditing performance varies on three different target models. We achieve the best auditing performance on SiameseNet and the worst on ProtoNet. This is due to the different memorization power of the target models. The

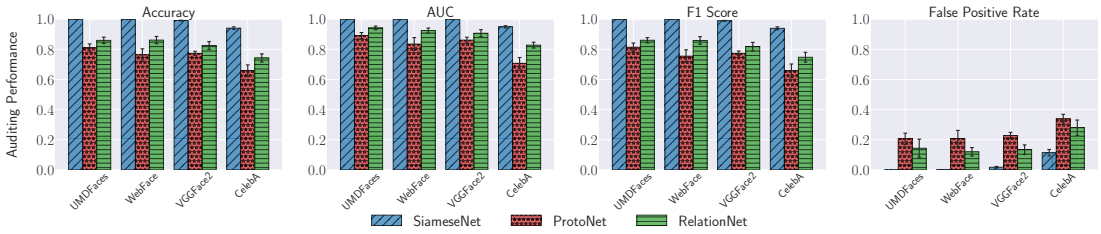


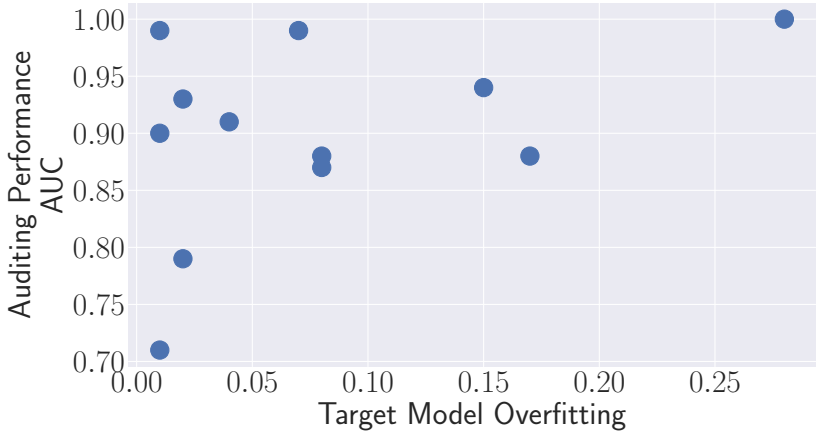
Figure 6.2: Overall auditing performance for four evaluation metrics.

memorization power of different models can be explained by the fact that member users' similarity between face images is optimized in the training process. SiameseNet has the highest memorization power since images of the member users are separately optimized in the training process. In contrast, that of ProtoNet and RelationNet are optimized together with other classes. Comparing ProtoNet and RelationNet, since RelationNet uses a trainable relation module to compute the similarity scores while ProtoNet directly computes the Euclidean distance; thus RelationNet has higher memorization power than ProtoNet.

Comparing different datasets, we observe the best auditing performance on UMDFaces and the worst on CelebA. This is because UMDFaces has the least users (i.e., 200 users in our experiment), and CelebA has the most users (i.e., 6348 users in our experiment). Besides, CelebA only contains 20 images for each user; the samples used to represent a user are much fewer than the other three datasets, thus further increasing the challenge for auditing.

While we observe that **it is easier to infer a target user's membership status when there are fewer users and each user has more samples in the dataset**. In practice, it is unnecessary to train few-shot learning-based facial recognition models on face datasets of more than $3k$ users since the objective of the few-shot learning is to learn the similarity information between classes, and $3k$ users are enough for learning a few-shot learning model.

Impact of Overfitting. Previous studies have shown that overfitting plays a crucial role in launching a successful membership inference [182, 132]. To investigate the impact of overfitting, we provide a scatter plot showing the relation between the overfitting and the auditing performance in Figure 6.3. Twelve dots in each subfigure represent the combination of three target model architectures and four datasets. The Pearson correlation values between auditing performance (for accuracy, AUC, and F1 Score) and overfitting level are 0.412, 0.406, and 0.412, respectively. We observe that higher



(a) AUC

Figure 6.3: Relation between target model overfitting and auditing performance.

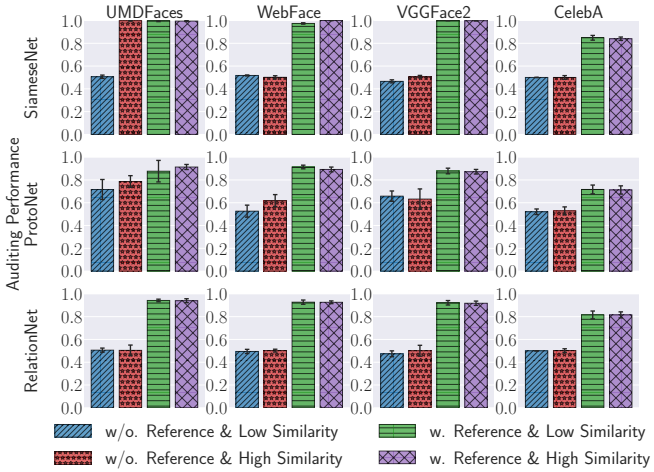
overfitting indeed leads to better auditing performance. Unlike classical sample-level membership inference requiring relatively high overfitting to achieve satisfying inference performance, Face-Auditor **can achieve good auditing performance even when the overfitting level is low**. For instance, when the overfitting level is 0.02, Face-Auditor can achieve 0.93 auditing accuracy. On the other hand, the classical sample-level membership inference can only achieve 0.6 accuracy when the overfitting is 0.02 (see Figure 2 in [132]).

6.2.3 Reference Information

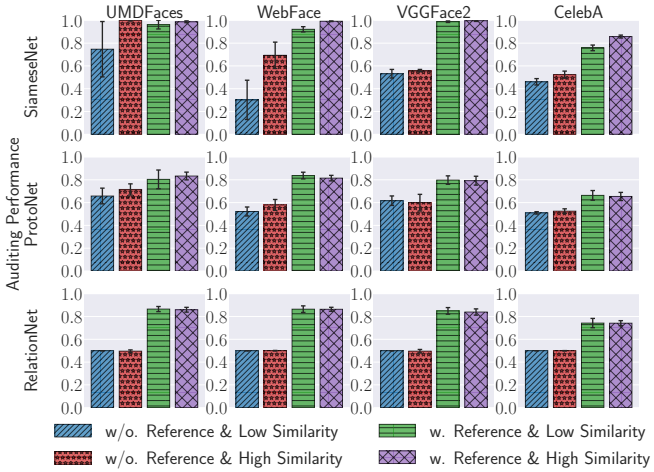
As discussed in Section 6.1.1, the reference information helps to improve auditing performance. In this subsection, we first validate the effectiveness of the reference information, then investigate the impact of different similarity metrics.

Effectiveness. We conduct experiments on four face image datasets and three target models to validate the effectiveness of the reference information. The experimental results in Figure 6.4 illustrate that **exploiting reference information can significantly improve the auditing performance in most of the settings** (by comparing the "w/o." and "w." bars).

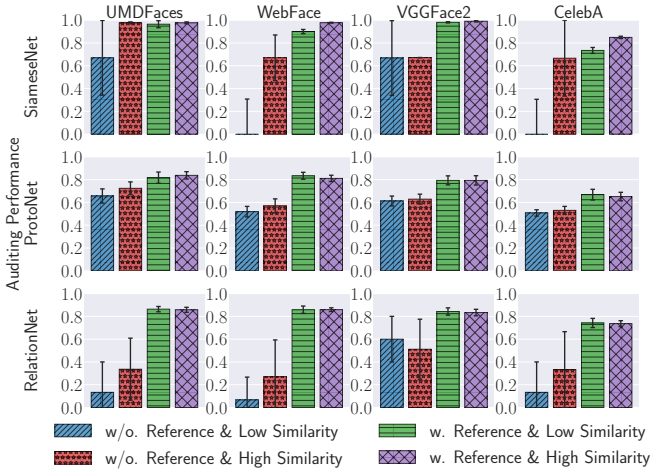
We further explore why the reference information can improve the auditing performance using a t-SNE plot in Figure 6.5. Each red triangle is a member sample, and each blue circle is a non-member sample of the UMDFaces dataset. Specifically, by comparing Figure 6.5c left and right subfigures, we observe that the member and non-member are much further from each other after exploiting the reference information. We also observe



(a) AUC



(b) Accuracy



(c) F1 Score

Figure 6.4: Impact of the reference information and the similarity selection.

different effects of reference information on the three target models. We suspect the reason is that the improvement level by reference information is positively correlated to the memorization power of different models, and ProtoNet has the lowest memorization power in terms of user-level membership inference as discussed in Section 6.2.2.

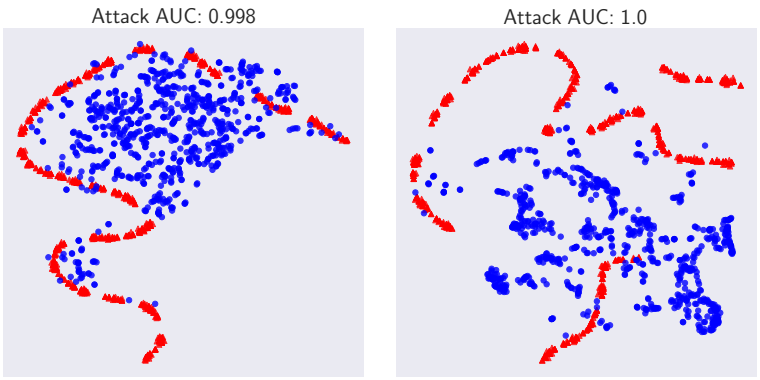
Impact of Similarity. Given the reference information of the raw face images, another question is whether to choose query images with high similarity or low similarity to the support set. To answer this, we compare the auditing performance when choosing the five highest-similarity query images and the five lowest-similarity query images from the testing dataset. The experimental results are shown in Figure 6.4.

By comparing the "Low Similarity" and "High Similarity" bars, we observe that the original similarity between the query images and the support set only slightly impacts the auditing performance on ProtoNet and RelationNet. When auditing the SiameseNet model, high similarity pairs can enhance the auditing performance. Take the SiameseNet trained on CelebA as an example, the "Low Similarity" query images can achieve 0.758 auditing accuracy, while the "High Similarity" query images can achieve 0.858 auditing accuracy.

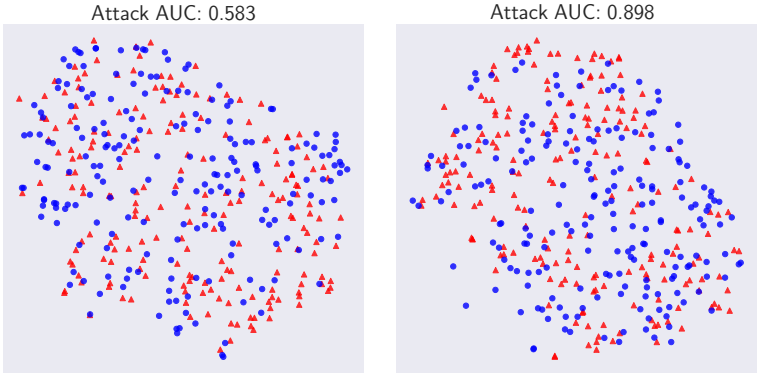
In summary, randomly choosing query images from the testing dataset when constructing the probing set can make the auditing model work well in most cases, but **to achieve the best auditing performance, "High Similarity" images are recommended.**

Choice of Similarity Metrics. We can adopt multiple metrics to measure the similarity between the target image and the support set as discussed in Section 6.1.2. Figure 6.6 illustrates the auditing performance when using different similarity metrics to generate the reference information. We first observe that all four metrics can achieve relatively high auditing performance on SiameseNet. Regarding the auditing performance on ProtoNet and RelationNet, the performance variance increases among different datasets. **In general, CosSim can achieve the best and the most stable performance in most of the settings.** We posit the reason is that it generates a bounded value (-1 to 1), which tends to be consistent with the normalized input feature of Face-Auditor.

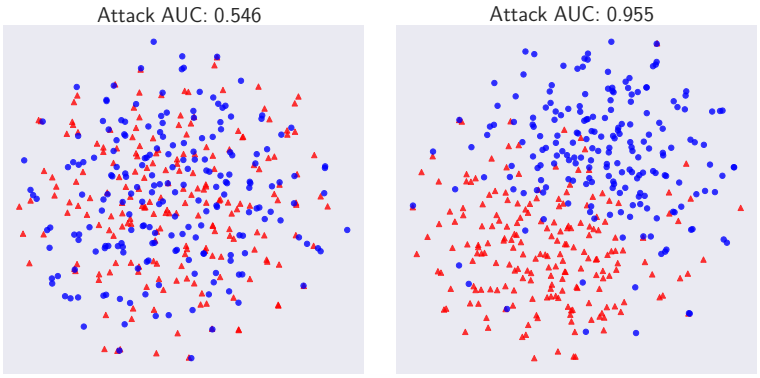
Impact of the Embedding Extractor. Recall that all three target models contain an embedding extractor that maps an image to an embedding. Tian et al. [159] showed that in few-shot learning, the quality of the embedding extractor has some impact on the target model performance. We now investigate the impact of the embedding extractor



(a) SiameseNet w/o.& w. reference.



(b) ProtoNet w/o.& w. reference.

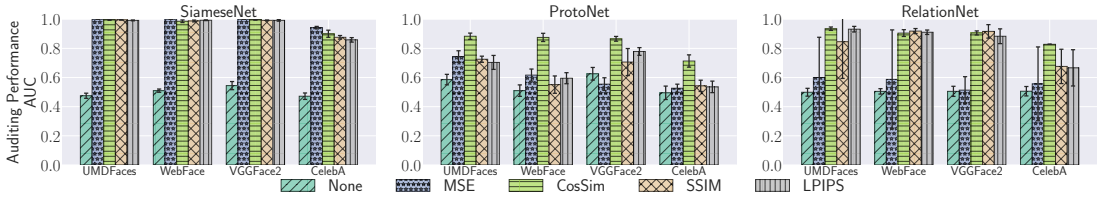


(c) RelationNet w/o.& w. reference.

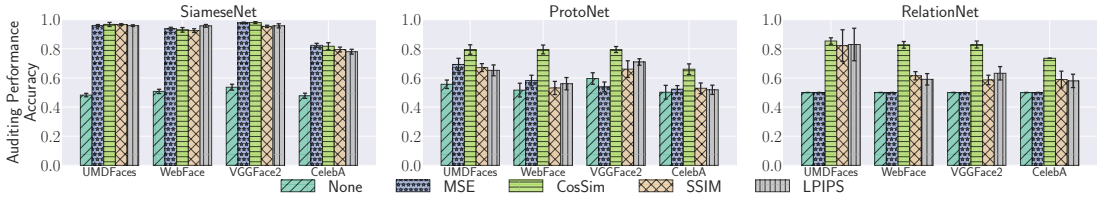
Figure 6.5: T-SNE visualization on the impact of reference information.

on the auditing performance for the following architectures:

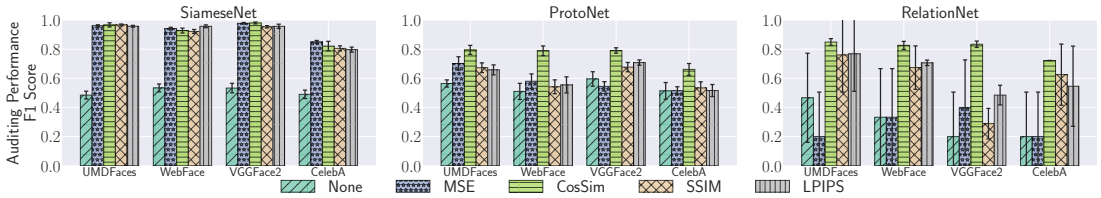
- **MobileNet** [133] is an efficient model pretrained on the ImageNet dataset and widely adopted for mobile and embedded vision applications.



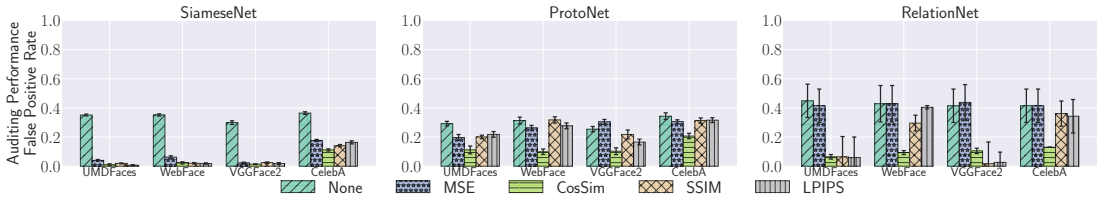
(a) AUC



(b) Accuracy



(c) F1 Score



(d) False Positive Rate

Figure 6.6: Auditing performance when using different similarity metrics to generate the reference information.

- **ResNet18** [66] is a public model with a deep residual network structure and pretrained on the ImageNet dataset.
- **ResNet50** [66] is a public model with a deeper residual network structure and pretrained on the ImageNet dataset.
- **GoogleNet** [154] is a type of convolutional neural network based on the Inception architecture, which allows the network to choose between multiple convolutional filter sizes in each block.

Table 6.3 illustrates the experimental results. In general, we observe that the embedding extractor can help improve the target model performance but only slightly

Table 6.3: Evaluation on different feature extractors F on four datasets and three target models.

Dataset	Target Model / Feature Extractor F	SiameseNet $\mathcal{M}_{Target}$ Acc.	SiameseNet $\mathcal{M}_{Auditor}$ AUC	ProtoNet $\mathcal{M}_{Target}$ Acc.	ProtoNet $\mathcal{M}_{Auditor}$ AUC	RelationNet $\mathcal{M}_{Target}$ Acc.	RelationNet $\mathcal{M}_{Auditor}$ AUC
UMDFaces	SimpleCNN	0.568	0.995 ± 0.002	0.806	0.853 ± 0.041	0.856	**0.900 ± 0.021**
	MobileNet	0.593	**0.999 ± 0.001**	0.699	0.823 ± 0.047	0.748	0.779 ± 0.032
	GoogleNet	0.535	0.993 ± 0.003	0.710	**0.885 ± 0.033**	0.735	0.839 ± 0.059
	ResNet18	0.593	0.992 ± 0.003	0.729	0.850 ± 0.028	0.848	0.862 ± 0.028
	ResNet50	0.595	0.996 ± 0.002	0.733	0.830 ± 0.021	0.782	0.884 ± 0.030
WebFace	SimpleCNN	0.398	0.956 ± 0.015	0.583	0.876 ± 0.023	0.711	0.900 ± 0.022
	MobileNet	0.545	**0.994 ± 0.007**	0.541	**0.890 ± 0.021**	0.786	**0.907 ± 0.033**
	GoogleNet	0.485	0.983 ± 0.008	0.568	0.843 ± 0.037	0.725	0.873 ± 0.051
	ResNet18	0.515	0.989 ± 0.006	0.593	0.847 ± 0.027	0.688	0.869 ± 0.043
	ResNet50	0.477	0.991 ± 0.007	0.548	0.793 ± 0.034	0.694	0.798 ± 0.036
VGGFace2	SimpleCNN	0.662	0.998 ± 0.002	0.882	0.867 ± 0.020	0.934	0.915 ± 0.021
	MobileNet	0.685	0.993 ± 0.001	0.868	0.867 ± 0.025	0.917	0.915 ± 0.011
	GoogleNet	0.640	**0.998 ± 0.004**	0.877	**0.870 ± 0.026**	0.903	0.921 ± 0.039
	ResNet18	0.677	0.995 ± 0.002	0.903	0.845 ± 0.034	0.935	0.892 ± 0.024
	ResNet50	0.637	0.997 ± 0.004	0.923	0.852 ± 0.023	0.952	**0.933 ± 0.036**
CelebA	SimpleCNN	0.522	**0.858 ± 0.019**	0.837	0.698 ± 0.045	0.886	0.830 ± 0.028
	MobileNet	0.590	0.812 ± 0.022	0.793	**0.721 ± 0.052**	0.872	0.853 ± 0.048
	GoogleNet	0.562	0.777 ± 0.020	0.764	0.700 ± 0.038	0.845	**0.871 ± 0.081**
	ResNet18	0.585	0.796 ± 0.020	0.804	0.690 ± 0.052	0.867	0.834 ± 0.033
	ResNet50	0.555	0.785 ± 0.014	0.869	0.700 ± 0.050	0.902	0.845 ± 0.018

impacts the auditing performance in most settings.

6.2.4 Hyperparameters

Recall that we need carefully design a probing set $\mathbb{P} = \langle \mathbb{S}, \mathbb{Q} \rangle$ to achieve an optimal auditing performance. We have three important hyperparameters in the probing set i.e., the number of ways k, the number of shots l, and the number of queries q. We investigate their impacts on auditing performance in Section 6.2.4.1, Section 6.2.4.2, and Section 6.2.4.3.

6.2.4.1 The Number of Ways k

In Figure 6.7, we observe only a slight decrease in the auditing accuracy (less than 4%) as we increase the number of ways k in the support set for all three model architectures. This parameter affects the search space of the target model (we observe a worse target model performance in more ways of the support set), but it does not significantly affect the auditing model, as we only use the largest similarity score to form the auditing feature. This also explains why Face-Auditor can work when thousands of users are in the training set of the target model.

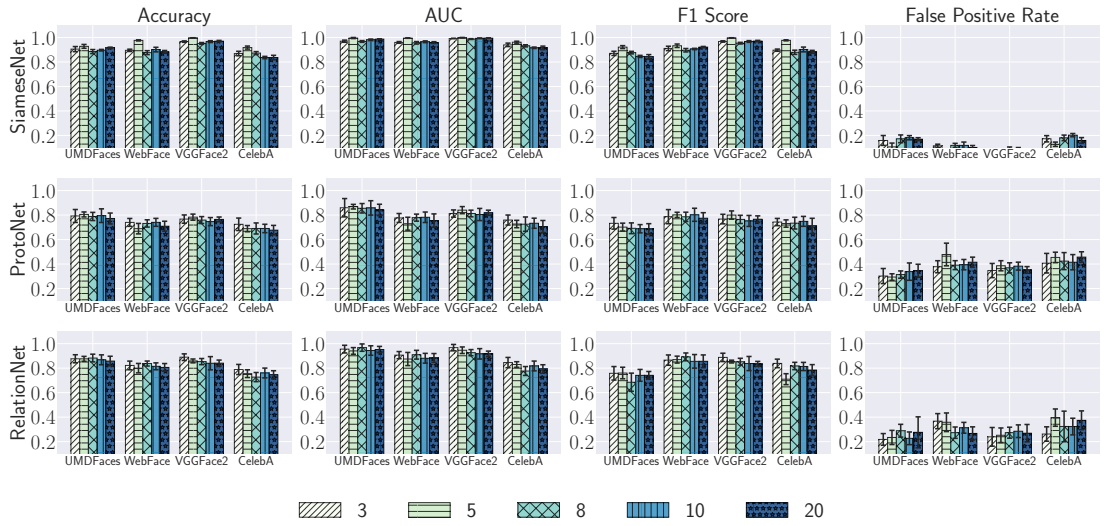


Figure 6.7: Number of ways (k) in the support set.

6.2.4.2 The Number of Shots l

The results in Figure 6.8 show that increasing the number of shots in the support set leads to a more precise description of a user, as reflected by better target model performance on ProtoNet and RelationNet. However, since SiameseNet only takes image pairs as input, the target model performance is unaffected by the number of shots. Interestingly, we found that the auditing performance consistently improved as the number of shots increased for the SiameseNet. We believe this is because ProtoNet and RelationNet represent each user's multiple images as a whole and calculate inter-class distances to discriminate between multiple users. When generating the posteriors, ProtoNet and RelationNet already consider the influence of multiple shots, resulting in each user being represented as a single vector for comparison, regardless of the number of shots in the support set. On the other hand, SiameseNet takes image pairs per probe, and more shots indicate more diverse probes from a single user. This allows for capturing a user's character from multiple probes, leading to an increase in auditing performance.

6.2.4.3 The Number of Query Images q

We investigated the impact of the number of query images q on three datasets with 100 images per user in their preprocessed dataset, providing a wide range of q values to explore. Our results, shown in Figure 6.9, demonstrate that auditing performance

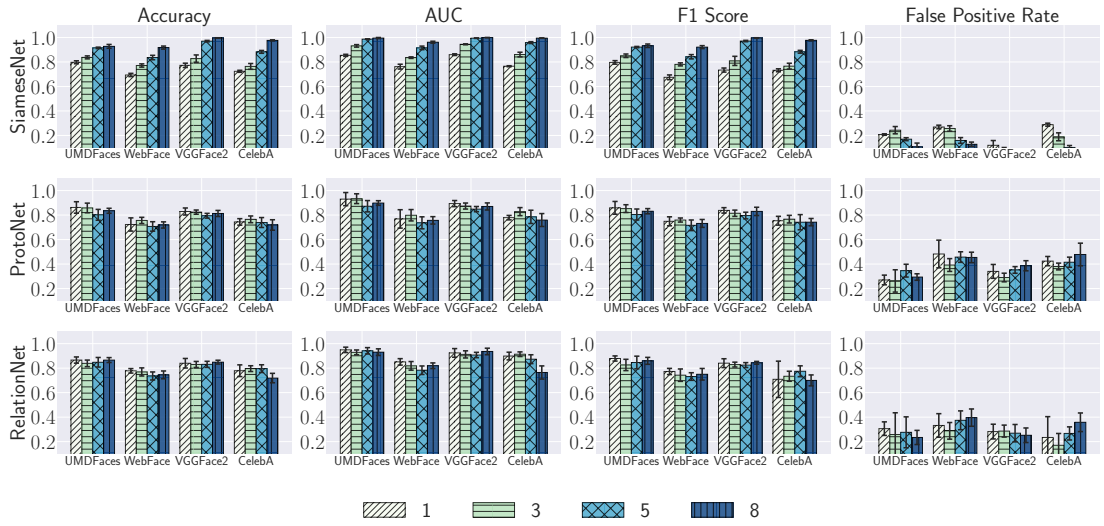


Figure 6.8: Number of shots (l) in the support set.

improves and the false positive rate decreases as the number of query images increases. The rationale is that more query images lead to a broader auditing feature that captures more information about the user, and more images of a user can help distinguish them from other users. The increasing trend is more pronounced for RelationNet and ProtoNet, suggesting that more diverse queries can reveal more information about the underlying training data of the few-shot facial recognition models, especially when the model has a higher memorization ability.

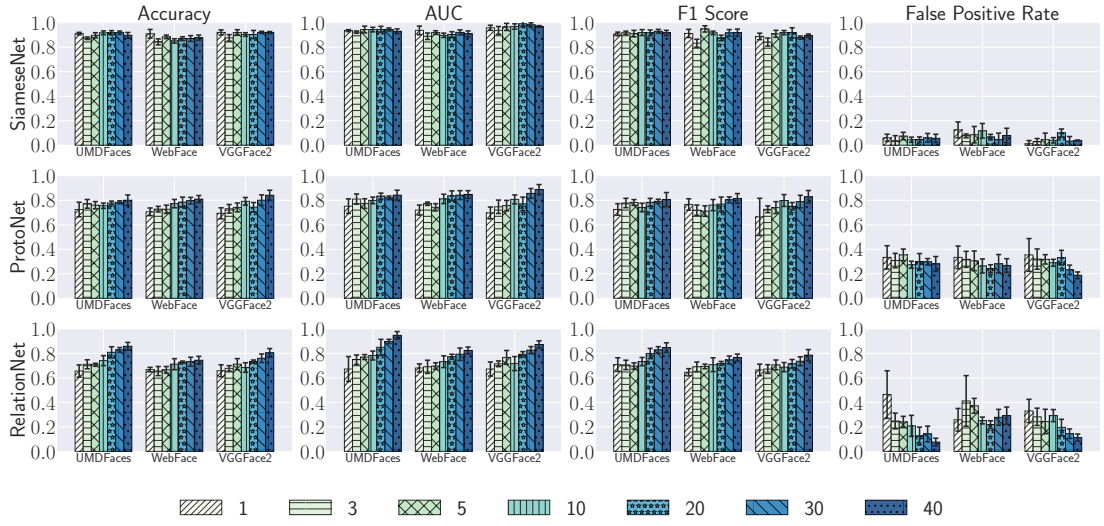
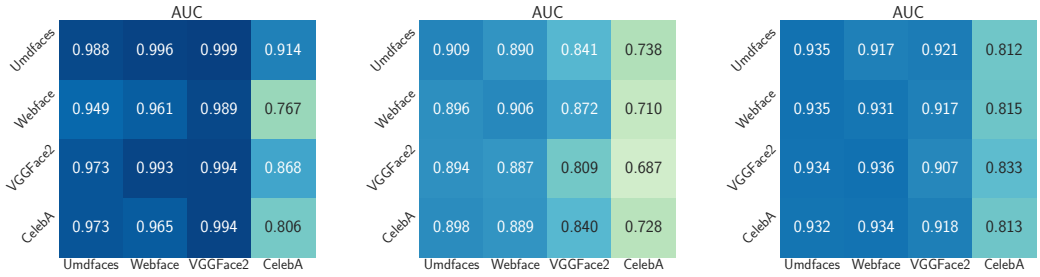


Figure 6.9: Number of query images (q).

6.2.5 Transferability

In practice, the auditor might not be aware of the target model's architecture or its training data distribution. Thus, in this section, we aim to evaluate the transferability of Face-Auditor. We first evaluate the dataset transferability when the training data of the shadow model comes from a different distribution than the target model and then evaluate the model transferability when the architecture of the shadow model is different from the target model.

Dataset Transferability. We conduct experiments on three target models. For each target model, we use one dataset as the auxiliary dataset and the other three datasets as target datasets. In total, we have 16 combinations. We report the experimental results for the AUC metric in Figure 6.10. The x-axis is the dataset used to train the shadow models and probe the target/shadow models and the y-axis is the dataset used to train the target models.



(a) SiameseNet (b) ProtoNet (c) RelationNet

Figure 6.10: Auditing performance (measured by AUC) under datasets transfer.

In general, we observe that Face-Auditor maintains a good performance when the target dataset and the auxiliary dataset come from different distributions in most of the cases. For instance, when the auxiliary dataset is VGGFace2, and the target dataset is WebFace, we can achieve up to 0.954 auditing accuracy, only 0.029 lower than the same distribution auxiliary dataset.

Two reasons can explain the high auditing performance under dataset transfer settings. On the one hand, the uniqueness of human faces does not change substantially. Once a user's image is seen during the training process of the target model, it is easy to distinguish it from those never seen before. Which also shows the severe privacy risks of facial recognition models. On the other hand, Face-Auditor is trained on a shadow dataset with no user overlap as the target model's training dataset. **The disjoint classes split forces the auditing model to not rely on the overfitting intuition to**

determine the membership status but learn to discriminate from the metric scores' internal correlations.

Model Transferability. We conduct experiments between RelationNet and ProtoNet due to the fact that they share the same input data format and report the experimental results in Figure 6.11. In each subfigure, the x-axis represents the target model, and the y-axis represents the shadow model.

We observe that the auditing performance slightly decreases when the architecture of the shadow model is different from the target model. The drop is significant when using RelationNet as the shadow model to audit ProtoNet. On the contrary, using ProtoNet as the shadow model to audit RelationNet can achieve better performance. We suspect the reason is that the linear Euclidean metric of ProtoNet is a particular case of non-linear metric, which supports a pre-trained linear model that still works in most cases.

6.3 Robustness

In practice, the target model might be equipped with different techniques to preserve the privacy of the training data. Therefore, we conduct experiments to validate the robustness of Face-Auditor when the training images or the target models are protected. Concretely, we consider three representative privacy-preserving mechanisms in a general ML model pipeline: Input perturbation (perturb the training images), training perturbation (perturb the training gradient by enforcing differential privacy), and output perturbation (perturb the output similarity scores). We also show the robustness of Face-Auditor under an adaptive attack setting, where the target model's output is perturbed specifically to evade the auditing from Face-Auditor.

In this section, we investigate the robustness of Face-Auditor when the target models' framework is perturbed to evade auditing. Concretely, we consider four defense mechanisms: *Input perturbation* in Section 6.3.1 (perturb the training images), *training perturbation* in Section 6.3.2 (perturb the training gradient by enforcing differential privacy), and *output perturbation* in Section 6.3.3 (perturb the similarity scores returned by the target models). In the end, we also explore an adaptive adversary in Section 6.3.4.

6.3.1 Input Perturbation

Methodology. Multiple techniques are proposed to perturb the face images before training the facial recognition models [31, 45] and prevent the face images from be-

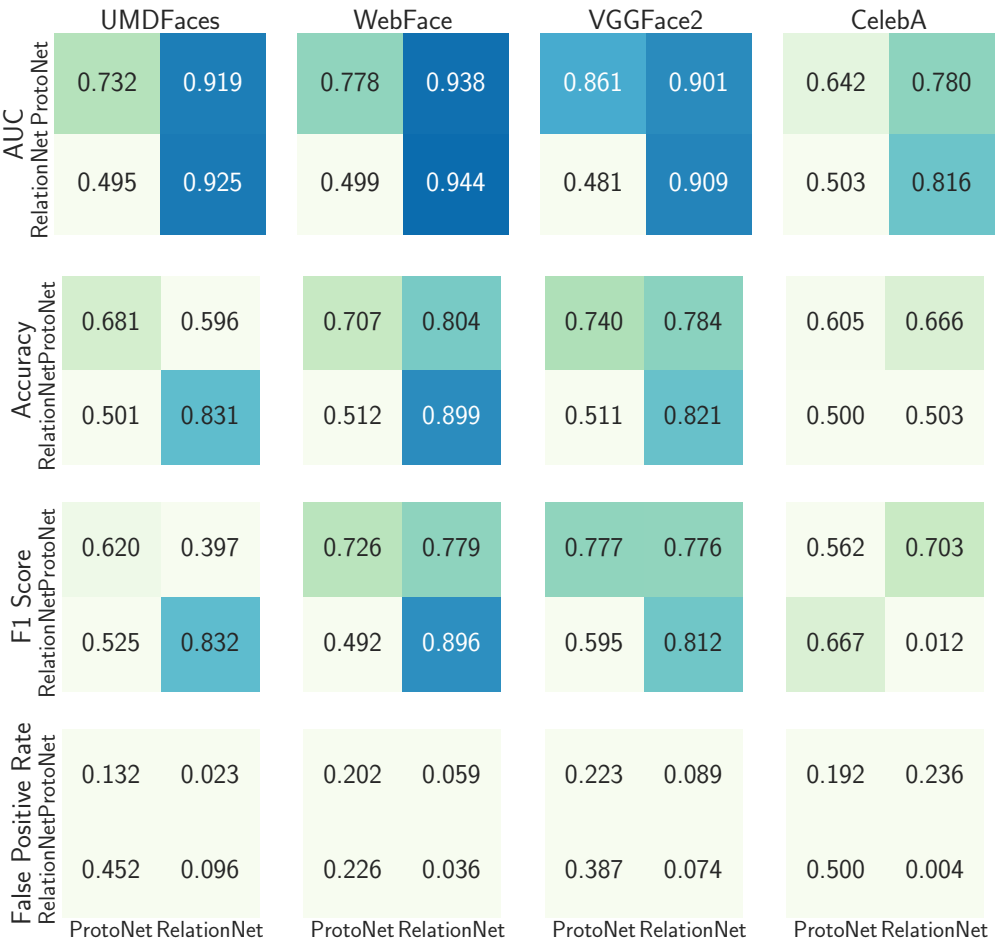


Figure 6.11: Auditing performance (measured by four metrics) under model transfer.

ing misused. In our experiments, we consider a recently proposed technique called Fawkes [137]. The general idea of Fawkes is to add imperceptible noise to the target images that drive the embeddings of the face images to deviate from that of the raw face images. According to its homepage, it has been downloaded more than 840,000 times and used in various applications.

Visualization of Adversarial Perturbation. The original Fawkes provides three adversarial perturbation levels [137], aims to strike a better model utility and face image protection ability. To give a direct impression of the input perturbation, we show a visualization of different perturbation levels in Figure 6.12.

Evaluation. The open-sourced Fawkes implementation[1] allows us to choose three

[1] https://github.com/Shawn-Shan/fawkes

Figure 6.12: An illustration of images under different levels of adversarial noise perturbation.

perturbation levels: Low, middle, and high. We experiment on the UMDFaces dataset and three target model architectures. Concretely, we use Fawkes with three perturbation levels to prepossess all the images in UMDFaces and then use the same pipeline introduced in Section 6.1.2 to build our shadow model and auditing model.

We report the target model performance and the auditing performance under different perturbation levels in Table 6.4. The higher the perturbation level, the better the privacy-preserving level.

We observe a slight performance drop of the target model when applying high-level perturbation, indicating the perturbed face images (especially under high-level perturbation) are more difficult to train. Regarding the auditing performance, we only observe a slight drop (the drop percentage is less than 6%), which indicates that Face-Auditor **is robust to input perturbation**.

Table 6.4: Auditing performance under input perturbation on UMDFaces.

Target Model	**SiameseNet**		**ProtoNet**		**RelationNet**	
Perturbation Level	$\mathcal{M}_{Target}$ Acc. (Δ)	$\mathcal{M}_{Auditor}$ Acc.	$\mathcal{M}_{Target}$ Acc. (Δ)	$\mathcal{M}_{Auditor}$ Acc.	$\mathcal{M}_{Target}$ Acc. (Δ)	$\mathcal{M}_{Auditor}$ Acc.
Original	**0.500**	**0.991 ± 0.000**	**0.782**	**0.879 ± 0.000**	**0.847**	**0.961 ± 0.000**
Low	0.485 (-0.015)	1.000 ± 0.001	0.803 (+0.021)	0.785 ± 0.073	0.874 (+0.027)	0.914 ± 0.019
Middle	0.496 (-0.004)	0.993 ± 0.002	0.843 (+0.061)	0.852 ± 0.032	0.877 (+0.030)	0.903 ± 0.017
High	0.477 (-0.023)	0.973 ± 0.004	0.777 (-0.005)	0.843 ± 0.027	0.838 (-0.009)	0.913 ± 0.021

6.3.2 Training Perturbation

Methodology. A generic approach to protect users' data privacy is differential privacy (DP), which guarantees that any sample in the input dataset has a limited impact on

the final output. For machine learning models, the most representative DP algorithm is Differentially-Private Stochastic Gradient Descent (DP-SGD) [12]. In general, DP-SGD adds Gaussian noise to gradient g during the target ML model's training process, i.e., $\tilde{g} = g + \mathcal{N}\left(0, \Delta_g^2 \sigma^2 \mathbf{I}\right)$. Note that there is no prior knowledge to determine the influence of a single training sample on the gradient g; thus, the sensitivity of g cannot be directly computed. To address this problem, DP-SGD proposes to bound the ℓ_2 norm of the gradient to C by clipping g to $g/\max\{1, ||g||_2/C\}$. This clipping ensures that if $||g||_2 \leq C$, g is preserved; otherwise, it gets scaled down to be the norm of C. As such, the sensitivity of g is bounded by C. Note that we aim to show the defensive performance of adding perturbation in the training process of the target model. Besides, existing user-level DP mainly focus on the federated learning setting [102, 104]. They do not fit to few-shot learning paradigms.

Evaluation. We conduct experiments on four datasets and three target models. The experimental results are in Table 6.5. We report the target model performance and auditing performance for three different privacy-preserving levels, i.e., Low, Middle, and High. Original means the target model without enforcing DP-SGD. The privacy budgets for three privacy levels are 1.02, 3.32, and 47.35.

We first observe that **DP-SGD has a severe impact on the target model performance**. We further observe variations in the auditing performance across the three model architectures: SiameseNet is more sensitive to DP-SGD while ProtoNet and RelationNet are less sensitive. Take VGGFace2 as an example. Applying a high-level noise to the training phase of SiameseNet makes target model accuracy drop by 55%, while the auditing accuracy only drops 1.4%. On the other hand, the auditing accuracy of ProtoNet and RelationNet remains almost the same.

6.3.3 Output Perturbation

Methodology. Another approach to protecting ML models from inference attacks is adding perturbations to the target models' outputs. In this subsection, we evaluate the robustness of our auditing model when the similarity scores returned by the target model are perturbed. We implement this defense by adding a zero-mean Laplace noise with a standard deviation δ to the target model's outputs. The auditing model feature is built on these noise-perturbed similarity scores or posteriors.

Evaluation. We conduct experiments on all four datasets and three target models. The experimental results are in Figure 6.13. The x-axis represents different noise levels

Table 6.5: Auditing performance under training perturbation.

Dataset	Target Model Perturbation Level	SiameseNet $\mathcal{M}_{Target}$ Acc.	SiameseNet $\mathcal{M}_{Auditor}$ AUC	ProtoNet $\mathcal{M}_{Target}$ Acc.	ProtoNet $\mathcal{M}_{Auditor}$ AUC	RelationNet $\mathcal{M}_{Target}$ Acc.	RelationNet $\mathcal{M}_{Auditor}$ AUC
UMDFaces	**Original**	**0.500**	**0.996 ± 0.003**	**0.793**	**0.883 ± 0.022**	**0.852**	**0.935 ± 0.012**
	Low	0.357	0.918 ± 0.009	0.454	0.919 ± 0.000	0.212	0.941 ± 0.018
	Middle	0.370	0.941 ± 0.010	0.455	0.905 ± 0.024	0.211	0.938 ± 0.021
	High	0.273	0.942 ± 0.008	0.490	0.902 ± 0.017	0.201	0.947 ± 0.012
WebFace	**Original**	**0.390**	**0.986 ± 0.009**	**0.607**	**0.875 ± 0.028**	**0.756**	**0.905 ± 0.023**
	Low	0.287	0.960 ± 0.008	0.364	0.940 ± 0.000	0.200	0.944 ± 0.000
	Middle	0.242	0.960 ± 0.018	0.370	0.920 ± 0.005	0.202	0.939 ± 0.000
	High	0.235	0.938 ± 0.010	0.353	0.911 ± 0.000	0.196	0.924 ± 0.015
VGGFace2	**Original**	**0.575**	**0.996 ± 0.002**	**0.868**	**0.866 ± 0.016**	**0.943**	**0.906 ± 0.014**
	Low	0.330	0.981 ± 0.004	0.433	0.929 ± 0.000	0.215	0.932 ± 0.011
	Middle	0.250	0.990 ± 0.007	0.425	0.877 ± 0.024	0.214	0.913 ± 0.019
	High	0.258	0.982 ± 0.005	0.405	0.885 ± 0.024	0.215	0.909 ± 0.012
CelebA	**Original**	**0.435**	**0.901 ± 0.024**	**0.812**	**0.713 ± 0.042**	**0.867**	**0.828 ± 0.000**
	Low	0.333	0.804 ± 0.020	0.355	0.727 ± 0.037	0.211	0.849 ± 0.000
	Middle	0.325	0.795 ± 0.028	0.430	0.751 ± 0.020	0.205	0.870 ± 0.000
	High	0.242	0.789 ± 0.011	0.390	0.623 ± 0.000	0.208	0.891 ± 0.000

used to perturb the target model's outputs. Higher values mean a stronger perturbation degree and the y-axis represents the auditing performance.

We observe Face-Auditor **is robust to output perturbation**. Concretely, the auditing performance on SiameseNet and ProtoNet does not drop significantly, and the auditing performance drop on RelationNet is in the scope of 15%. We also observe a slight drop in the target model performance when the noise perturbation level increases, which indicates the robustness of Face-Auditor.

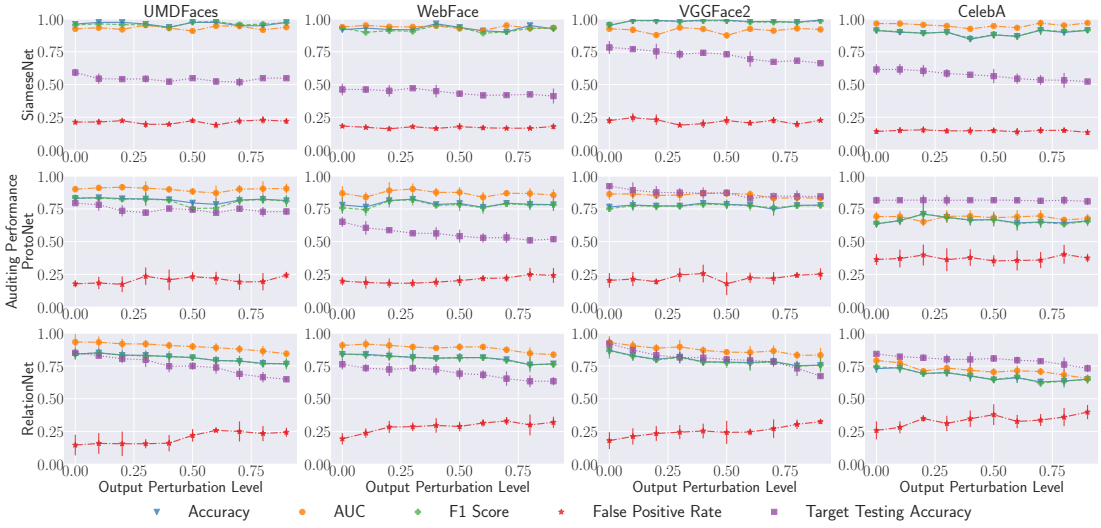


Figure 6.13: Auditing performance under output perturbations.

6.3.4 MemGuard

Threat Model. In practice, a malicious data collector might be aware of the existence of Face-Auditor. They modify their facial recognition models in a way to evade auditing and gain financial benefits or avoid a lawsuit. We evaluate the performance of Face-Auditor when the target model's output is perturbed to avoid membership auditing.

Methodology. We follow the design intuition of MemGuard [77] and perform adaptive attacks against Face-Auditor. The general idea is to perturb the similarity scores (outputs of the target model) while achieving two objectives: Minimum label loss and maximum auditing confusion. The first goal guarantees the noisy posteriors do not change the predicted labels of the target model given any inputs. The second goal aims to make Face-Auditor randomize its predictions of the user-level membership status given any face images to be audited. Concretely, to make Face-Auditor unable to distinguish member and non-member users, the adaptive attacker aims to add the maximum noise on the similarity scores under the constraint of not affecting the corresponding label. To ensure that the final summation of the target model's output is valid (summing to one), after adding the maximum noise to the target similarity score, we apply a SoftMax function to the entire similarity score vector, generating the final perturbed score vector. Note that the adversary cannot perturb the reference information as it is prepared by Face-Auditor and is a fixed value given any input images; thus, we concatenate the original reference information and the perturbed similarity scores as the auditing feature.

Results. Figure 6.14 illustrates the auditing performance of Face-Auditor under an adaptive attack. We observe that adaptive perturbation on the target model's outputs only slightly affects the auditing performance. The drop in auditing performance is less than 5%. This differs from the sample-level membership inference case, in which MemGuard leads to near-random guessing attack performance. There are three reasons. First, MemGuard can only perturb one value of the auditing feature per query, while Face-Auditor queries the target model multiple times and combine the similarity scores of multiple queries as the auditing feature. Second, in sample-level membership inference, an adaptive adversary can perturb the whole attack feature (the posterior of the target sample) simultaneously, but it can only perturb one value per query in our user-level membership inference setting. Third, the reference information helps maintain the relative correlation of the query images and captures the subtle difference between member and non-member users. Additionally, our experimental results (in Section 6.3.3) show a limited impact of output perturbation even without the minimum label loss

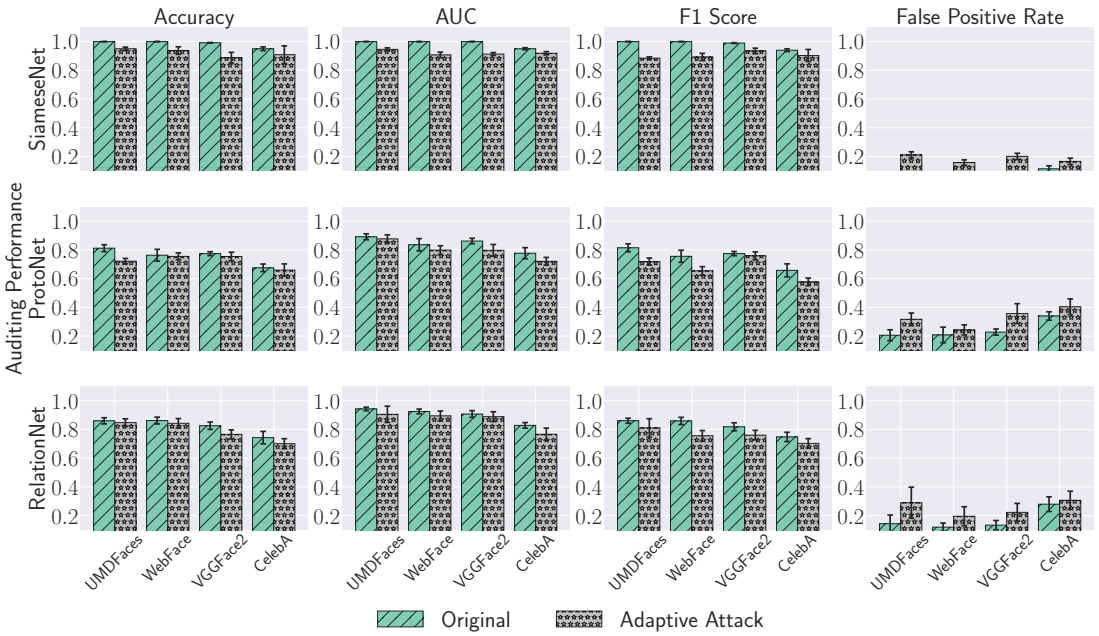


Figure 6.14: Auditing performance comparison under an adaptive adversary against Face-Auditor.

constraint. In summary, the similarity scores are more difficult for an adaptive adversary to perturb than output perturbation due to the minimum label loss constraint, which keeps the predicted label of a query sample fixed for a given support set and leaves the adversary little room to perturb.

6.4 Discussion

Practical Impacts of Face-Auditor. Face-Auditor can serve as a complementary tool for existing privacy-protective actions. Governments and regulators can use Face-Auditor as a tool for enforcing privacy regulations by determining if models are misusing data and violating individuals' privacy rights. Face-Auditor can be used by individuals as an auditing tool to detect potential misuse of face data. If a misuse happens, they can take legal actions to correct or withdraw their data (according to GDPR Articles 15, 16, 17, 18). Face-Auditor can also be employed by model developers to conduct self-inspection and ensure that their models are compliant with privacy regulations while demonstrating transparency in their data processing practices.

Potential Risks of Using Face-Auditor. While Face-Auditor has the potential to increase transparency for users contributing their data to train a model, it also poses a

threat to the intellectual property of the model provider. A malefactor could exploit Face-Auditor to launch user-level membership inference attacks against models with sensitive training data and use Face-Auditor as a stepping stone for other malicious activities, such as attribute inference attacks. Knowing that a user is in a sensitive facial recognition-based system's authorized zone could also allow an attacker to design adversarial examples to become an authorized user. On a facial recognition-based disease diagnosis system, a known member user might also expose to the privacy of having a particular disease. Although model developers can introduce protective mechanisms, our evaluation of the robustness of Face-Auditor in Section 6.3 demonstrates that its inference performance remains high even when subjected to four perturbations. In situations where private information is at risk of being inferred by malicious users, criminal laws such as the UK's Data Protection Act 2018 (sections 170 and 171) can deter such activities from occurring [11].

Extend to Other Data Domain. Our evaluation in this chapter mainly focuses on facial recognition models, so we use the term "user-level" instead of "class-level". We believe our method can be adapted to other objects as well, and the key challenge lies in choosing the appropriate reference information. For instance, when dealing with text data, a better reference might be the frequency of rare words rather than sentence-level or word-level similarity [151].

6.5 Conclusion

One privacy-sensitive application of the machine learning model is the facial recognition system, which determines whether a face image being verified belongs to an authorized user. However, the sensitivity of face images often faces high risk of being misused by malicious model trainers. To prevent the face images from being misused, one straightforward strategy is to modify the raw face images before uploading them to the Internet. However, these approaches inevitably destroy the semantic information of the face images and might be bypassed by a stronger attacker. Therefore, an auditing method that does not interfere with the facial recognition model's utility and cannot be quickly bypassed is urgently needed.

In this chapter, we proposed Face-Auditor to determine whether a target user's face images were used to train a few-shot-based facial recognition system relying on the user-level membership inference. We formulate the face auditing process as a user-level membership inference problem, and propose a complete toolkit Face-Auditor that can

carefully choose the probing set to query the few-shot-based facial recognition model and determine whether any of a user's face images is used in training the model. We further propose to use the similarity scores between the original face images as reference information to improve the auditing performance. Extensive experiments on multiple real-world face image datasets show that Face-Auditor can achieve auditing accuracy of up to 99%. We showed that Face-Auditor is robust when the users' face images or the target models are equipped with different perturbation mechanisms to the training images or the target models. In the end, we discuss the practical implications and potential risks of using Face-Auditor. Face-Auditor provides a new solution to prevent data misuse, especially in facial recognition systems.

7

Summary and Conclusion

7.1 Summary

Implementing the requirements of the privacy regulations in the machine learning systems is complicated and involves many tasks, such as deleting the storage of many copies, updating or regenerating the commercial models, and providing sufficient proof-of-privacy-protection to avoid fining or redesigning the whole system. However, machine learning's powerful memorization ability makes every step challenging.

This dissertation first understands the privacy in machine learning models through a technical implementation of the requirement of the privacy regulation, then launches two privacy attacks against machine unlearning and graph embedding sharing to challenge preconceived notions of privacy protection in machine learning models. We found unintended privacy leakage could happen counterfactually or under a stronger adversary scenario. To avoid our discovered privacy risks happening in the real world, we design mitigation methods for each of the located privacy attacks. Besides, we also design an auditing tool for unconsent data misuse detection, which protects users' privacy right.

Our first work [P1] starts with implementing "the right to be forgotten" in machine learning models and understanding the difficulties of balancing the model utility and unlearning efficiency. Due to the uniqueness of graph data, Graph Neural Networks are proposed to transform non-Euclidean graphs into low-dimensional vectors that are optimized with gradient descent algorithms. Deleting nodes/edges from a GNN model's training set can be challenging due to the entanglements of the nodes and edges. More importantly, breaking the training graph's structural information can cause severe model utility degradation. In Chapter 3, we propose an end-to-end framework called GraphEraser, which can achieve deterministic unlearning with high unlearning efficiency and low model utility degradation.

Our second work [P2] answers why implementing the right to be forgotten in machine learning models is more challenging than other data-oriented products like databases or search engines. In Chapter 4, we propose a membership inference attack against the "machine unlearning" setting where an adversary can access both the original and unlearned models. Our experimental results showed that machine unlearning was initially designed to lower the privacy risks of revoked data. On the contrary, the deleting process exposures more chances of being inferred the membership status. We investigated various unlearning scenarios and found that the different model behavior caused by machine unlearning cause an intrinsic privacy leakage in all the settings. To reduce practical risks, we assessed four defense mechanisms for our proposed attack and

found that only three of them were effective.

In the context of the machine learning paradigm, privacy risks can arise in data collection, model training, model deployment, or model updating. Our third work [P3] in Chapter 5 examines a common practice when sharing graph embedding with third parties that could severely leak information about the training graph. We launch three attacks against the graph embedding and successfully infer the properties of its original graph, whether a subgraph structure is in its original graph or even reconstructs the original graph. Our three attacks break the ingrained perception that "sharing graph embedding is more secure than sharing raw graph". Through extensive experiments, we understand the unintended privacy leakage in the model deployment phase.

Our fourth work [P4] utilizes privacy attacks against ML models for good. In Chapter 6, we design Face-Auditor based on the notion of user-level membership inference and provide practical guidelines for using it in facial recognition systems. In practice, some malicious companies might misuse users' data without their consent. In such cases, an auditing tool enables users to check whether a model is using/misusing their data is essential. Once misuse is detected, users could resort to "the right to restrict processing" to stop the unlawful commercial data process.

7.2 Future Research Directions

Machine learning models are known to be vulnerable to various attacks, such as membership inference, property inference, attribute inference, model inversion, data reconstruction, model stealing, backdoor attacks, etc. To build a trustworthy AI system, we need to understand the vulnerabilities of the pipeline, assess the privacy leakage risks, and design mitigation methods to avoid potential attacks from the real world. The privacy regulations provide a way to guide the improvement of privacy protection, yet the implementation details in machine learning systems still need more work. We summarize some future work that was raised during the writing of this dissertation into three themes, understanding, assessment, and mitigation.

First, the technical implementation of privacy-related regulations (e.g., GDPR) for machine learning (ML) models needs more *understanding*. How to design/redesign the ML paradigm that complies with the privacy regulations is still an open question and needs cooperation from the ML community and law conducts. Model providers need to respond to the enactment of privacy regulations actively, otherwise, they might meet a huge fine. But efficiently satisfying the requirement of privacy regulations only to find

another cropping up, as we discovered in Chapter 4 (While originally designed to protect the privacy of the data owner, machine unlearning may leave some imprint of the data in the ML model and thus create unintended privacy risks.) Sharding the dataset can be an effective and general way to reduce privacy risks and achieves a high model utility. Our design in Chapter 3 can strike a balance between unlearning efficiency and model utility, and we also showed low privacy leakage through a membership inference attack. But it needs to redesign the whole ML pipeline and is difficult to achieve in the model deployment phase. This raises the second research direction of achieving the unlearning goal without changing the ML pipeline and cost the minimum economic loss.

Second, empirically *assessing* provides a good way to gain an intuitive sense of privacy leakage in different ML models, as some existing paradigms can provide a false sense of security or privacy. We found a counterintuitive privacy leakage of sharing graph embedding in Chapter 5 and unintended privacy degradation caused by machine unlearning in Chapter 4. However, given an ML system, assessing the privacy leakage needs multi-dimensional attacks against the system. Launching one attack on a specific condition is not enough to gain an overall evaluation of the privacy risks. The attack methods might differ due to different threat models and the quality of the auxiliary information an adversary could access. Thus, designing a systematic privacy evaluation tool or designing a once-for-all privacy probing mechanism might be important for understanding the practical risks.

Third, designing multiple mechanisms to *mitigate* the vulnerabilities of different ML models needs more work. In Chapter 4, we evaluate four mitigation methods, i.e., publishing label, publishing top-k posteriors, scaling the temperature score to strike a balance between model confidence and privacy leakage, and adding differential privacy bounded Gaussian noise to the model's outputs. Three of these mitigation methods can survive our proposed attacker. But it is not clear if an advanced adversary who utilizes our known intuition can design a more advanced attack to bypass these defenses. In Chapter 6 we design Face-Auditor to detect unconsent data misuse. However, our false positive rate is not always to be 0. Thus one important future work includes further reducing the false positive rate of Face-Auditor, adapting Face-Auditor to other potentially sensitive domains, and designing Face-Auditor under more severe attack scenarios, for instance, when a user-level defense mechanism is equipped to the malicious model trainer to evade the auditing.

A

Appendix

A.1 Hyperparameter Settings of Simple Models

We use multiple ML models to evaluate the unintended privacy leakage in machine unlearning systems. All models are implemented by sklearn version 0.22 except for the logistic regression classifier. For reproduction purposes, we list their hyperparameter settings as follows:

- **Logistic Regression.** We implement a single linear logistic regression classifier with PyTorch. Training with Adam optimizer for 100 epochs.
- **Decision Tree.** We use the Gini index as the criterion, set parameter max_leaf_nodes as 10, and set other hyperparameters as default.
- **Random Forest.** We use the Gini index as the criterion, use 100 estimators, set min_samples_leaf=30, and set other hyperparameters as default.
- **Multi-layer Perceptron.** For the multi-layer-perceptron classifier, we use the Adam optimizer and Relu activation function. And set the hidden layer size and learning rate to 128 and 0.001, respectively.

A.1.1 Implementation of SimpleCNN

The architecture of our SimpleCNN is illustrated in Table A.1. For the MNIST dataset, input_channel $C_i = 1$, image width W, and height H are both 28. The kernel_size of convolution layer K_c and Max-pooling layer K_m are 3 and 2, respectively. We train the SimpeCNN for 100 epochs and use an SGD optimizer with a learning rate of 0.001.

Table A.1: SimpleCNN structure and hyperparameter.

Layer	**Hyperparameters**
Conv2D_1	(C_i, 32, K_c=3, 1)
Relu	-
Conv2D_2	(32, H, K_c, 1)
Maxpolling2D	K_m=2
Dropout_1	(0.25)
Flatten	1
Linear_1	($H \times (W/2 - K + 1) \times (H/2 - K + 1)$, 128)
Relu	-
Dropout_2	0.5
Linear_2	(128, #classes)
Softmax	dim=1

Bibliography

Author's Papers for this Thesis

[P1] Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. Graph Unlearning. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2022, 499–513.

[P2] Chen, M., Zhang, Z., Wang, T., Backes, M., Humbert, M., and Zhang, Y. When Machine Unlearning Jeopardizes Privacy. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021, 896–911.

[P3] Zhang, Z., Chen, M., Backes, M., Shen, Y., and Zhang, Y. Inference Attacks Against Graph Neural Networks. In: *USENIX Security Symposium (USENIX Security)*. 2022.

[P4] Chen, M., Zhang, Z., Wang, T., Backes, M., and Zhang, Y. FACE-AUDITOR: Data Auditing in Facial Recognition Systems. In: *USENIX Security Symposium (USENIX Security)*. 2023.

Other references

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

[2] https://oag.ca.gov/privacy/ccpa.

[3] https://laws-lois.justice.gc.ca/ENG/ACTS/P-8.6/index.html.

[4] https://iapp.org/media/pdf/resource_center/Brazilian_General_Data_Protection_Law.pdf.

[5] https://personalinformationprotectionlaw.com/.

[6] https://equalais.media.mit.edu/.

[7] https://archive.ics.uci.edu/ml/datasets/adult.

[8] https://www.kaggle.com/sobhanmoosavi/us-accidents.

[9] http://yann.lecun.com/exdb/mnist/.

[10] https://www.cs.toronto.edu/~kriz/cifar.html.

[11] https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted.

[12] Abadi, M., Chu, A., Goodfellow, I., McMahan, B., Mironov, I., Talwar, K., and Zhang, L. Deep Learning with Differential Privacy. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2016, 308–318.

[13] Albert, K., Penney, J., Schneier, B., and Kumar, R. S. S. Politics of Adversarial Machine Learning. *CoRR abs/2002.05648* (2020).

[14] Andersen, R., Chung, F. R. K., and Lang, K. J. Local Graph Partitioning using PageRank Vectors. In: *Annual Symposium on Foundations of Computer Science (FOCS)*. 2006, 475–486.

[15] Atwood, J. and Towsley, D. Diffusion-Convolutional Neural Networks. In: *Annual Conference on Neural Information Processing Systems (NIPS)*. 2016, 1993–2001.

[16] Avdiukhin, D., Pupyrev, S., and Yaroslavtsev, G. Multi-Dimensional Balanced Graph Partitioning via Projected Gradient Descent. *Proceedings of the VLDB Endowment* (2019).

[17] Awadelkarim, A. and Ugander, J. Prioritized Restreaming Algorithms for Balanced Graph Partitioning. In: *ACM Conference on Knowledge Discovery and Data Mining (KDD)*. 2020, 1877–1887.

[18] Backes, M., Humbert, M., Pang, J., and Zhang, Y. walk2friends: Inferring Social Links from Mobility Profiles. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2017, 1943–1957.

[19] Bansal, A., Nanduri, A., Castillo, C. D., Ranjan, R., and Chellappa, R. UMDFaces: An Annotated Face Dataset for Training Deep Networks. In: *International Joint Conference on Biometrics (IJCB)*. 2017, 464–473.

[20] Baumhauer, T., Schöttle, P., and Zeppelzauer, M. Machine Unlearning: Linear Filtration for Logit-based Classifier. *CoRR abs/2002.02730* (2020).

[21] Béguelin, S. Z., Wutschitz, L., Tople, S., Rühle, V., Paverd, A., Ohrimenko, O., Köpf, B., and Brockschmidt, M. Analyzing Information Leakage of Updates to Natural Language Models. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020, 363–375.

[22] Bertram, T., Bursztein, E., Caro, S., Chao, H., Chin, R., Fleischer, F., Gustafsson, A., Hemerly, J., Hibbert, C., InvernizziLanah, L., Donnelly, K., Ketover, J., Laefer, J., Nicholas, P., Niu, Y., Obhi, H., Price, D., Strait, A., Thomas, K., and Verney, A. Five Years of the Right to be Forgotten. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2019, 959–972.

[23] Bianchi, F. M., Grattarola, D., and Alippi, C. Spectral Clustering with Graph Neural Networks for Graph Pooling. In: *International Conference on Machine Learning (ICML)*. 2020, 874–883.

[24] Biswas, S., Dong, Y., Kamath, G., and Ullman, J. R. CoinPress: Practical Private Mean and Covariance Estimation. In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 2020.

[25] Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C., Jia, H., Travers, A., Zhang, B., Lie, D., and Papernot, N. Machine Unlearning. In: *IEEE Symposium on Security and Privacy (S&P)*. 2021.

[26] Brophy, J. and Lowd, D. Machine Unlearning for Random Forests. In: *International Conference on Machine Learning (ICML)*. 2021, 1092–1104.

[27] Cao, Q., Shen, L., Xie, W., Parkhi, O. M., and Zisserman, A. VGGFace2: A Dataset for Recognising Faces across Pose and Age. In: *International Conference on Automatic Face & Gesture Recognition (FG)*. 2018, 67–74.

[28] Cao, Y. and Yang, J. Towards Making Systems Forget with Machine Unlearning. In: *IEEE Symposium on Security and Privacy (S&P)*. 2015, 463–480.

[29] Cao, Y., Yu, A. F., Aday, A., Stahl, E., Merwine, J., and Yang, J. Efficient Repair of Polluted Machine Learning Systems via Causal Unlearning. In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2018, 735–747.

[30] Carlini, N., Hayes, J., Nasr, M., Jagielski, M., Sehwag, V., Tramèr, F., Balle, B., Ippolito, D., and Wallace, E. Extracting Training Data from Diffusion Models. *CoRR abs/2301.13188* (2023).

[31] Chandrasekaran, V., Gao, C., Tang, B., Fawaz, K., Jha, S., and Banerjee, S. Face-Off: Adversarial Face Obfuscation. *Privacy Enhancing Technologies Symposium* (2021).

[32] Chen, D., Yu, N., Zhang, Y., and Fritz, M. GAN-Leaks: A Taxonomy of Membership Inference Attacks against Generative Models. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020, 343–362.

[33] Chen, W., Liu, Y., Kira, Z., Wang, Y. F., and Huang, J. A Closer Look at Few-shot Classification. In: *International Conference on Learning Representations (ICLR)*. 2019.

[34] Chen, Z., Villar, S., Chen, L., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with GNNs. In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 2019, 15868–15876.

[35] Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In: *ACM Conference on Knowledge Discovery and Data Mining (KDD)*. 2019, 257–266.

[36] Coates, A., Ng, A. Y., and Lee, H. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011, 215–223.

[37] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In: *Annual Conference on Neural Information Processing Systems (NIPS)*. 2016, 3837–3845.

[38] Delling, D., Goldberg, A. V., Razenshteyn, I. P., and Werneck, R. F. F. Graph Partitioning with Natural Cuts. In: *International Symposium on Parallel and Distributed Processing (IPDPS)*. 2011, 1135–1146.

[39] Diao, Z., Wang, X., Zhang, D., Liu, Y., Xie, K., and He, S. Dynamic Spatial-Temporal Graph Convolutional Neural Networks for Traffic Forecasting. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2019, 890–897.

[40] Duddu, V., Boutet, A., and Shejwalkar, V. Quantifying Privacy Leakage in Graph Embedding. *CoRR abs/2010.00906* (2020).

[41] Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking Graph Neural Networks. *CoRR abs/2003.00982* (2020).

[42] Dwork, C. and Roth, A. *The Algorithmic Foundations of Differential Privacy*. Now Publishers Inc., 2014.

[43] Ellers, M., Cochez, M., Schumacher, T., Strohmaier, M., and Lemmerich, F. Privacy Attacks on Network Embeddings. *CoRR abs/1912.10979* (2019).

[44] Errica, F., Podda, M., Bacciu, D., and Micheli, A. A Fair Comparison of Graph Neural Networks for Graph Classification. In: *International Conference on Learning Representations (ICLR)*. 2020.

[45] Evtimov, I., Sturmfels, P., and Kohno, T. FoggySight: A Scheme for Facial Lookup Privacy. *Privacy Enhancing Technologies Symposium* (2021).

[46] Faraki, M., Yu, X., Tsai, Y., Suh, Y., and Chandraker, M. Cross-Domain Similarity Learning for Face Recognition in Unseen Domains. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, 15292–15301.

[47] Fredrikson, M., Jha, S., and Ristenpart, T. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2015, 1322–1333.

[48] Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., and Ristenpart, T. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. In: *USENIX Security Symposium (USENIX Security)*. 2014, 17–32.

[49] Ganju, K., Wang, Q., Yang, W., Gunter, C. A., and Borisov, N. Property Inference Attacks on Fully Connected Neural Networks using Permutation Invariant Representations. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2018, 619–633.

[50] Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[51] Ginart, A. A., Guan, M. Y., Valiant, G., and Zou, J. Making AI Forget You: Data Deletion in Machine Learning. In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 2019, 3513–3526.

[52] Girvan, M. and Newman, M. E. J. Community Structure in Social and Biological Networks. *Proceedings of the National Academy of Sciences* (2002).

[53] Golatkar, A., Achille, A., and Soatto, S. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, 9301–9309.

[54] Gomrokchi, M., Amin, S., Aboutalebi, H., Wong, A., and Precup, D. Membership Inference Attacks Against Temporally Correlated Data in Deep Reinforcement Learning. *CoRR abs/2109.03975* (2021).

[55] Good, B. H., Montjoye, Y.-A. D., and Clauset, A. Performance of Modularity Maximization in Practical Contexts. *Physical Review E* (2010).

[56] Gregory, S. Finding Overlapping Communities in Networks by Label Propagation. *New Journal of Physics* (2010).

[57] Grover, A. and Leskovec, J. node2vec: Scalable Feature Learning for Networks. In: *ACM Conference on Knowledge Discovery and Data Mining (KDD)*. 2016, 855–864.

[58] Gu, T., Dolan-Gavitt, B., and Grag, S. Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *CoRR abs/1708.06733* (2017).

[59] Guo, C., Goldstein, T., Hannun, A. Y., and Maaten, L. van der. Certified Data Removal from Machine Learning Models. In: *International Conference on Machine Learning (ICML)*. 2020, 3832–3842.

[60] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On Calibration of Modern Neural Networks. In: *International Conference on Machine Learning (ICML)*. 2017.

[61] Gupta, V., Jung, C., Neel, S., Roth, A., Sharifi-Malvajerdi, S., and Waites, C. Adaptive Machine Unlearning. In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 2021.

[62] Gurovich, Y., Hanani, Y., Bar, O., Nadav, G., Fleischer, N., Gelbman, D., Basel-Salmon, L., Krawitz, P. M., Kamphausen, S. B., Zenker, M., Bird, L. M., and Gripp, K. W. Identifying Facial Phenotypes of Genetic Disorders Using Deep Learning. *Nature Medicine* (2019).

[63] Hamilton, W. L. *Graph Representation Learning*. Morgan and Claypool, 2020.

[64] Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive Representation Learning on Large Graphs. In: *Annual Conference on Neural Information Processing Systems (NIPS)*. 2017, 1025–1035.

[65] Hayes, J., Melis, L., Danezis, G., and Cristofaro, E. D. LOGAN: Evaluating Privacy Leakage of Generative Models Using Generative Adversarial Networks. *Privacy Enhancing Technologies Symposium* (2019).

[66] He, K., Zhang, X., Ren, S., and Sun, J. Deep Residual Learning for Image Recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 770–778.

[67] He, S., Bastani, F., Jagwani, S., Park, E., Abbar, S., Alizadeh, M., Balakrishnan, H., Chawla, S., Madden, S., and Sadeghi, M. A. RoadTagger: Robust Road Attribute Inference with Graph Neural Networks. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2020, 10965–10972.

[68] He, X. and Zhang, Y. Quantifying and Mitigating Privacy Risks of Contrastive Learning. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021, 845–863.

[69] He, Y., Rahimian, S., Schiele1, B., and Fritz, M. Segmentations-Leak: Membership Inference Attacks and Defenses in Semantic Image Segmentation. In: *European Conference on Computer Vision (ECCV)*. 2020, 519–535.

[70] Hu, H., Salcic, Z., Dobbie, G., Chen, J., Sun, L., and Zhang, X. Membership Inference via Backdooring. In: *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2022, 3832–3838.

[71] Huang, G., Liu, Z., Maaten, L. van der, and Weinberger, K. Q. Densely Connected Convolutional Networks. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, 2261–2269.

[72] Huang, S., Li, Y., Bao, Z., and Li, Z. Towards Efficient Motif-based Graph Partitioning: An Adaptive Sampling Approach. In: *International Conference on Data Engineering (ICDE)*. 2021, 528–539.

[73] Izzo, Z., Smart, M. A., Chaudhuri, K., and Zou, J. Approximate Data Deletion from Machine Learning Models: Algorithms and Evaluations. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2021, 2008–2016.

[74] Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., and Papernot, N. High Accuracy and High Fidelity Extraction of Neural Networks. In: *USENIX Security Symposium (USENIX Security)*. 2020, 1345–1362.

[75] Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., and Li, B. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. 2018, 19–35.

[76] Jayaraman, B. and Evans, D. Evaluating Differentially Private Machine Learning in Practice. In: *USENIX Security Symposium (USENIX Security)*. 2019, 1895–1912.

[77] Jia, J., Salem, A., Backes, M., Zhang, Y., and Gong, N. Z. MemGuard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2019, 259–274.

[78] Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In: *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2017, 615–629.

[79] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2002).

[80] Karypis, G. and Kumar, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* (1998).

[81] Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. Molecular Graph Convolutions: Moving Beyond Fingerprints. *Journal of Computer-Aided Molecular Design* (2016).

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

[83] Koch, G., Zemel, R., and Salakhutdinov, R. Siamese Neural Networks for One-Shot Image Recognition. In: *ICML Workshop on Deep Learning (DL)*. 2015.

[84] Komkov, S. and Petiushko, A. AdvHat: Real-World Adversarial Attack on ArcFace Face ID System. In: *International Conference on Pattern Recognition (ICPR)*. 2020, 819–826.

[85] Kulynych, B., Overdorf, R., Troncoso, C., and Gùrses, S. F. POTs: Protective Optimization Technologies. In: *Conference on Fairness, Accountability, and Transparency (FAT)*. 2020, 177–188.

[86] Lane, N. D. and Georgiev, P. Can Deep Learning Revolutionize Mobile Sensing? In: *International Workshop on Mobile Computing Systems and Applications (HotMobile)*. 2015, 117–122.

[87] LaSalle, D. and Karypis, G. A Parallel Hill-Climbing Refinement Algorithm for Graph Partitioning. In: *International Conference on Parallel Processing (ICCP)*. 2016, 236–241.

[88] Li, J., Li, N., and Ribeiro, B. Membership Inference Attacks and Defenses in Classification Models. In: *ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2021, 5–16.

[89] Li, N., Lyu, M., Su, D., and Yang, W. *Differential Privacy: From Theory to Practice*. Morgan & Claypool Publishers, 2016.

[90] Li, T. and Lin, L. AnonymousNet: Natural Face De-Identification With Measurable Privacy. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, 56–65.

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

[92] Li, Z. and Zhang, Y. Membership Leakage in Label-Only Exposures. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021, 880–895.

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

[94] Liu, H., Jia, J., Qu, W., and Gong, N. Z. EncoderMI: Membership Inference against Pre-trained Encoders in Contrastive Learning. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021.

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

[96] Liu, Y., Wen, R., He, X., Salem, A., Zhang, Z., Backes, M., Cristofaro, E. D., Fritz, M., and Zhang, Y. ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models. In: *USENIX Security Symposium (USENIX Security)*. 2022.

[97] Liu, Z., Luo, P., Wang, X., and Tang, X. Deep Learning Face Attributes in the Wild. In: *IEEE International Conference on Computer Vision (ICCV)*. 2015, 3730–3738.

[98] Long, Y., Bindschaedler, V., Wang, L., Bu, D., Wang, X., Tang, H., Gunter, C. A., and Chen, K. Understanding Membership Inferences on Well-Generalized Learning Models. *CoRR abs/1802.04889* (2018).

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

[100] Malinen, M. I. and Fränti, P. Balanced K-Means for Clustering. *Structural, Syntactic, and Statistical Pattern Recognition* (2014).

[101] Malmi, E. and Weber, I. You Are What Apps You Use: Demographic Prediction Based on User's Apps. In: *International Conference on Weblogs and Social Media (ICWSM)*. 2016, 635–638.

[102] McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. Learning Differentially Private Recurrent Language Models. In: *International Conference on Learning Representations (ICLR)*. 2018.

[103] McSherry, F. Spectral Partitioning of Random Graphs. In: *Annual Symposium on Foundations of Computer Science (FOCS)*. 2001, 529–537.

[104] Melis, L., Song, C., Cristofaro, E. D., and Shmatikov, V. Exploiting Unintended Feature Leakage in Collaborative Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. 2019, 497–512.

[105] Miao, Y., Xue, M., Chen, C., Pan, L., Zhang, J., Zhao, B. Z. H., Kaafar, D., and Xiang, Y. The Audio Auditor: User-Level Membership Inference in Internet of Things Voice Services. *Privacy Enhancing Technologies Symposium* (2021).

[106] Moraes, T. G., Almeida, E. C., and Pereira, J. R. L. de. Smile, You are being Identified! Risks and Measures for the Use of Facial Recognition in (Semi-)public Spaces. *AI Ethics* (2021).

[107] Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUDataset: A collection of benchmark datasets for learning with graphs. In: *The ICML Workshop on Graph Representation Learning and Beyond (GRL)*. 2020.

[108] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2019, 4602–4609.

[109] Nasr, M., Shokri, R., and Houmansadr, A. Machine Learning with Membership Privacy using Adversarial Regularization. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2018, 634–646.

[110] Nasr, M., Shokri, R., and Houmansadr, A. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. 2019, 1021–1035.

[111] Nasr, M., Song, S., Thakurta, A., Papernot, N., and Carlini, N. Adversary Instantiation: Lower Bounds for Differentially Private Machine Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. 2021.

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

[113] Newman, M. E. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Sciences* (2006).

[114] Nogueira, R. F., Alencar Lotufo, R. de, and Machado, R. C. Fingerprint Liveness Detection Using Convolutional Neural Networks. *IEEE Transactions on Information Forensics and Security* (2016).

[115] Oh, S. J., Augustin, M., Schiele, B., and Fritz, M. Towards Reverse-Engineering Black-Box Neural Networks. In: *International Conference on Learning Representations (ICLR)*. 2018.

[116] Orekondy, T., Schiele, B., and Fritz, M. Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks. In: *International Conference on Learning Representations (ICLR)*. 2020.

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

[118] Pan, X., Wang, W., Zhang, X., Li, B., Yi, J., and Song, D. How You Act Tells a Lot: Privacy-Leaking Attack on Deep Reinforcement Learning. In: *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*. 2019, 368–379.

[119] Papernot, N., McDaniel, P. D., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical Black-Box Attacks Against Machine Learning. In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2017, 506–519.

[120] Papernot, N., McDaniel, P. D., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The Limitations of Deep Learning in Adversarial Settings. In: *IEEE European Symposium on Security and Privacy (Euro S&P)*. 2016, 372–387.

[121] Papernot, N., McDaniel, P., Sinha, A., and Wellman, M. SoK: Towards the Science of Security and Privacy in Machine Learning. In: *IEEE European Symposium on Security and Privacy (Euro S&P)*. 2018, 399–414.

[122] Papernot, N., Song, S., Mironov, I., Raghunathan, A., Talwar, K., and Erlingsson, Ú. Scalable Private Learning with PATE. In: *International Conference on Learning Representations (ICLR)*. 2018.

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

[124] Pyrgelis, A., Troncoso, C., and Cristofaro, E. D. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. In: *Network and Distributed System Security Symposium (NDSS)*. 2018.

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

[126] Radiya-Dixit, E., Hong, S., Carlini, N., and Tramér, F. Data Poisoning Won't Save You From Facial Recognition. In: *ICML Workshop on Adversarial Machine Learning (AdvML)*. 2022.

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

[128] Rahimian, S., Orekondy, T., and Fritz, M. Differential Privacy Defenses and Sampling Attacks for Membership Inference. In: *PriML Workshop (PriML)*. 2020.

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

[130] Saha, A., Subramanya, A., and Pirsiavash, H. Hidden Trigger Backdoor Attacks. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2020, 11957–11965.

[131] Salem, A., Bhattacharya, A., Backes, M., Fritz, M., and Zhang, Y. Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning. In: *USENIX Security Symposium (USENIX Security)*. 2020, 1291–1308.

[132] Salem, A., Zhang, Y., Humbert, M., Berrang, P., Fritz, M., and Backes, M. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In: *Network and Distributed System Security Symposium (NDSS)*. 2019.

[133] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, 4510–4520.

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

[135] Schroff, F., Kalenichenko, D., and Philbin, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, 815–823.

[136] Shafahi, A., Huang, W. R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., and Goldstein, T. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 2018, 6103–6113.

[137] Shan, S., Wenger, E., Zhang, J., Li, H., Zheng, H., and Zhao, B. Y. Fawkes: Protecting Privacy against Unauthorized Deep Learning Models. In: *USENIX Security Symposium (USENIX Security)*. 2020, 1589–1604.

[138] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2016, 1528–1540.

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

[140] Shen, Y., He, X., Han, Y., and Zhang, Y. Model Stealing Attacks Against Inductive Graph Neural Networks. In: *IEEE Symposium on Security and Privacy (S&P)*. 2022.

[141] Shervashidze, N., Schweitzer, P., Leeuwen, E. J. van, Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* (2011).

[142] Shi, J. and Malik, J. Normalized Cuts and Image Segmentation. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1997, 731–737.

[143] Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In: *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2021, 1548–1554.

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

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

[146] Simonovsky, M. and Komodakis, N. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. In: *International Conference on Artificial Neural Networks (ICANN)*. 2018, 412–422.

[147] Snell, J., Swersky, K., and Zemel, R. S. Prototypical Networks for Few-shot Learning. In: *Annual Conference on Neural Information Processing Systems (NIPS)*. 2017, 4077–4087.

[148] Sommer, D. M., Song, L., Wagh, S., and Mittal, P. Towards Probabilistic Verification of Machine Unlearning. *CoRR abs/2003.04247* (2020).

[149] Song, C. and Raghunathan, A. Information Leakage in Embedding Models. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020, 377–390.

[150] Song, C., Ristenpart, T., and Shmatikov, V. Machine Learning Models that Remember Too Much. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2017, 587–601.

[151] Song, C. and Shmatikov, V. Auditing Data Provenance in Text-Generation Models. In: *ACM Conference on Knowledge Discovery and Data Mining (KDD)*. 2019, 196–206.

[152] Song, C. and Shokri, R. Membership Encoding for Deep Learning. In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2020, 344–356.

[153] Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H. S., and Hospedales, T. M. Learning to Compare: Relation Network for Few-Shot Learning. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, 1199–1208.

[154] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going Deeper with Convolutions. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, 1–9.

[155] Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, 1701–1708.

[156] Tan, M., Zhou, Z., and Li, Z. The Many-faced God: Attacking Face Verification System with Embedding and Image Recovery. In: *Annual Computer Security Applications Conference (ACSAC)*. 2021, 17–30.

[157] Tang, X., Mahloujifar, S., Song, L., Shejwalkar, V., Nasr, M., Houmansadr, A., and Mittal, P. Mitigating Membership Inference Attacks by Self-Distillation Through a Novel Ensemble Architecture. In: *USENIX Security Symposium (USENIX Security)*. 2022, 1433–1450.

[158] Thys, S., Ranst, W. V., and Goedemé, T. Fooling Automated Surveillance Cameras: Adversarial Patches to Attack Person Detection. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, 49–55.

[159] Tian, Y., Wang, Y., Krishnan, D., Tenenbaum, J. B., and Isola, P. Rethinking Few-Shot Image Classification: A Good Embedding is All You Need? In: *European Conference on Computer Vision (ECCV)*. 2020, 266–282.

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

[161] Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. Ensemble Adversarial Training: Attacks and Defenses. In: *International Conference on Learning Representations (ICLR)*. 2017.

[162] Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. Stealing Machine Learning Models via Prediction APIs. In: *USENIX Security Symposium (USENIX Security)*. 2016, 601–618.

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

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

[165] Wang, B. and Gong, N. Z. Stealing Hyperparameters in Machine Learning. In: *IEEE Symposium on Security and Privacy (S&P)*. 2018, 36–52.

[166] Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., and Zhao, B. Y. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In: *IEEE Symposium on Security and Privacy (S&P)*. 2019, 707–723.

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

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

[169] Wang, M. and Deng, W. Deep Face Recognition: A Survey. *Neurocomputing* (2021).

[170] Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. Generalizing from a Few Examples: A Survey on Few-shot Learning. *ACM Computing Surveys* (2020).

[171] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image Quality Assessment: from Error Visibility to Structural Similarity. *IEEE Transactions on Image Process* (2004).

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

[173] Wenger, E., Shan, S., Zheng, H., and Zhao, B. Y. SoK: Anti-Facial Recognition Technology. In: *IEEE Symposium on Security and Privacy (S&P)*. 2022.

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

[175] Wu, Y., Dobriban, E., and Davidson, S. B. DeltaGrad: Rapid retraining of machine learning models. In: *International Conference on Machine Learning (ICML)*. 2020, 10355–10366.

[176] Wu, Z., Lim, S., Davis, L. S., and Goldstein, T. Making an Invisibility Cloak: Real World Adversarial Attacks on Object Detectors. In: *European Conference on Computer Vision (ECCV)*. 2020, 1–17.

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

[178] Xue, M., Magno, G., Cunha, E., Almeida, V., and Ross, K. W. The Right to be Forgotten in the Media: A Data-Driven Study. *Privacy Enhancing Technologies Symposium* (2016).

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

[180] Yang, S., Luo, P., Loy, C. C., and Tang, X. WIDER FACE: A Face Detection Benchmark. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 5525–5533.

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

[182] Yeom, S., Giacomelli, I., Fredrikson, M., and Jha, S. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In: *IEEE Computer Security Foundations Symposium (CSF)*. 2018, 268–282.

[183] Yi, D., Lei, Z., Liao, S., and Li, S. Z. Learning Face Representation from Scratch. *CoRR abs/1411.7923* (2014).

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

[185] Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical Graph Representation Learning with Differentiable Pooling. In: *Annual Conference on Neural Information Processing Systems (NeurIPS)*. 2018, 4805–4815.

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

[187] Zhang, M., Ren, Z., Wang, Z., Ren, P., Chen, Z., Hu, P., and Zhang, Y. Membership Inference Attacks Against Recommender Systems. In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021, 864–879.

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

[189] Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, 586–595.

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

[191] Zhang, Y., Humbert, M., Surma, B., Manoharan, P., Vreeken, J., and Backes, M. Towards Plausible Graph Anonymization. In: *Network and Distributed System Security Symposium (NDSS)*. 2020.

[192] Zhang, Z., Wang, T., Honorio, J., Li, N., Backes, M., He, S., Chen, J., and Zhang, Y. PrivSyn: Differentially Private Data Synthesis. In: *USENIX Security Symposium (USENIX Security)*. 2021, 929–946.

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

[194] Zhou, J., Chen, Y., Shen, C., and Zhang, Y. Property Inference Attacks Against GANs. In: *Network and Distributed System Security Symposium (NDSS)*. 2022.

[195] Zhu, D., Chen, D., Grossklags, J., and Fritz, M. Data Forensics in Diffusion Models: A Systematic Analysis of Membership Privacy. *CoRR abs/2302.07801* (2023).