Connecting the dots
Exploring backdoor attacks on graph neural networks

Xu, J.

Publication date
2024

Document Version
Submitted manuscript

Citation (APA)
Xu, J. (2024). *Connecting the dots: Exploring backdoor attacks on graph neural networks*. [Dissertation (TU Delft), Delft University of Technology]. ProefschriftMaken.

Connecting the Dots

Exploring Backdoor Attacks on Graph Neural Networks

CONNECTING THE DOTS

EXPLORING BACKDOOR ATTACKS ON GRAPH NEURAL NETWORKS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen
chair of the Board for Doctorates
to be defended publicly on
Monday 13 May 2024 at 10:00 o'clock

by

Jing XU

Master of Engineering in Optical Engineering, Beihang University, China
born in Zhejiang, China

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. R.L. Lagendijk	Delft University of Technology, promotor
Dr. F.A. Oliehoek	Delft University of Technology, promotor
Dr. S. Picek	Delft University of Technology, copromotor

Independent members:

Prof. dr. G. Smaragdakis	Delft University of Technology
Prof. dr. A. Sadeghi	Technical University of Darmstadt
Prof. dr. L. Cavallaro	University College London
Dr. E. Isufi	Delft University of Technology
Prof. dr. M.M. de Weerdt	Delft University of Technology, reserve member

This research was financed by the China Scholarship Council.

Keywords: Backdoor Attacks, Graph Neural Networks, Security

Printed by: ProefschriftMaken Printing

Front & Back: Jing Xu & Jinke He. Original image from istockphoto.com

ISBN 978-94-6469-918-0

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

In memory of my beloved grandparents.
致我亲爱的外公外婆

The true sign of intelligence is not knowledge but imagination.

Albert Einstein

Contents

SUMMARY

Deep Neural Networks (DNNs) have found extensive applications across diverse fields, such as image classification, speech recognition, and natural language processing. However, their susceptibility to various adversarial attacks, notably the backdoor attack, has repeatedly been demonstrated in recent years. The backdoor attack aims to misclassify inputs with specific trigger pattern(s) into the pre-determined label(s) by training the model on the poisoned dataset. Backdoor attacks on DNNs can lead to severe real-world consequences, e.g., a deep leaning-based classifier in a self-driving car can be backdoored to misclassify a stop sign as a speed limit sign.

With an increasing of real-world data being represented as graphs, Graph Neural Networks (GNNs), a subset of the DNNs, have demonstrated remarkable performance in processing graph data. Despite their efficiency, GNNs, similar to other DNNs, are also vulnerable to backdoor attacks, which can lead to severe results, especially when GNNs are applied in security-related scenarios. Although backdoor attacks have been extensively studied in the image domain, we still need dedicated efforts for the graph domain due to the difference between graph data and other data, e.g., images.

This thesis embarks on an exploration of backdoor attacks on GNNs. Chapter 3 focuses on designing and investigating backdoor attacks on centralized GNNs. Specifically, we explore the influence of trigger injecting position on the backdoor attack performance on GNNs. To explore this impact, we propose approaches based on explanation techniques on GNNs, which contributes to exploring the interaction between the explainability and robustness of GNNs. Furthermore, we design a clean-label backdoor attack on GNNs to make the poisoned inputs more challenging to be detected.

Considering the growing privacy concern, we focus on backdoor attacks on federated GNNs in Chapter 4. We propose a label-only membership inference attack on GNNs in the scenario that the attacker can only get label output from the GNN models. Moreover, we investigate centralized and distributed backdoor attacks on federated GNNs.

Besides designing efficient backdoor attacks on GNNs, we also explore the possibility of leveraging backdoor attacks for defensive purposes for GNNs. Chapter 5 introduces a watermarking framework for GNNs based on backdoor attacks. Our research outcomes will deepen the understanding of backdoor attacks on GNNs and push the GNN model designers to develop more secure models.

Samenvatting

Deep Neural Networks (DNNs) hebben hun toepassingen gevonden in diverse gebieden, zoals het classificeren van afbeeldingen, spraakherkenning, en het automatisch verwerken van taal. Echter zijn zij wel vatbaar voor aanvallen die de bruikbaarheid kunnen verminderen, in het bijzonder backdoor attacks.

Het doel van een backdoor attack is om misclassificaties te veroorzaken door een ‘trigger’ patroon toe te voegen aan voorbeelden uit een dataset zodat het model een specifieke voorspelling maakt. Backdoor attacks kunnen tot ernstige gevolgen leiden in de echte wereld, bijvoorbeeld wanneer een deep-learning model wordt gebruikt in een zelfrijdende auto en een aanvaller een stopbord kan laten misclassificeren als snelheidsbord.

Nu data uit de echte wereld steeds vaker wordt gerepresenteerd in de vorm van grafen zijn Graph Neural Networks (GNNs), een speciale vorm van een DNNs, bijzonder goed geworden in het verwerken van graafdata. Maar hoewel ze efficient zijn, zijn ze net als DNNs kwetsbaar voor backdoor attacks en dit kan tot ernstige gevolgen leiden wanneer GNNs worden toegepast in scenarios waar de veiligheid belangrijk is. Terwijl er al veel onderzoek is geweest naar backdoor attacks voor data in de vorm van afbeeldingen, is er nog veel onderzoek nodig voor data in de vorm van grafen vanwege de grote verschillen in hoe deze soorten data gerepresenteerd worden.

Dit proefschrift verkent het gebied van backdoor attacks voor GNNs. Hoofdstuk 3 focust op het ontwerpen en onderzoeken van backdoor attacks op gecentraliseerde GNNs.

Specifiek onderzoeken we het effect van de trigger positie op de backdoor attack prestaties van GNNs. Hiervoor stellen wij methodes voor die gebaseerd zijn op uitlegtechnieken voor GNNs, wat bijdraagt aan onderzoek naar de interactie tussen de uitlegbaarheid en robuustheid van GNNs. Verder ontwerpen wij ook een clean-label backdoor attack voor GNNs om het moeilijker te maken om veranderingen aan de dataset te detecteren.

Wegens de toenemende bezorgdheid over privacy leggen wij in Hoofdstuk 4 de focus op backdoor attacks tegen federated GNNs. We stellen een label-only membership inference attack voor tegen GNNs waarbij de aanvaller alleen toegang heeft tot de voorspellingen van het model. Verder onderzoeken we ook gecentraliseerde en gedistribueerde backdoor attacks tegen federated GNNs.

Naast het ontwerpen van efficiënte backdoor attacks tegen GNNs verkennen we ook de mogelijkheid om backdoor attacks te gebruiken als verdedigingsmiddel. Hoofdstuk 5 introduceert een framework om watermerken toe te voegen aan GNNs door middel van backdoor attacks.

Onze onderzoeksresultaten zullen ons begrip van backdoor attacks voor GNNs verbeteren and de ontwerpers van GNNs aanmoedigen om veiligere modellen te ontwikkelen.

综述

深度神经网络在各个领域都有广泛的应用，比如图像分类、语音识别和自然语言处理等。然而，已有研究证明深度神经网络易收到各种对抗性攻击的影响，尤其是后门攻击。后门攻击旨在通过在中毒数据集上训练模型，将具有特定触发模式的输入数据错误分类到预定目标标签中。对深度神经网络的后门攻击可导致严重的现实后果，例如，自动驾驶汽车中基于深度学习的分类器可能会被设置后门，将停车标志错误分类为限速标志，从而导致交通事故。

随着越来越多的现实世界数据以图的形式表示，图神经网络（深度神经网络的一种）在处理图数据方面表现出了卓越的性能。尽管性能优越，但与其他深度神经网络类似，图神经网络也容易受到后门攻击。尽管后门攻击在图像领域已经得到了广泛的研究，但由于图数据与其他数据（例如图像）之间的差异，我们仍然需要进一步深入研究图领域的后门攻击。

本论文旨在探索图神经网络中的后门攻击。在第三章中，我们重点介绍集中式图神经网络上后门攻击的设计和研究。具体来说，我们探讨了触发器注入位置对图神经网络后门攻击性能的影响。为了探索这种影响，我们提出了基于图神经网络模型解释技术的方法，这有助于探索图神经网络可解释性和鲁棒性之间的关系。此外，我们设计了一种针对图神经网络的干净标签后门攻击，使有毒输入数据更难被检测到。

考虑到越来越受模型用户关注的隐私问题，我们在第四章中重点研究了对联邦图神经网络的后门攻击。首先，在攻击者只能从图神经网络模型中获取标签输出的情况下，我们提出了对图神经网络的标签成员推理攻击。然后，我们研究了对联邦图神经网络的集中式和分布式后门攻击。

除了开发图神经网络上有效的后门攻击之外，我们还探索了如何利用后门攻击来实现图神经网络防御。我们在第五章介绍了一种基于后门攻击的图神经网络水印框架，可以有效验证图神经网络的所有权，从而保护模型的知识产权。

此论文提出的图神经网络后门攻击方法旨在帮助设计人员开发更安全的模型。作者希望我们的研究成果可以加深我们对图神经网络后门攻击的理解，并推动图神经网络模型设计者开发更安全的模型。

1

Introduction

Data can be naturally represented by graph structures in many application areas, including proteomics [6], image analysis [79], software engineering [8, 27], and natural language processing [140]. However, these tasks require dealing with graph-structured data that contains rich relational information between elements and cannot be well handled by traditional deep learning models, e.g., Convolutional Neural Networks (CNNs). Graph Neural Networks (GNNs) are proposed to process graph-structure data to learn representations of graphs via feature propagation and aggregation [141].

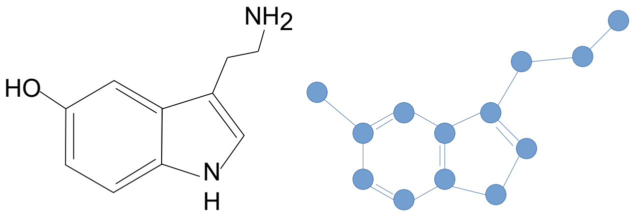
Given the importance of graph-related applications and the successful applications of GNNs, both academia and industry are interested in the security and privacy of GNNs. In recent years, some researchers have begun to focus on adversarial attacks on GNNs, and it has been demonstrated that GNNs are vulnerable to many adversarial attacks, one of which is the backdoor attack. Backdoor attacks on GNNs aim to train a GNN that performs normally on clean data but will output a pre-determined label for data with a certain trigger. Backdoor attacks on GNNs can lead to severe security concerns, e.g., the attacker can escape malicious user detection on social networks by implementing backdoor attacks.

Exploring the backdoor attacks on GNNs is not a trivial task due to the specific characteristics of graph-structure data. Still, it is an essential part of developing robust GNNs, as it helps mitigate backdoor attacks on GNNs and develop possible approaches to protect the ownership of GNNs.

In this chapter, we introduce the concept of backdoor attacks on GNNs that form the basis of this thesis. Section 1.1 introduces graph-structured data and the relationship between graph-structured data and other domain data. Section 1.2 introduces how to apply machine learning methods to graph-structured data. Specifically, we here first discuss why CNNs cannot be directly applied to graph-structured data and then introduce GNNs, which are designed to perform on graph-structured data. We here also introduce the Federated GNNs in which federated learning is applied due to privacy concerns. Section 1.3 introduces the security risks of GNNs and specifically discusses backdoor attacks on GNNs, which is one of the essential concepts this thesis addresses. The problem statement and research questions this thesis addresses are discussed in Section 1.4. Finally, Section 1.5 and 1.6 list the contributions and outline of this thesis, respectively.

1.1. GRAPH-STRUCTURED DATA

Graph-structured data is ubiquitous throughout the natural and social sciences, from quantum chemistry to social networks. In the most general view, a graph is simply a collection of objects, i.e., nodes, along with a set of interactions, i.e., edges, between pairs of these objects. On the one hand, graph-structured data can be used to represent physical networks. For instance, in the chemical domain, we can use the nodes in a graph to represent proteins and the edges to represent the bonds between the proteins, as shown in Figure 1.1. On the other hand, graph-structured data can also be applied to model the complex relationships in our lives. For example, to encode a social network as a graph, we can use nodes to represent individuals and edges to represent that two individuals are friends, as shown in Figure 1.2.



(a) (b)

Figure 1.1: Graph representation of the serotonin molecule. (a) is the molecular structure of serotonin, and (b) is the graph representation of the molecule.

The power of graph formalism lies in its focus on *relationships between points* rather than only the properties of individual points, as well as in its generality. The same graph formalism can be used to represent chemical structures, social networks, or the connections between terminals in a telecommunications network. The key structural property of graphs is that the nodes in a graph are usually not assumed to be provided in any

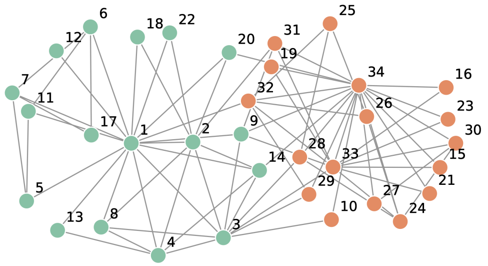


Figure 1.2: A network representation of social relationships among the 34 individuals in the karate club studied by Zachary [159]. The network captures 34 members of a karate club, documenting links between pairs of members who interacted outside the club.

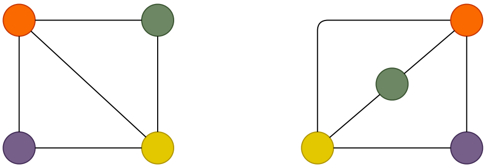


Figure 1.3: Isomorphic graphs. *Isomorphic* is an edge-preserving bijection between two graphs. Two isomorphic graphs shown here are identical up to the reordering of their nodes.

particular order, and thus, any operations performed on graphs should not depend on the ordering of nodes [17]. The property that functions acting on graphs should satisfy is thus *permutation invariance*, and it implies that for any two *isomorphic* graphs, the outcomes of these functions are identical. Figure 1.3 shows two isomorphic graphs.

It is well-known that the impact of deep learning was particularly dramatic in computer vision, natural language processing, and speech recognition. These applications share a geometric common denominator: an underlying Euclidean or grid-like structure [17]. From the perspective of Bronstein et al. [17], grids can be obtained as a particular case of graphs with special adjacency. For example, we can think of images as fixed-size grid graphs. Text and speech are sequences, so that we can think of them as line graphs. However, since the order of nodes in a grid is fixed, machine learning models for signals defined on grids are no longer required to account for permutation invariance and have a strong geometric prior, i.e., translation invariance. Since Euclidean spaces are prototypically defined by $\mathbb{R}^n$ (with dimension n), *Euclidean data* refers to data that is sensibly modeled as being plotted in n-dimensional linear space, such as images where the x and y coordinates refer to the location of each pixel and the z coordinate refers to its value. However, some data cannot be mapped neatly into the Euclidean spaces (i.e., $\mathbb{R}^n$), for example, graphs, whose underlying structure is non-Euclidean.

1.2. MACHINE LEARNING ON GRAPH DATA

Machine learning is a branch of artificial intelligence (AI) and computer science. It is a set of algorithms that parse data, learn from them, and then apply what they have learned to make intelligent decisions. Given the widespread prevalence of graphs in real-world applications, there has been a surge of interest in applying machine learning methods to graph-structured data. However, for graph-structured data, it is challenging to define networks with strong structural priors, as structures can be arbitrary and vary significantly across different graphs. Even different nodes within the same graph can have different structures. It also does not help that most existing machine learning algorithms (e.g., CNNs) have a core assumption that nodes in a sample (e.g., pixels in an image) are independent, which is not true for graph data because each node in a graph is related to others by links of various types. We here first introduce the widely-used convolutional neural networks and then discuss how to apply convolution operation on graph data.

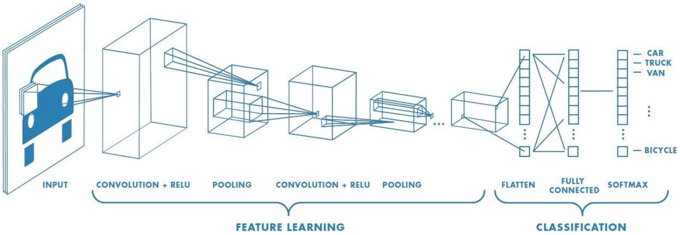


Figure 1.4: CNNs in an autonomous driving system.

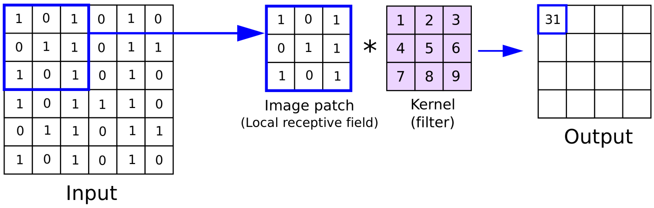


Figure 1.5: An example of a convolution operation.

1.2.1. Convolutional Neural Networks

A Convolutional Neural Network is a class of artificial neural networks most commonly applied to analyze visual imagery [126]. CNNs use a mathematical operation called *convolution* in place of general matrix multiplication in at least one of their layers. A CNN consists of an input layer, some hidden layers, and an output layer. The hidden layers include one or more layers that perform convolutions. Figure 1.4 presents an architecture of a CNN, which is applied in an autonomous driving system to detect the objects on the way. Typically, a convolution operation performs a dot product of the convolution kernel with the layer's input matrix, and its activation function can be ReLU. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. Convolution layers are followed by other layers, such as pooling layers, fully connected layers, and normalization layers. Figure 1.5 shows an example of a convolution operation.

1.2.2. Graph Neural Networks

The non-Euclidean nature of the graph-structured data implies that there are no such familiar properties as global parameterization, a common system of coordinates, and vector space structure. Consequently, basic operations like convolutions that are taken for granted in the Euclidean data cannot be directly applied to irregular graph domains. Specifically, CNNs have a property of global parameterization because, in images, each pixel has the same neighborhood structure, allowing to apply the same filter weights at multiple locations in the image. However, in graphs, each node can have a different number of neighbors, making it impossible to apply the same filter weights for each node in a graph. What's more, CNNs are not permutation invariant as the output of CNNs will

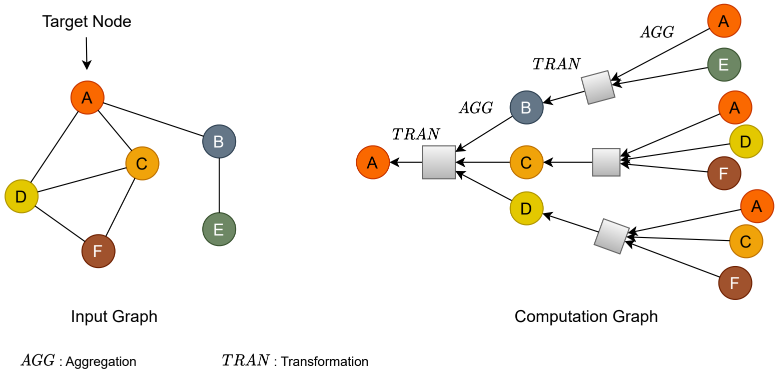


Figure 1.6: An input graph and the computation graph of the target node.

change when the order of its input is altered. However, as mentioned in Section 1.1, the nodes in a graph are usually not assumed to be provided in any particular order, and any operations performed on graphs should not depend on the ordering of nodes, i.e., *permutation invariance*. Therefore, we cannot perform CNNs directly on graphs.

Graph Neural Networks are a class of deep learning methods designed to perform inference on data described by graphs. The output of a GNN model is usually *Node Embedding*, which maps nodes to a d-dimensional embedding space so that similar nodes in the graph are embedded close to each other in the d-dimensional space. Specifically, our goal is to map nodes so that similarity in the embedding space approximates similarity in the graph. GNNs are methods that use the graph structure and node features to learn a representation vector of a node or the entire graph. For each node in a graph, we can determine a node computation graph. Figure 1.6 shows an input graph and the computation graph of the target node in the graph. With the node computation graph, we can see all the connections between the target node and other nodes, and the grey boxes in Figure 1.6 represent the aggregation functions in a GNN with three layers.

Node classification is perhaps the most popular machine-learning task on graph data. Beyond detecting malicious users in social networks, examples of node classification tasks include classifying the function of proteins in the molecular network [53] and classifying the topic of documents based on hyperlink or citation graphs [67]. Another commonly used application of GNNs is the graph classification task. The goal of the graph classification task is to predict the class label(s) for an entire graph. For instance, given a graph representing the structure of a molecule, we can build a regression model predicting the molecule's toxicity or solubility [45]. Another practical application of the graph classification task is to build a classification model to detect whether a computer program is malicious by analyzing a graph-based representation of its syntax and data flow [77].

1.2.3. Federated GNNs

GNNs face challenges when centrally trained because of privacy concerns, regulatory restrictions, and commercial competition. For example, the financial institution may utilize GNN as a fraud detection model, but they can only have transaction data of its registered users (no data of other users because of privacy concerns). Thus, the model is not effective for other users. Similarly, in a drug discovery industry that applies GNNs, pharmaceutical research institutions can dramatically benefit from other institutions' data, but they cannot disclose their private data for commercial reasons [55].

Federated Learning (FL) is a distributed learning paradigm that works on isolated data. In FL, clients can collaboratively train a shared global model under the orchestration of a central server while keeping the data decentralized [66, 91]. FL can be categorized into three categories based on how data is partitioned in the sample and feature space: Horizontal Federated Learning (HFL), Vertical Federated Learning (VFL), and Federated Transfer Learning (FTL) [84]. While FL has been widely studied in Euclidean data, e.g., images, texts, and sound, there are increasing studies about FL in graph data, which we denote as *Federated GNNs*.

1.3. BACKDOOR ATTACKS ON GNNS

As new generalizations of traditional deep neural networks to graphs, GNNs inherit both the advantages and disadvantages of traditional DNNs. Traditional DNNs are easily fooled by adversarial attacks [47, 71]. In other words, the adversary can insert slight perturbation during either the training or testing phases, and the DNN models will fail. It is evident [169, 31] that GNNs also inherit this drawback. The attacker can generate graph adversarial perturbations by manipulating the graph structure or node features to fool the GNN models. As illustrated in Figure 1.7, originally, node E was classified by the GNN model as a green node; after node E creates a new connection with node C and modifies its features, the GNN model misclassifies it as a yellow node. Such vulnerability of GNNs has raised tremendous concerns about applying them in security-critical applications such as financial systems and risk management. For example, in a credit scoring system, fraudsters can fake connections with several high-credit customers to evade the fraudster detection models, and spammers can easily create fake followers to increase the chance of fake news being recommended and spread. Though adversarial attacks have been extensively studied in the image domain, we still need dedicated efforts for graphs due to unique challenges – (1) graph structure is discrete; (2) the nodes in the graph are not independent; and (3) it is difficult to measure whether the perturbation on the graph is imperceptible or not.

The adversarial attacks on GNNs can be categorized into evasion attacks and poisoning attacks, depending on which phase the attacker inserts the adversarial perturbation. Specifically, evasion attacks happen in the model testing phase, while poisoning attacks happen in the model training phase. A backdoor attack is one type of poisoning attack. In a backdoor attack, the adversary aims to train a neural network that correctly solves the desired task on clean data but will exhibit malicious behavior once presented with a certain trigger [50]. A backdoor attack is a practical attack as it is both powerful and stealthy [50].

In recent years, there have been many works exploring backdoor attacks in many domains, such as image, video, and text domains [50, 166, 152]. However, backdoor attacks on GNNs are not extensively explored. Since GNNs are increasingly used for security applications [1, 80], it is important to study the backdoor attack on GNNs. Otherwise, secu-

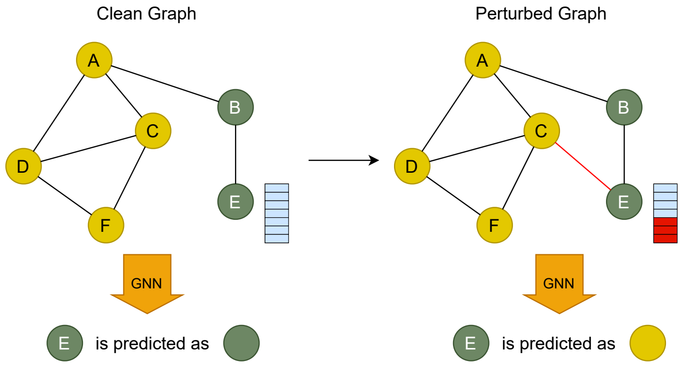


Figure 1.7: An example of an adversarial attack on graph data. The goal of the GNN is to predict the color of the nodes. Here, node E is the target node. The attacker aims to change the prediction of GNN on node E by modifying its connections with other nodes and features.

rity concerns will remain. For instance, in a Bitcoin transaction ego network [135], where the nodes are the transactions, and the edge between two nodes indicates the flow of Bitcoin from one transaction to another, the attacker can attack the GNNs to classify an illegal transaction as a legal one. Also, most existing works on backdoor attacks are applied to the Euclidean data, e.g., images and words. The backdoor trigger generation methods and injecting position are different between graph data and images/words. Thus, the backdoor attacks on other domains cannot be directly applied to the graph domain.

Although there are some works conducting research on backdoor attacks on GNNs. However, these works focus on GNN models in centralized training. In the Federated GNNs, the malicious updates will be weakened in the aggregation function. Also, there can be more than one malicious client, while in the centralized GNNs, there is only one client. Thus, we should expect different behavior of backdoor attacks on the Federated GNNs. Then, it is crucial to investigate if existing countermeasures that have been tested mostly with Euclidean data are still effective for backdoor attacks on Federated GNNs to understand how to deploy trustworthy AI systems.

In addition to considering backdoor attacks on GNNs for offensive purposes, backdoor attacks on GNNs can also be used to protect the IP (Intellectual Property) of the GNN models. Building a powerful GNN model is not a trivial task, as it requires a large amount of training data, powerful computing resources, and human expertise. Moreover, with the development of adversarial attacks, e.g., model stealing attacks, GNNs raise challenges to model authentication. To avoid copyright infringement on GNNs, verifying the ownership of the GNN models is necessary. Digital watermarking is typically used to identify ownership of the copyright of media signals, e.g., audio, video, and image data [73]. The idea of watermarking neural networks is similar to traditional digital watermarking in multimedia data. Following the idea of [2], this thesis explores a watermarking method on GNNs based on backdoor attacks. More precisely, the backdoor triggers are used as digital watermarks to identify the ownership of a GNN model.

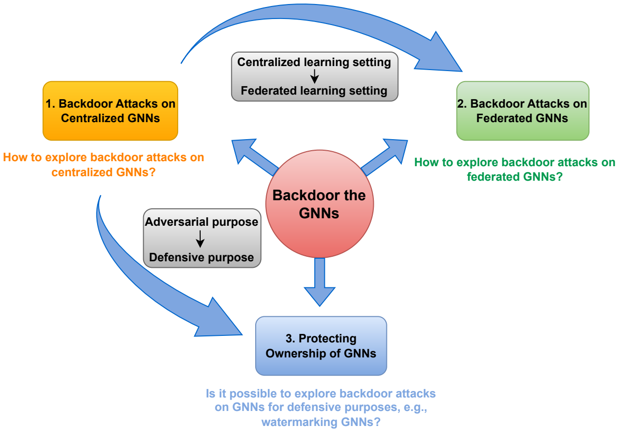

Centralized learning setting
Federated learning setting
1. Backdoor Attacks on Centralized GNNs
How to explore backdoor attacks on centralized GNNs?
2. Backdoor Attacks on Federated GNNs
How to explore backdoor attacks on federated GNNs?
Backdoor the GNNs
Adversarial purpose
Defensive purpose
3. Protecting Ownership of GNNs
Is it possible to explore backdoor attacks on GNNs for defensive purposes, e.g., watermarking GNNs?

Figure 1.8: The illustration of three research subquestions in this thesis.

1.4. PROBLEM STATEMENT

Considering that existing backdoor attacks on Euclidean data cannot be directly applied to the graph data due to the differences between graph-structured data and other Euclidean data, this thesis aims to answer the following:

Can we design backdoor attacks on GNNs?

This question is split into three subquestions according to the different learning settings of GNNs and the different purposes of backdoor attacks on GNNs, as shown in Figure 1.8:

- Can we design backdoor attacks on centralized GNNs?
- Can we design backdoor attacks on federated GNNs?
- Is it possible to design backdoor attacks on GNNs for defensive purposes, e.g., watermarking GNNs?

1.5. CONTRIBUTION OF THE THESIS

This thesis designs and evaluates efficient backdoor attacks on the centralized GNNs, investigates the backdoor attacks on the Federated GNNs, and investigates the potential application of backdoor attacks on GNNs for defensive purposes. The contributions of this thesis are as follows:

- We provide the first study on the explainability of triggers for backdoor attacks on GNNs. In Chapter 3, we introduce the explainability-based backdoor attacks on

GNNs. This work is, to the best of our knowledge, the first published work to explore the impact of trigger-injecting positions on the backdoor attack performance on GNNs.

- Our design of a novel explanation framework in Chapter 3 to analyze the causes of the difference between two trigger-injecting strategies is, to the best of our knowledge, the first method that can verify the difference with quantitative analysis (recall score). This contribution provides insights into the interaction between the explainability and backdoor attacks on GNNs, enhancing our understanding of the robustness of GNNs.

- We develop the clean-label backdoor attack on GNNs in Chapter 3, which is, to the best of our knowledge, the first work on clean-label backdoor attacks on GNNs.

- We propose a label-only membership inference attack (MIA) against GNNs for the node classification task, which achieves competitive or better performance than state-of-the-art probability-based MIAs, in Chapter 4.

- We develop two types of backdoor attacks on Federated GNNs in Chapter 4, which is, to the best of our knowledge, the first work studying backdoor attacks specifically in the context of Federated GNNs. This research fills a significant gap in our understanding of backdoor attacks in distributed GNN settings.

- We propose a watermarking framework to verify the ownership of GNN models. The watermarking mechanism proposed in Chapter 5 is, to the best of our knowledge, the first watermarking framework for GNNs on the graph classification task.

1.6. OUTLINE

The thesis is structured as follows:

Chapter 2
Background

In Chapter 2, we introduce the key concepts throughout this thesis, including the mathematical formulation of graph data, representative GNN models, commonly used applications of GNNs, and federated learning on GNNs.

Chapter 3
Backdoor Attacks on Centralized GNNs

In Chapter 3, we explore the impact of trigger-injecting position on the performance of backdoor attacks on GNNs by conducting an experimental investigation. We apply two powerful GNN explainability approaches to select the most or least important area of the sample as the trigger-injecting position. Also, we propose a new backdoor attack on GNNs specifically targeting the node classification task. This attack leverages a subset of node features as a trigger pattern, demonstrating a novel approach to compromising

GNN models for the node classification task. To further explain the different attack performances of these two trigger-injecting strategies, we also design a novel explanation framework with quantitative analysis. In addition to the dirty-label backdoor attacks on centralized GNNs, we further design the clean-label backdoor attacks on centralized GNNs, which are more stealthy and require less information for the attacker. The papers included in this chapter are:

- Explainability-based Backdoor Attacks Against Graph Neural Networks. ***Xu, J.***, *Xue, M., & Picek, S. (2021). The ACM Workshop on Wireless Security and Machine Learning.*
- Rethinking the Trigger-injecting Position in Graph Backdoor Attack. ***Xu, J.***, *Abad, G., & Picek, S. (2023). The International Joint Conference on Neural Networks.*
- Poster: Clean-label Backdoor Attack on Graph Neural Networks. ***Xu, J.***, *Picek, S. (2022). The ACM Conference on Computer and Communications Security.*

Chapter 4
Backdoor Attacks on Federated GNNs

In Chapter 4, firstly, we propose a method of implementing MIA against GNNs under the label-only condition. The average attack accuracy, precision, and AUC values of our label-only MIA are competitive or even better than previous probability-based MIAs. Second, we investigate two types of backdoor attacks in horizontal Federated GNNs: centralized backdoor attacks (CBA) and distributed backdoor attacks (DBA). We also evaluate the attack performance for a different number of clients and different percentages of malicious clients. Based on our experiments of evaluating two types of backdoor attacks on Federated GNNs, we provide a number of observations, lessons learned, and general takeaways. We emphasize the importance of artifact evaluation and sharing for reproducible research and how they can greatly help the research process. The papers included in this section are:

- Label-Only Membership Inference Attack against Node-Level Graph Neural Networks. *Conti, M., Li, J., Picek, S.,* ***Xu, J.*** *(2022). the 15th ACM Workshop on Artificial Intelligence and Security.*
- More is Better (Mostly): On the Backdoor Attacks in Federated Graph Neural Networks. ***Xu, J.***, *Wang, R., Koffas, S., Liang, K., & Picek, S. (2022). The Annual Computer Security Applications Conference.*
- On Exploring Backdoor Attacks in Federated Graph Neural Networks. ***Xu, J.***, *Koffas, S., & Picek, S. (2022). The Learning from Authoritative Security Experiment Results (LASER) workshop.*
- Unveiling the Threat: Investigating Distributed and Centralized Backdoor Attacks in Federated Graph Neural Networks. ***Xu, J.***, *Koffas, S., & Picek, S. (2023). Digital Threats: Research and Practice.*

Chapter 5
Protecting Ownership of GNNs

In Chapter 5, we propose a watermarking framework to verify the ownership of GNN models for both the node and graph classification tasks. We clarify three phases of the watermarking framework, i.e., watermarked data generation, watermark embedding, and ownership verification. We further employ hypothesis testing to provide statistical analysis for the model ownership verification results. We show in Chapter 5 that in our watermarking method, the watermarked model has significantly different watermark accuracy from the clean model, validating the effectiveness of our watermarking method. This chapter has been published as:

- Watermarking Graph Neural Networks based on Backdoor Attacks. ***Xu, J.***, *Koffas, S., Ersoy, O., & Picek, S. (2023). The IEEE European Symposium on Security and Privacy*

Chapter 6
Discussion

Chapter 6 concludes this work and presents our findings to the research questions in Section 1.4. We also look forward to future work and provide ideas and research gaps that we have learned from the works included in this thesis.

2

Background

In this chapter, we introduce the key concepts throughout this thesis. First, we introduce the mathematical formulation of graph data in Section 2.1. Second, we introduce four representative GNN models, which are selected to evaluate in this thesis in Section 2.2. Two commonly used applications of GNNs are introduced in Section 2.3. We finally introduce the federated learning on GNNs in Section 2.4.

2.1. What is a graph?

Formally, a graph $G = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ between these nodes. We denote an edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{E}$. In many cases, we will be concerned with *simple graphs*, where the edges are all undirected, i.e., $(u, v) \in \mathcal{E} \leftrightarrow (v, u) \in \mathcal{E}$.

In the generic setting $\mathcal{E} \neq \emptyset$, the graph connectivity can be represented by an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{N \times N}$ (N is the number of nodes, $N = |\mathcal{V}|$). To represent a graph with an adjacency matrix, we order the nodes in the graph so that every node indexes a particular row and column in the adjacency matrix. We can then represent the presence of edges as entries in the matrix, i.e., $\mathbf{A}[u, v] = 1$ if $(u, v) \in \mathcal{E}$ and $\mathbf{A}[u, v] = 0$ otherwise. If the graph is *undirected*, then $\mathbf{A}$ will be a *symmetric* matrix, but if the graph is *directed* (i.e., edges are not in pairs), then $\mathbf{A}$ will not necessarily be symmetric. Some graphs can also have *weighted* edges, where the entries in the adjacency matrix are real-values rather than 0, 1. For example, a weighted edge in a chemical graph might indicate the strength of the association between two proteins. In this thesis, we focus on the undirected graphs.

In addition to the adjacency matrix, we also have *attribute* or *feature* information associated with a graph, e.g., the user properties like age and profile picture associated with a user in a social network. Most often, these are node-level attributes that we represent using a real-valued matrix $\mathbf{X} \in \mathbb{R}^{N \times m}$, where we assume that the ordering of the nodes is consistent with the ordering in the adjacency matrix. It is also often possible to endow the edges, or the entire graph, with features, but this thesis focuses on the graphs with only node features.

In modern GNNs, the node representation is computed by recursive aggregation and transformation of feature representations of its neighbors. After k iterations of aggregation, a node's representation captures both structure and feature information within its k-hop network neighborhood. Formally, the k-th layer of a GNN is:

$$x_v^{(k)} = AGGREGATION^{(k)}(\left\{z_v^{(k-1)}, \left\{z_u^{(k-1)} | u \in \mathcal{N}_v\right\}\right\}), \tag{2.1}$$

$$z_v^{(k)} = TRANSFORMATION^{(k)}(x_v^{(k)}), \tag{2.2}$$

where $z_v^{(k)}$ is the representation of node v computed in the k-th iteration. $\mathcal{N}_v$ are 1-hop neighbors of node v, and the $AGGREGATION(\cdot)$ is an aggregation function that can vary for different GNN models. $z_v^{(0)}$ is initialized as node feature, i.e., $z_v^{(0)} = x_v$. The $TRANSFORMATION(\cdot)$ function consists of a learnable weight matrix and activation function. For the node classification task, the node representation z_v is used for prediction. For the graph classification task, the READOUT function pools the node representations for a graph-level representation z_G:

$$z_G = READOUT(z_v; v \in \mathcal{V}). \tag{2.3}$$

READOUT can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function [156, 163]. These two popular applications of GNNs, i.e., node and graph classification tasks, are introduced in Section 2.3.

2.2. Representative GNN models

Graph Convolutional Networks (GCN)

GCN is an approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks [67]. GCN model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes. A multi-layer GCN follows the following layer-wise propagation rule:

$$\mathbf{Z}^{(k+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}^{(k)}\mathbf{W}^{(k)}\right) \tag{2.4}$$

Here, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix of the undirected graph G with added self-connections. $\mathbf{I}_N$ is the identity matrix, and $\tilde{\mathbf{D}}$ is the diagonal node degree matrix of $\tilde{\mathbf{A}}$, and $\mathbf{W}^{(k)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the $ReLU(\cdot) = max(0, \cdot)$. $\mathbf{Z}^{(k)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the k^{th} layer which outputs D-dimensional hidden representations; $\mathbf{Z}^{(0)} = \mathbf{X}$. Combining Equation 2.1, 2.2 and 2.4, the aggregation operation for each node in GCN is given as:

$$x_v^{(k)} \leftarrow \sum_{u \in \mathcal{N}_v \cup v} \frac{1}{\sqrt{d_v d_u}} z_u^{(k-1)}. \tag{2.5}$$

where d_v denotes the degree of node v.

GCN performs a non-linear transformation over the aggregated features to compute the node representation at layer k:

$$z_v^{(k)} \leftarrow ReLU(x_v^{(k)}\mathbf{W}^{(k)}).$$

Graph Isomorphism Network (GIN)

GIN is developed as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test [146]. With a simple architecture, GIN generalizes the WL test and hence achieves maximum discriminative power among GNNs. Let ϵ be a learnable parameter or a fixed scalar. The aggregation operation in GIN is given as follows:

$$x_v^{(k)} \leftarrow (1+\epsilon^{(k)}) \cdot z_v^{(k-1)} + \sum_{u \in \mathcal{N}_v} z_u^{(k-1)} \tag{2.6}$$

In GIN, multilayer perceptrons (MLPs) are used as the transformation operation:

$$z_v^{(k)} \leftarrow MLP^{(k)}(x_v^{(k)}) \tag{2.7}$$

Graph Sample and Aggregate (GraphSAGE)

The original GCN algorithm aforementioned is designed for semi-supervised learning in a transductive setting, and the exact algorithm requires that the full graph Laplacian is known during training. GraphSAGE is a framework for inductive representation learning on large graphs. It leverages node attribute information to efficiently generate representation on previously unseen data [53]. Specifically, there are three candidate aggregator functions in the GraphSAGE algorithm:

Mean aggregator. The mean aggregator is nearly equivalent to the convolutional propagation rule used in the transductive GCN framework. In particular, an inductive variant of the GCN approach can be derived with the following:

$$z_v^{(k)} \leftarrow \sigma(W^{(k)} \cdot \mathrm{MEAN}(z_v^{(k-1)} \cup z_u^{(k-1)} | u \in \mathcal{N}_v)) \tag{2.8}$$

LSTM aggregator. A more complex aggregator is based on an LSTM architecture. Compared to the mean aggregator, LSTMs have the advantage of larger expressive capability. However, it is important to note that LSTMs are not inherently symmetric (i.e., they are not permutation invariant) since they process their inputs in a sequential manner. In GraphSAGE, LSTMs are adapted to operate on an unordered set by simply applying the LSTMs to a random permutation of the node's neighbors.

Pooling aggregator. This aggregator is both symmetric and trainable. In this pooling approach, each neighbor's vector is independently fed through a fully connected neural network; following this transformation, an elementwise max-pooling operation is applied to aggregate information across the neighbor set:

$$z_v^{(k)} \leftarrow \max(\sigma(\mathbf{W}^{(k)} z_u^{(k-1)} + b) | u \in \mathcal{N}_v) \tag{2.9}$$

where max denotes the element-wise max operator. By applying the max-pooling operator to each of the computed features, the model effectively captures different aspects of the neighborhood set. In this thesis, GraphSAGE is applied with the max-pooling aggregator.

Graph Attention Networks (GAT)

In addition to the standard neighbor aggregation scheme mentioned above in (2.1) and (2.2), there are other non-standard neighbor aggregation schemes, e.g., weighted average via attention in GAT [127]. Specifically, given a shared attention mechanism a, attention coefficients can be computed by:

$$e_{vu} = a(\mathbf{W}z_v^{(k-1)}, \mathbf{W}z_u^{(k-1)}) \tag{2.10}$$

that indicate the importance of node u's features to node v. Then, the normalized coefficients can be computed by using the softmax function:

$$\alpha_{vu} = softmax_u(e_{vu}). \tag{2.11}$$

Finally, the next-level feature representation of node $\boldsymbol{v}$ is:

$$z_v^{(k)} = \sigma\left(\frac{1}{P}\sum_{p=1}^{P}\sum_{u\in\mathcal{N}_v}\alpha_{vu}^{p}\mathbf{W}^{p}z_u^{(k-1)}\right), \tag{2.12}$$

where α_{vu}^p are the normalized coefficients computed by the p-th attention mechanism a^p and $\mathbf{W}^p$ is the corresponding input linear transformation's weight matrix.

2.3. APPLICATIONS OF GNN

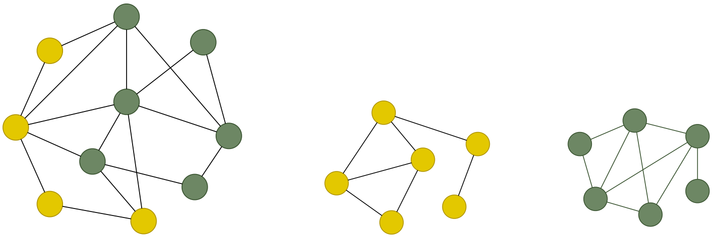
There are some general types of prediction tasks on graphs: node classification task, graph classification task, and link prediction task. We here introduce two commonly-used applications of GNNs, which are also two applications this thesis focuses on.

Node Classification Task

The node classification task is concerned with predicting the identity or role of each node within a graph, given the true labels on a training set of nodes $V_{train} \subset V$. Suppose we are given a large social network dataset with millions of users, but we know that a significant number of these users are actually malicious (e.g., people who spread terrorism information). Identifying these users could be important for many reasons: the corresponding organizations can prevent terrorist attacks, or society can be more stable without these malicious users. Manually examining every user to determine if they are malicious would be prohibitively expensive, so ideally, we would like to have a model that could classify users as malicious (or not) given only a small number of manually labeled examples. Node classification is perhaps the most popular machine-learning task on graph data. Beyond detecting malicious users in social networks, examples of node classification tasks include classifying the function of proteins in the molecular network [53] and classifying the topic of documents based on hyperlink or citation graphs [67].

Graph Classification Task

The goal of the graph classification task is to predict the class label(s) for an entire graph.



(a) Node classification task.

(b) Graph classification task

Figure 2.1: Illustration of node and graph classification tasks of GNNs. (a) Node classification task where the goal is to predict the classes or labels of nodes. (b) Graph classification task where the goal is to classify a whole graph into different categories.

For instance, given a graph representing the structure of a molecule, we can build a regression model predicting the molecule's toxicity or solubility [45]. Another practical application of the graph classification task is to build a classification model to detect whether a computer program is malicious by analyzing a graph-based representation of its syntax and data flow [77].

The illustrations of node and graph classification tasks of GNNs are shown in Figure 2.1.

2.4. FEDERATED LEARNING ON GNNS

Federated Learning enables n clients to train a global model $\boldsymbol{w}$ collaboratively without revealing local datasets. Unlike centralized learning, where local datasets must be collected by a central server before training, FL performs training by uploading the weights of local models ($\{\boldsymbol{w}^i \mid i \in n\}$) to a parametric server. Specifically, FL aims to optimize a loss function:

$$\min_{\boldsymbol{w}} \ell(\boldsymbol{w}) = \sum_{i=1}^{n} \frac{k_i}{n} L_i(\boldsymbol{w}), L_i(\boldsymbol{w}) = \frac{1}{k_i} \sum_{j \in \mathscr{P}_i} \ell_j(\boldsymbol{w}, x_j), \tag{2.13}$$

where $L_i(\boldsymbol{w})$ and k_i are the loss function and local data size of i-th client, and $\mathscr{P}_i$ refers to the set of data indices with size k_i.

At the t-th iteration, the training can be divided into three steps:

- *Global model download.* All clients download the global model $\boldsymbol{w}_t$ from the server.
- *Local training.* Each client updates the global model by training with their datasets: $\boldsymbol{w}_t^i \leftarrow \boldsymbol{w}_t^i - \eta \frac{\partial L(\boldsymbol{w}_t, b)}{\partial \boldsymbol{w}_t^i}$, where η and b refer to learning rate and local batch, respectively.
- *Aggregation.* After the clients upload their local models $\{\boldsymbol{w}_t^i \mid i \in n\}$, the server updates the global model by aggregating the local models. In this paper, we use the averaging aggregation function: $\boldsymbol{w}_{t+1} \leftarrow \sum_{i=1}^{n} \frac{1}{n} \boldsymbol{w}_t^i$.

FL on graph data was introduced in [72], where each client is regarded as a node in a graph. When it comes to detecting financial crimes (e.g., fraud or money laundering), traditional machine learning tends to lead to severe overreporting of suspicious activities. Thanks to the reasoning ability of the graph neural network, its advantages can be well-reflected. Considering the need for privacy, [121] proposed the framework for Federated GNNs to optimize the machine learning model. Besides, other research works [64, 21, 138] have been dedicated to enhancing the security of Federated GNNs. By using secure aggregation, [64] proposed a method to predict the trajectories of objects via aggregating both spatial and dynamic information without information leakage. With differential privacy, [21] and [138] put forward a framework to train Federated GNNs for VFL and recommendation systems, respectively. Moreover, SpreadGNN was proposed in [56] to perform FL without a server.

3

Backdoor Attacks on Centralized GNNs

Backdoor attacks have been demonstrated as a security threat for machine learning models. While there are already some works on backdoor attacks on Graph Neural Networks, the backdoor trigger in the graph domain is mostly injected into random positions of the sample. There is no extensive research analyzing and explaining the impact of trigger injecting position on the performance of backdoor attacks on GNNs. Also, in prior backdoor attacks on GNNs, the adversary introduces arbitrary, often clearly mislabeled inputs to the training set. In such a scenario, the poisoned inputs are likely to be detected as outliers. To make the resulting poisoned inputs appear consistent with their original labels so that it is more difficult to detect the poisoned inputs, it is crucial to study the clean-label backdoor attacks on GNNs. Therefore, in this chapter, we aim to answer the first subquestion in Section 1.4. Specifically, we here provide solutions to the following research questions, i.e., What is the impact of trigger injection position on graph backdoor? and How can we explain it? and Can we design the clean-label backdoor attack on GNNs?

This chapter designs explainability-based backdoor attacks against GNNs applying two GNN explainability approaches. Also, this chapter designs a novel explanation framework to analyze and explain the impact of trigger injecting position on the performance of backdoor attacks on GNNs. What's more, this chapter explores a new kind of backdoor attack, i.e., clean-label backdoor attack, on GNNs. In a clean-label backdoor attack, the resulting poisoned inputs appear to be consistent with their label and, thus, are less likely to be filtered as outliers.

This chapter is based on: 1) "Explainability-based Backdoor Attacks Against Graph Neural Networks." ***Xu, J.**, Xue, M., & Picek, S. (2021). The ACM Workshop on Wireless Security and Machine Learning.* 2) "Rethinking the Trigger-injecting Position in Graph Backdoor Attack." ***Xu, J.**, Abad, G., & Picek, S. (2023). The International Joint Conference on Neural Networks.* 3) "Poster: Clean-label Backdoor Attack on Graph Neural Networks." ***Xu, J.**, Picek, S. (2022). The ACM Conference on Computer and Communications Security.*

3.1. Introduction

Several studies showed that GNNs are also vulnerable to backdoor attacks. Zhang et al. proposed a subgraph-based backdoor attack to GNNs for graph classification task [164]. Xi et al. presented a subgraph-based backdoor attack to GNNs, but this attack can be instantiated for both node classification and graph classification tasks [143]. In the backdoor attacks on GNNs, the trigger injecting position impacts the attack's performance in terms of the attack success rate and clean accuracy drop. The existing works exploring backdoor attacks on GNNs either select trigger injecting position randomly, in which situation the attack may be easily detected by the defender [164], or use a computationally intensive algorithm to get the trigger injecting position, as shown in [143]. If we know how to quickly select the optimal (or close to optimal) trigger-injecting position in backdoor attacks on GNNs, we can achieve high attack performance and good evasion of the defender's detection mechanisms. Further, we can develop more robust GNN models. Unfortunately, since graph data have characteristics of complex relationships and interdependencies between objects, common explainability approaches for CNNs, such as Shapley value [3], are not suitable to explain the predictions of GNNs to select the optimal trigger injecting position.

Although interpretability methods for non-graph neural networks are not suitable for explaining predictions made by GNNs, recently, some works try to interpret GNNs. Ying et al. proposed to utilize mutual information to find a subgraph with associated features for interpreting the predicted label of a node or graph being explained [155]. Huang et al. presented a method utilizing predicted labels from both the node being explained and its neighbors, which enables to capture more local information around the node and give a finite number of features as explanations in an intuitive way [61]. If we can utilize powerful neural network approaches that explain predictions made by GNNs to understand the performance of backdoor attacks on GNNs, we can gain knowledge of how to quickly select the optimal trigger-injecting position and explore the impact of different trigger-injecting positions on the attack performance.

Once we explore the impact of the backdoor trigger-injecting positions on GNNs, it is intuitive to further explore the reason behind the different performance of different trigger-injecting positions. There are already some works on explaining backdoor attacks in the image domain through visualization techniques [50, 144]. For example, [50] plotted the average activations of the backdoored model's last convolutional layer over clean and backdoored images to explain their attack. [144] used the Grad-CAM [109] visualization method to explain the backdoor attack in federated learning. One example of explaining a backdoor attack in the image domain with Grad-CAM is shown in Fig. 3.1. Comparing the heatmaps of the clean and poisoned images on the backdoored model, we can clearly understand how the backdoored model recognizes the trigger pattern to achieve the backdoor attack. In contrast, applying visualization techniques to explain the backdoor attack behavior in the graph domain is difficult. First, the complexity of the visual representation of a graph is much larger than visualizing an image, especially for large graphs [60]. Second, visualizing the graph neural networks to explain the backdoor attack is not trivial as it is a time-consuming or even impossible process [65]. Therefore, instead of using the visualization method, we explain the difference between different trigger-injecting strategies by computing an evaluation metric.

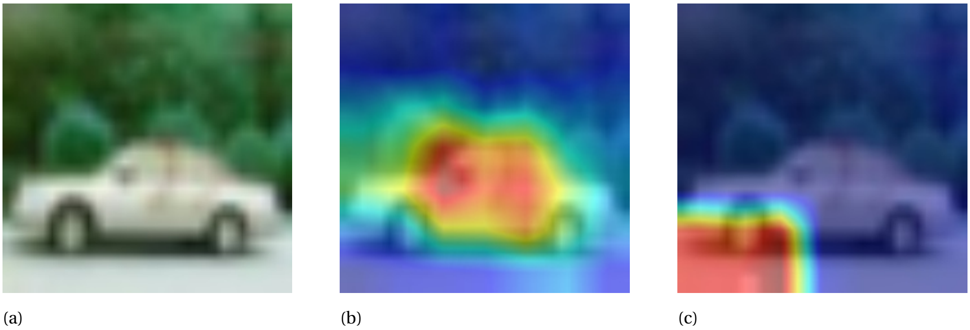


Figure 3.1: An example of using Grad-CAM to explain a backdoor attack in the image domain. (a) clean image, (b) heatmap of clean image for the true label on the backdoored model (predicted as the true label), (c) heatmap of the poisoned image for the target label on the backdoored model (predicted as the target label).

Like prior backdoor attacks on CNNs, recent studies about backdoor attacks on GNNs assume the adversary can often introduce mislabeled inputs to the training dataset [164, 143]. Specifically, the attacker modifies the training dataset by injecting a backdoor trigger into some training samples and relabeling these samples to the chosen target label. Then the backdoored neural network classifier will output the attacker-chosen label when a trigger is injected into a testing sample. However, it has been demonstrated for CNNs that even a fairly simple filtering process will detect the poisoned samples as outliers [124]. More importantly, any subsequent human inspection will deem these inputs suspicious and thus potentially reveal the attack [124]. To make the resulting poisoned inputs appear consistent with their labels so that it is more difficult to detect the poisoned inputs, a clean-label backdoor attack was proposed [124]. In the clean-label backdoor attack, the adversary only poisons inputs of the target class without changing the true labels. The backdoored model aims to predict the testing sample with a trigger into the target label. Although graphs are difficult to visualize directly for humans, unlike images and texts [155], there are still some straightforward methods to detect poisoned graph samples. For instance, we can apply the GNN prediction explanation tool, e.g., GNNExplainer [155], to visualize semantically relevant graph structures that are interpretable for humans. Specifically, we first get explanation subgraphs for each class, and then for each graph sample, check whether it contains the corresponding class explanation subgraphs. If a graph sample in a specific class does not contain the corresponding class explanation subgraphs, we can consider it an outlier. Therefore, it is also crucial to study the clean-label backdoor attacks on GNNs.

In this chapter, we design and explore backdoor attacks on centralized GNNs. First, in Section 3.2, we propose utilizing powerful neural network approaches that explain predictions made by GNNs to understand the performance of backdoor attacks on GNNs. Indeed, backdoor attacks on GNNs have been presented [164, 143] but how to quickly select the optimal trigger injecting position and what is the impact of different trigger injecting position on the attack performance have not been explored. In this section, we seek to bridge this gap. We utilize GNNExplainer, an approach for explaining predictions made by GNNs, to analyze the impact of trigger injecting position for the backdoor attacks on GNNs for the graph classification task. We propose a new backdoor attack on GNNs for the node classification task, which uses a subset of node features as a trigger pattern. Additionally, we explore GraphLIME, a local interpretable model explanation for graphs, to explore the proposed backdoor attack on the node classification task through modifying a different subset of node features. Through empirical evaluation using benchmark datasets and state-of-the-art models, we verify that our approach can quickly select the optimal trigger-injecting position to implement a powerful backdoor attack on GNNs. Furthermore, we see that the attacker can select the least important parts of the graph to inject the trigger, thus reducing the chances of easy detection by the defender.

Second, in Section 3.3, we design a novel explanation framework to analyze the causes of the difference between different trigger-injecting strategies. Specifically, we compute the similarity of the predicted mask of the representative features from the backdoored model and the target mask of the representative features from the clean model. As there has been an increasing number of studies on trustworthy GNNs [160, 30, 132], this work also contributes to the exploration of GNN robustness by investigating the effectiveness of backdoor attacks in GNN models using an explainability tool.

Third, in Section 3.4, we explore a new backdoor attack on GNNs, i.e., clean-label backdoor attack, that only poisons inputs of the target class without changing the original labels. More specifically, we aim to investigate whether using clearly mislabeled graphs is necessary for implementing a backdoor attack on GNNs. In this section, we aim to answer this question: can we carry out such backdoor attacks by insisting that each poisoned graph and its label must be consistent? Initial experimental evaluations showed that a clean-label backdoor attack could achieve a high attack success rate and low clean accuracy drop. This study highlights the concern of clean-label backdoor attacks on GNNs, which are more insidious.

3.2. EXPLAINABILITY-BASED BACKDOOR ATTACKS

Recently, several explainability techniques in GNNs have been proposed, such as XGNN [158], GNNExplainer [155], PGExplainer [89], and GraphLIME [61]. These methods are developed from different angles and provide different levels of explanations. For instance, GNNExplainer is a model-agnostic approach for providing explanations on predictions of any GNN-based model. Given a trained GNN model and its prediction(s), GNNExplainer returns an explanation in the form of a small subgraph of the input graph together with a small subset of node features that are most influential for the prediction(s) [155]. In addition, GraphLIME is a local interpretable model explainability method for graphs. More specifically, to explain a node, GraphLIME generates a nonlinear interpretable model from its N-hop neighborhood and then computes the most n representative features as the explanations of its prediction using HSIC Lasso [61]. We here utilize two explainability tools for GNNs, i.e., GNNExplainer and GraphLIME, to analyze the impact of trigger-injecting position for the backdoor attacks on GNNs for both graph and node classification tasks.

3.2.1. EXPLAINABLE BACKDOOR ATTACKS

Threat Model

We assume our threat model similar to the existing backdoor attacks, e.g., [74]. Given a pre-trained GNN model Φ_o, the adversary forges a backdoored GNN Φ by perturbing its model parameters without modifying the neural network architecture. We assume the attacker has access to a dataset D sampled from the training dataset. Specifically, in the graph classification task, the attacker can inject a trigger (graph) to each intended poisoned training graph and change the label to an attacker-chosen target label. In the node classification task, the attacker can inject a feature trigger (feature vector) to each intended poisoned training node and relabel the label to the target label. Consequently, our attack is a gray-box attack that does not modify the GNN's model architecture but perturbs the model parameters. This represents a realistic model occurring in real-world settings. For instance, if the training dataset is collected from public users, the malicious users can provide trigger-embedded training data.

Backdoor Attacks on Graph Classification

Since most graph classification tasks are implemented by utilizing GNNs to learn the network structure, we focus on subgraph-based backdoor attacks on the graph classification task. Figure 3.2 illustrates the pipeline of subgraph-based backdoor attack on GNNs for the graph classification task. Formally, in the training phase, the attacker injects a trigger (graph) g_t to a subset of the original training dataset and changes their labels to the attacker-chosen target label to obtain the backdoored training dataset. A GNN model trained using the backdoored training dataset is called backdoored GNN Φ. Then in the testing phase, the adversary injects the same trigger to a given graph G. If we define such trigger-embedded graph as G_{g_t}, the adversary's objectives can be defined as:

$$\begin{gathered}\Phi(G_{g_t}) = y_t \\ \Phi(G) = \Phi_o(G)\end{gathered} \tag{3.1}$$

The first objective in Eq. (3.1) means that all the trigger-embedded graphs are required to be misclassified to the target class y_t, i.e., attack effectiveness. In contrast, the second objective ensures that the backdoored GNN performs indistinguishably on normal graphs compared to the original GNN, i.e., attack evasiveness.

It is challenging to find the optimal trigger injecting position so that the adversary can reach several goals: 1) high attack success rate, 2) high accuracy in normal graphs, and 3) difficult to detect by the defender. More precisely,

- if we sample t nodes from the graph uniformly at random as the trigger nodes in the trigger graph, the trigger will likely be injected into a subgraph that is important for the GNN's final prediction. As a result, the defender can detect the trigger-embedded graphs easily;
- to achieve the second objective, we can select a subgraph similar to g_t in the graph as the trigger injecting position. However, the computation of the similarity between graphs is a complex task as subgraph isomorphism is known to be NP-complete. The graph matching algorithms require an exponential time for computation [29].

To overcome the above challenges, we utilize GNNExplainer to optimize the trigger

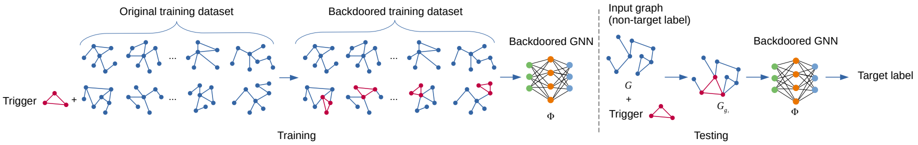


Figure 3.2: Illustration of subgraph based backdoor attack on GNNs for graph classification task.

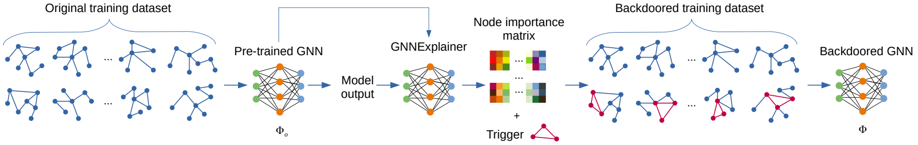


Figure 3.3: The framework of the backdoor attack on graph classification task based on GNNExplainer.

injecting position to ensure the attack effectiveness and attack evasiveness at the same time.

- We first apply GNNExplainer to analyze the prediction of GNNs to understand the impact of each structure in the graph on the classification result from GNNs.
- Instead of selecting t nodes from the graph uniformly at random as the trigger nodes, we select the t least important nodes in the graph as the trigger injecting position, which results in difficult-to-detect trigger-embedded graphs.

The overall framework of backdoor attack on graph classification task based on GNNExplainer is shown in Figure 3.3. Given a pre-trained GNN and its predictions, through GNNExplainer, the importance value of nodes for each graph can be computed. Based on the node importance matrix, we select the optimal trigger injecting position for each intended poisoned graph and then obtain the backdoored GNN by training the pre-trained GNN model Φ_o with the backdoored training dataset.

Backdoor Attacks on Node Classification

The currently proposed backdoor attack on GNNs for the node classification task defines triggers as specific subgraphs, i.e., given an arbitrary subgraph g in G, by replacing g with the trigger (graph) g_t, the adversary attempts to force the unlabeled nodes within K hops to g to be misclassified into the target label y_t.

Here, we propose a new method to implement backdoor attacks on GNNs for the node classification task. We assume that the adversary has access to G, including the graph structure information A and the node feature information X. Each node v in the graph G has its feature vector x. Given an arbitrary node in the graph, by changing the value of a subset of its features as a feature trigger, the attacker aims to force the node to be classified to the target class y_t and simultaneously perform normally in other unmodified nodes. Formally, the adversary's objectives can be defined as:

$$\begin{gathered}\Phi(v, x_t; G) = y_t \\ \Phi(v, x; G) = \Phi_o(v, x; G)\end{gathered} \tag{3.2}$$

Here, x_t represents the feature vector with trigger, obtained by changing the features' values in specific dimensions.

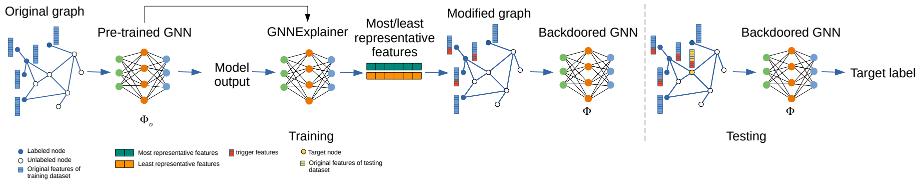


Figure 3.4: The framework of backdoor attack on node classification task based on GraphLIME.

Table 3.1: Dataset statistics.

Datasets	# Graphs	Avg. # nodes	Avg. # edges	Classes	Graphs [Class]	Target class
Mutagenicity	4,337	30.32	30.77	2	2,401[0], 1,936[1]	1
facebook_ct1	995	95.72	269.01	2	498[0], 497[1]	0
Cora	1	2,708	5,429	7	351[0], 217[1], 418[2], 818[3], 426[4], 298[5], 180[6]	6
CiteSeer	1	3,327	4,608	6	264[0], 590[1], 668[2], 701[3], 596[4], 508[5]	5

Similar to the graph classification task, these two objectives ensure the attack effectiveness and attack evasiveness. The key point is how to select specific dimensions of a feature vector as a trigger injecting position. Intuitively, we can select n features from the total features uniformly at random and change their values to a fixed value as the feature trigger. We can also use a GNN explainability method - GraphLIME to select the specific feature dimensions:

- We first apply GraphLIME to analyze the output of GNNs on the node classification task to compute the n most/least representative features.
- We change the value of the n most/least representative features to a fixed value as the feature trigger and retrain the GNN with backdoored training dataset to get the backdoored GNN.

The overall framework of the proposed backdoor attack on node classification task based on GraphLIME is shown in Figure 3.4.

3.2.2. EXPERIMENTAL ANALYSIS

Experimental Setting

Our experiments were run on an Intel Core i7-8650U CPU processor with 1.90 GHz frequency and 15.5 GiB memory. For all the experiments, we use the PyTorch framework.

Dataset. For the graph classification task, we use two publicly available real-world graph datasets. (i) Mutagenicity [92] - molecular structure graphs of mutagen and non-mutagen; (ii) facebook_ct1 [92] - a subset of the activity of the New Orleans Facebook community over three months, used to implement the classification task of distinguishing temporal graphs with vertex labels corresponding to observations of a dissemination process and temporal graphs in which the labeling is not a result of a dissemination process. We also use two real-world datasets for node classification task: Cora [110] and CiteSeer [110]. Table 3.1 shows the statistics of these datasets.

Dataset splits and parameter setting. In the graph classification task, for each graph classification dataset, we sample 2/3 of the graphs as the original training dataset and treat the remaining graphs as the original testing dataset. Among the original training dataset, we randomly sample η fraction of graphs to inject the trigger and relabel them with the target label, called the backdoored training dataset. We also inject our trigger to each original testing graph whose label is not the target label to generate the backdoored testing dataset, which is used to evaluate the attack effectiveness. There are several parameters in the attack's implementation: trigger size s, trigger density ρ, and poisoning intensity η. We set the trigger size s to be the γ fraction of the graph dataset's average number of nodes. Since the trigger size affects the attack effectiveness dramatically, we explore the impact of trigger size on the attack results. At the same time, we set other parameters as: $\rho = 0.8$ and $\eta = 5\%$ following the parameter setting in [164]. We use Erdős-Rényi (ER) model [44] to generate the trigger with graph density $\rho = 0.8$.

In the node classification task, we split 20% of the total nodes as the original training dataset (labeled) for each dataset. To generate the backdoored training dataset, we sample 15% of the original training dataset to inject the feature trigger and relabel these nodes with the target label. The trigger size is set to 10% of the total number of node feature dimensions. We set these parameters as they provided the best results after conducting a tuning phase. In the node classification task, each node feature has a value of 0 or 1, and here we set the value of the modified node features to 1 (note, the values could also be set to 0).

Models. In our experiment, we use the popular GIN [146] and GraphSAGE [53] models for the graph classification task as these two methods are the state-of-the-art GNN models. For node classification, we use GAT [127] model as the pre-trained GNN model.

Attack evaluation metrics. We use the *attack success rate (ASR)* to evaluate the attack effectiveness. Specifically, in the graph classification task, the ASR measures the proportion of trigger-embedded inputs (the original label is not the target label) that are misclassified by the backdoored GNN into the target class y_t chosen by the adversary. The trigger-embedded inputs are

$$\begin{aligned} D_{g_t} &= \left\{(G_{1,g_t}, y_1), \ldots, (G_{n,g_t}, y_n)\right\} \mathit{and} \\ D_{x_t} &= \left\{(v_{1,x_t}, y_1), \ldots, (v_{n,x_t}, y_n)\right\} \end{aligned}$$

for graph and node classification tasks, respectively. Formally, the ASR can be defined as:

$$\begin{aligned} Attack\,Success\,Rate &= \frac{\sum_{i=1}^{n} \mathbb{I}(\Phi(G_{i,g_t}) = y_t)}{n} \\ or &= \frac{\sum_{i=1}^{n} \mathbb{I}(\Phi(v_{i,x_t}) = y_t)}{n}, \end{aligned}$$

where $\mathbb{I}$ is an indicator function.

To evaluate the attack evasiveness, we use *clean testing dataset accuracy drop (CAD)*, which is the classification accuracy difference of the original GNN Φ_o and the backdoored GNN Φ over the clean testing dataset.

Results for Graph Classification

This set of experiments evaluates the backdoor attack on GNNs for the graph classification task with the explainability results obtained from GNNExplainer. Based on the node importance matrix from GNNExplainer, we conduct three attacks with different trigger nodes selecting strategies, two among which are proposed here as new strategies: 1) **Random selecting attack strategy (RSAS)** - As in [164], in this strategy, we sample t nodes from the graph uniformly at random and replace their connection with that in the trigger graph. 2) **Most important nodes selecting attack strategy (MIAS)** - We choose the t most important nodes based on the node importance matrix and replace their connection as that of the trigger graph. 3) **Least important nodes selecting attack strategy (LIAS)** - Instead of selecting the most important t nodes, we select the least important nodes as the trigger nodes.

Table 3.2 presents the experimental results of the backdoor attack on the graph classification task based on three attacks. For each result in the table, the first value is ASR, and the second value is CAD. The results are conducted for the trigger size $\gamma = 0.2$. Finally, we include the performance of two different GNN models - GIN and GraphSAGE. We can observe that overall, all three backdoor attacks on GIN achieve high attack effectiveness (each with an attack success rate over 93%) and low clean accuracy drop (each with accuracy drop below 4%), while performances in GraphSAGE degrade with attack success rate up to 82%. This may be explained by GIN having a more powerful graph representation capability so the trigger graph can be learned better. The rank between these three attacks, except for the result of the facebook_ct1 dataset on GraphSAGE, is $LIAS \approx RSAS > MIAS$ in terms of attack effectiveness. The ASR of MIAS is lower than the other two attacks probably because after replacing the most important subgraph with the trigger graph, it is more difficult for the GNN to distinguish the graphs of the target class and non-target class. More precisely, GNN needs to dedicate more network capacity to learn specific patterns for each class sample, which negatively influences recognizing the trigger patterns. The result that ASR of RSAS and LIAS is close means the attacker can inject the trigger to the least important structure of the graph to achieve its goal of being less likely to be detected by the defender.

We also evaluate the impact of trigger size - γ fraction of the average number of nodes on the backdoor attack's performance. Figure 3.5 shows the attack performance under different trigger sizes varying from 5% to 20%. Obviously, the attack effectiveness of all attacks monotonically increases with the trigger size. This can be easily explained as with larger triggers, backdoored GNNs can better learn the difference between trigger-embedded and normal graphs. Additionally, the clean accuracy drop of all strategies slightly increases as well when the trigger size grows. This may be explained as the trigger size increases, more graph structure information has been modified, and the border between samples from different classes becomes less distinctive, so the performance of the backdoored GNNs on clean dataset drops.

This set of experiments takes on average 13.49min and 16.72min to implement back-

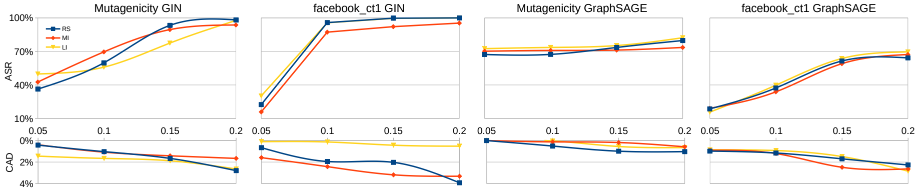


Figure 3.5: Impact of trigger size γ on the attack success rate (ASR) and clean accuracy drop (CAD) of backdoor attack on the graph classification task.

Table 3.2: Backdoor attack results on graph classification task based on different trigger nodes selecting strategies ($\gamma = 0.2$).

ASR(%) | CAD(%)	GIN			GraphSAGE		
	RSAS	MIAS	LIAS	RSAS	MIAS	LIAS
Mutagenicity	98.24 2.80	93.66 1.66	97.69 2.65	79.73 1.03	73.55 0.58	82.24 0.65
facebook ct1	100 3.93	95.35 3.32	100 0.52	64.23 2.27	67.22 2.64	69.57 2.85

door attacks on the GIN model on Mutagenicity and facebook_ct1 dataset, respectively. The GraphSAGE model takes around 13.37min and 16.35min on Mutagenicity and facebook_ct1 dataset, respectively. Clearly, the process of selecting the optimal trigger injecting position takes a short time on both GNN models on two datasets, e.g., 0.65s per graph on the GIN model on the Mutagenicity dataset. Consequently, utilizing the GNNExplainer method to select the optimal trigger injecting position for a backdoor attack on GNNs for graph classification task is practical and feasible. What is more, we can select the least important structure of the graph to evade the detection from a defender.

Results for Node Classification

Next, we evaluate the backdoor attack on GNNs for the node classification task under the explainability results of GraphLIME.

Based on the feature importance results of GraphLIME, we propose three attacks with different trigger feature dimension selecting strategies: 1) **Random selecting attack (RSAS)** - select n features from the node feature vector uniformly at random. 2) **Most important features selecting attack strategy (MIAS)** - select the most n representative features and change their values. 3) **Least important features selecting attack strategy (LIAS)** - select the least n representative features and change their values.

Table 3.3 summarizes the attack performance under different trigger feature dimension selecting strategies (the first value is the ASR, and the second value is the CAD). Observe that these three backdoor attacks on GAT obtain high attack success rate, i.e., over 84% and 95% for Cora and CiteSeer, respectively, and low clean accuracy drop at the same time. This is similar to the graph classification task results: the ASR of LIAS is close to RSAS, while MIAS has slightly degraded performance compared to the other two attacks. Therefore, the attacker can select the least representative features of a node to inject the feature trigger to achieve a high attack success rate, and the trigger-embedded node has a lower probability of being detected by the defender.

The running time for backdoor attack implementation on the GAT model on two-node classification datasets is on average 27.30s and 59.02s, respectively. In the process

Table 3.3: Backdoor attack results on node classification task based on different trigger features selecting strategies.

ASR(%) | CAD(%)	GAT		
	RSAS	MIAS	LIAS
Cora	86.01 2.23	84.11 0.66	84.22 1.95
CiteSeer	96.35 1.72	95.28 1.39	96.26 1.72

of selecting the optimal trigger injecting position, it only takes 0.06s and 0.09s per node for Cora and CiteSeer dataset, respectively. Similar to the conclusion of graph classification experiments, it is feasible to apply the explainability approach - GraphLIME to select trigger injecting position for a backdoor attack on the node classification task.

3.3. Rethinking the Trigger-injecting Position

Although in Section 3.2, we explore the impact of the backdoor trigger-injecting positions on GNNs with explanation techniques, we only provide the hypothesis about the results, and no experimental analysis is given to confirm the hypothesis. In this section, we give an empirical analysis of the attack results, which leads to a further understanding of the backdoor attack behavior in GNNs. In this section, we focus on the GNNExplainer [155] method as it can explain predictions of any GNN on any graph-based machine learning task without requiring modification of the underlying GNN architecture or re-training.

With the thriving development of explainability techniques in machine learning, the attacker can use model explanations to gain knowledge about the model to perform the adversarial attacks [93]. Kuppa et al. [70] used counterfactual explanations to find the malware features that most heavily impact the classifier decision. They used this knowledge to craft adversarial training samples that efficiently poison the model. Severi et al. [111] used SHAP to craft backdoor triggers in malware detectors. Utilizing the explanation, they determined which features to poison, resulting in a success rate of up to three times higher than that of a greedy algorithm that does not use explainable artificial intelligence (XAI). While there has been an increasing number of studies on utilizing explanation techniques to implement backdoor attacks in deep learning models, there has been no research on using explanation tools to clarify the backdoor attack behavior in the graph domain.

3.3.1. Methodology

Threat Model

We consider a *gray-box* threat model assuming the attacker can freely modify a small portion of the training dataset. Since the explanation masks in GNNExplainer are generated through gradients of the GNN model, the attacker also has knowledge of the gradient information of the target model on the chosen training dataset. We also assume the attacker performs a *dirty-label* backdoor attack, where the poisoned samples' labels are changed to the target label. Although this kind of attack is weaker than *clean-label* backdoor attacks [124], where the labels remain unaltered, dirty label attacks are the most common in the literature [143, 164]. The attacker's goal is to inject a backdoor in the given pre-trained clean GNN model through training over the poisoned training dataset, which achieves misclassification under the presence of a trigger while maintaining clean high accuracy on the original task. This threat model is realistic in real-world settings. For example, if the training dataset is collected from public users, the adversary can provide trigger-embedded training data to implement the backdoor attack.

General Framework

As stated before, we aim to discover if and how the explainability techniques in GNNs help improve the performance of backdoor attacks. Here, we focus on utilizing the feature-trigger backdoor attack from Section 3.2 for the node classification task. The trigger used in the backdoor attacks in our paper is defined as:

Definition 1 (Trigger) *In our backdoor attacks, the trigger is a specific feature pattern that is created by modifying the value of a subset of a node's features.*

Generally, two steps are conducted to implement backdoor attacks using explainability techniques:

(1) We apply an explainability technique (i.e., GNNExplainer) on a pre-trained clean GNN model to implement backdoor attacks based on two trigger-injecting strategies defined below.

Definition 2 (The Most/Least Representative Features) *Through applying the GNNExplainer on the pre-trained clean GNN model on the target node, we can obtain the original importance order of the node features. Based on the importance order information, we can locate the most or least representative features.*

Definition 3 (Most Important Area Strategy (MIAS)) *We select the most representative features of the target node and inject the feature trigger into the corresponding dimensions.*

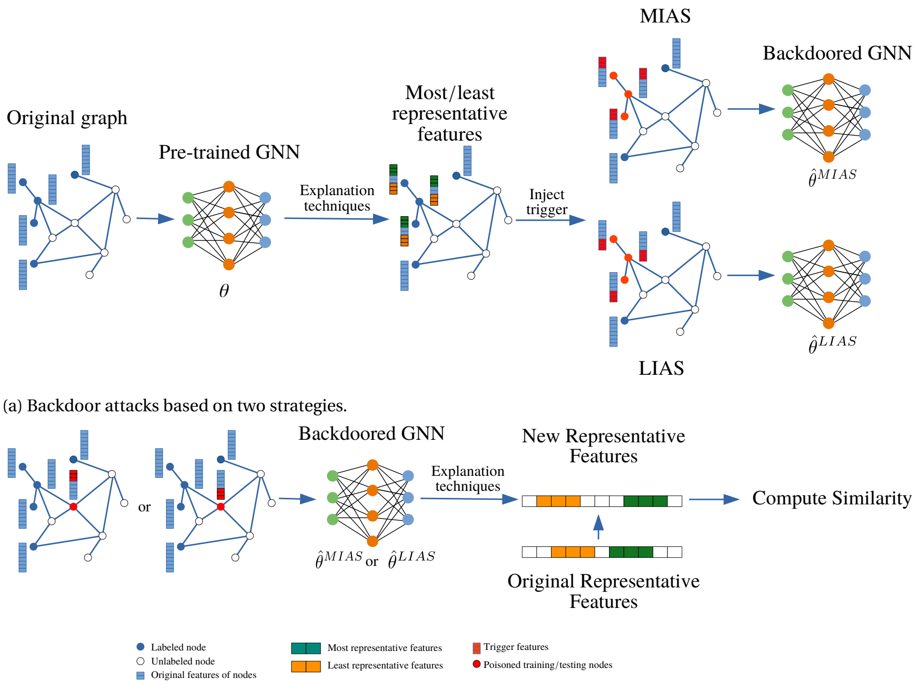
Definition 4 (Least Important Area Strategy (LIAS)) *We select the least representative features of the target node and inject the feature trigger into the corresponding dimensions.*

We then compare the attack performance based on these two strategies, including the attack success rate and clean accuracy drop.

(2) Next, we try to explain the attack performance of these two strategies by again applying the explainability techniques on the backdoored model over the poisoned testing dataset. As a result, we can obtain the new importance order of the node features, which is used to compute the similarity with the original feature importance order. The proposed framework is presented in Fig. 3.6.

Explanation Design

The detailed process of generating poisoned training dataset and target masks is presented in Algorithm 1. $EXP(\cdot)$ is the applied GNN explanation technique, i.e., GNNExplainer, and s is the trigger-injecting strategies, i.e., MIAS or LIAS. The algorithm first samples a subset from the original training dataset with a poisoning rate r (line 2). For each sampled node, the algorithm will compute the corresponding feature order to determine the trigger-injecting location for MIAS and LIAS. Meanwhile, the label of the


MIAS
Backdoored GNN
Most/least representative features
Original graph
Pre-trained GNN
Explanation techniques
Inject trigger
θ
$\hat{\theta}^{MIAS}$
$\hat{\theta}^{LIAS}$
LIAS
Backdoored GNN
New Representative Features
or
Explanation techniques
Compute Similarity
$\hat{\theta}^{MIAS}$ or $\hat{\theta}^{LIAS}$
Original Representative Features
Labeled node
Unlabeled node
Original features of nodes
Most representative features
Least representative features
Trigger features
Poisoned training/testing nodes

(a) Backdoor attacks based on two strategies.

(b) Procedure of explaining two attack strategies.

Figure 3.6: An illustration of backdoor attack and explanation framework.

poisoned training dataset will change to the target label. The trigger size n is the number of the features in the feature trigger, which means n node features will be modified. The poisoned testing dataset is obtained by injecting a trigger (following the same strategy as the poisoned training dataset) into the samples and changing their labels to the target label. Finally, based on the order of representative features, we can generate a target mask for each node in the poisoned testing dataset (line 17). The target mask has the same shape as the node feature vector, and the most (least) n important features are masked in while other features are masked out. To evaluate whether the backdoored model can recognize the trigger pattern precisely, the number of features to be masked in is set to be n. The definition of the target mask is as follows:

Definition 5 (Target Mask) *The target mask is a boolean tensor that indicates which n features contribute more to the final prediction from the pre-trained clean model θ for the target node compared to other features.*

Once the poisoned training dataset is generated, we can obtain the backdoored models $\hat{\theta}^s$ by retraining the clean model θ with the backdoored training dataset. [1] The process of training the backdoored models and obtaining predicted masks is shown in Al-

[1] In this work, we combine the original training dataset and the poisoned training dataset as the backdoored training dataset.

Algorithm 1: Generate Poisoned Training Dataset and Target Masks

```
Input: Pre-trained clean GNN model $\theta$, Training set $D_{train}$, Testing set $D_{test}$,
       Trigger-injecting strategy $s \in \{MIAS, LIAS\}$, Target label $y_t \in [0, C)$
Output: Poisoned training dataset $\hat{D}^s_{train}$, Poisoned testing dataset $\hat{D}^s_{test}$, Target masks
        $M^s_t$
/* Sampling Training Dataset to Inject Trigger */
$\hat{D}^s_{train} \leftarrow sample(D_{train}, r, y \neq y_t)$
foreach $\{x, y\} \in \hat{D}^s_{train}$ do
    /* Computing Order of Representative Features */
    $feature\_order = EXP(\theta, x, y)$
    $\hat{x}^s = Inject\_Trigger(x, feature\_order, s)$
    $\hat{y}^s = y_t$
end
$\hat{D}^s_{test} \leftarrow D_{test}[\backslash y_t]$
$M^s_t \leftarrow \emptyset$
foreach $\{x, y\} \in \hat{D}^s_{test}$ do
    /* Computing Order of Representative Features */
    $feature\_order = EXP(\theta, x, y)$
    $\hat{x}^s = Inject\_Trigger(x, feature\_order, s)$
    $\hat{y}^s = y_t$
    /* Generating Target Mask */
    $m^s_i = Get\_Mask(feature\_order, s)$
    $M^s_t = M^s_t \cup m^s_i$
end
return $\hat{D}^s_{train}, \hat{D}^s_{test}, M^s_t$
```

Algorithm 2: Train Backdoored GNN Models and Generate Predicted Masks

```
Input: Pre-trained clean GNN model $\theta$, Training set $D_{train}$, Poisoned training dataset
       $\hat{D}_{train}$, Poisoned testing dataset $\hat{D}_{test}$, Trigger-injecting strategy $s \in \{MIAS, LIAS\}$
Output: Backdoored GNN model $\hat{\theta}^s$, Predicted Masks $M^s_p$
/* Training Backdoored Models */
/* $\{x, y\} \in D_{train}$, $\{\hat{x}^s, \hat{y}^s\} \in \hat{D}^s_{train}$ */
$\hat{\theta}^s = \mathrm{argmin}_\theta (\sum_i L(x_i, y_i; \theta) + \sum_i L(\hat{x}^s_i, \hat{y}^s_i; \theta))$
$M^s_p \leftarrow \emptyset$
foreach $\{\hat{x}^s, \hat{y}^s\} \in \hat{D}^s_{test}$ do
    /* Getting Predictive Mask */
    $feature\_order = EXP(\hat{\theta}^s, \hat{x}^s, \hat{y}^s)$
    $m^s_i = Get\_Mask(feature\_order)$
    $M^s_p = M^s_p \cup m^s_i$
end
return $\hat{\theta}^s, M^s_p$
```

gorithm 2. To analyze the impact of injecting trigger into the most/least important part of the node features on the attack performance, we compare the attack performance of $\hat{\theta}^{MIAS}$ and $\hat{\theta}^{LIAS}$, including the attack success rate and clean accuracy drop. Finally, for the poisoned testing dataset, which we used to calculate the attack success rate, we again utilize the GNNExplainer to obtain the new feature importance order for each node on the backdoored GNN model $\hat{\theta}^{MIAS}$ or $\hat{\theta}^{LIAS}$ (line 7). The new feature importance order is used to generate the predicted mask. The definition of the predicted mask is as follows:

Definition 6 (Predicted Mask) *The predicted mask is a boolean tensor which indicates which n features contribute more to the final prediction from the backdoored GNN model $\hat{\theta}$ for the target node compared to other features.*

Combining the target masks we get in Algorithm 1, we can compute the similarity between the ordering of the new representative features and the old ones by calculating the recall score of the target mask and the predicted mask:

$$\begin{aligned} RS_i^s &= \frac{TP(M_{t,i}^s, M_{p,i}^s)}{TP(M_{t,i}^s, M_{p,i}^s) + FN(M_{t,i}^s, M_{p,i}^s)}, i \in N \\ M_{t,i}^s(M_{p,i}^s) &= [0, \cdots, 1, \cdots, 1, \cdots, 0], \end{aligned} \tag{3.3}$$

where RS_i^s is the recall score of the ith poisoned testing sample with s strategy, $M_{t,i}^s$ and $M_{p,i}^s$ is the target mask, and predictive mask of the ith poisoned testing sample, TP and FN is the true positive and false negative rate of these two masks, respectively, and N is the number of the poisoned testing dataset. We assume that higher similarity indicates that the backdoored model can better recognize the trigger pattern, contributing to better attack performance.

3.3.2. Experimental Results

Experimental Setting

We implemented the backdoor attack on the node classification task using the PyTorch framework. All experiments were run on a server with 2 Intel Xeon CPUs, 1 NVIDIA 1080 Ti GPU with 32GB RAM, and Ubuntu 20.04 LTS OS. Each experiment was repeated 10 times to obtain the average result.

Dataset. For our experiments, we use two publicly available real-world datasets for the node classification task: Cora [110] and CiteSeer [110]. These two datasets are citation networks in which each publication is described by a binary-valued word vector indicating the absence/presence of the corresponding word in the collection of $1,433$ and $3,703$ unique words, respectively. For each node classification dataset, we split 20% of the total nodes as the original training dataset (labeled), and the rest of the nodes are treated as the original testing dataset. To generate the backdoored training dataset, we sample 10% of the original training dataset to inject the feature trigger and relabel these nodes with the target label. The trigger size is set to 5% of the total number of node feature dimensions. We set these parameters as they provided the best results after conducting a tuning phase.

Models and training. We use the popular GAT [127] and GCN [67] models, as these two methods are commonly-used GNN models for the node classification task. We train the clean and backdoored GNN models with a learning rate of 0.005 and use Adam as the optimizer.

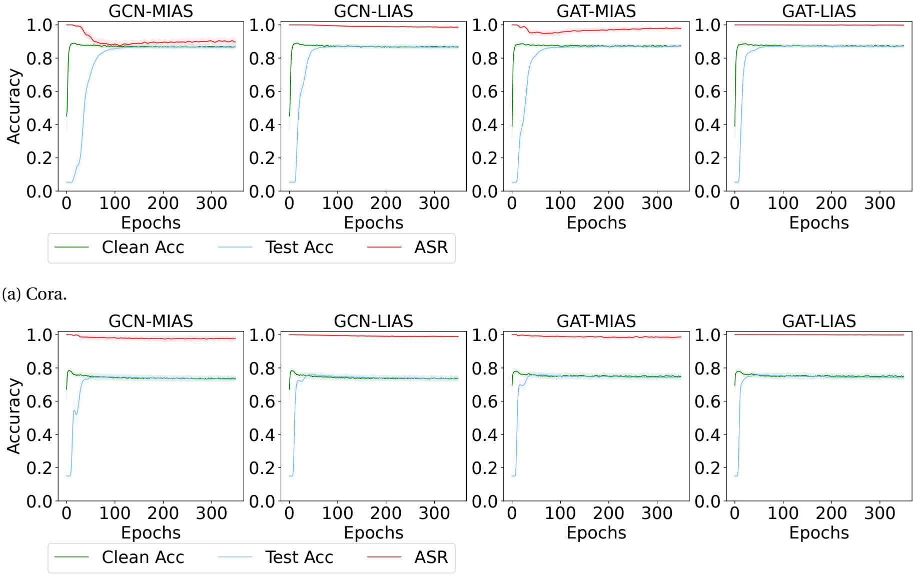
Attack evaluation metrics. To compare the attack performance of MIAS and LIAS, we utilize two commonly used backdoor attack evaluation metrics:

1. **Attacks Success Rate** (ASR): measures the backdoor performance of the model on a fully poisoned dataset $\hat{D}$. It is computed as $ASR = \frac{\sum_{i=1}^{N} \mathbb{I}(\hat{\theta}(\hat{x}_i)=y_t)}{N}$ where $\hat{\theta}$ is the poisoned model, $\hat{x}_i$ is a poisoned input, $\hat{x}_i \in \hat{D}$, y_t is the target class, and $\mathbb{I}$ is an indicator function.
2. **Clean Accuracy Drop** (CAD): measures the effect of the backdoor attack on the original task. It is calculated by comparing the performance of the poisoned and clean models on a clean holdout testing set. The accuracy drop should generally be small to keep the attack stealthy.

Results and Analysis

Results. The backdoor attack results on two graph datasets based on two models and two trigger-injecting strategies are shown in Fig. 3.7. In particular, the ASR and CAD of two GNN models on two datasets are presented in Table 3.4. We can observe that both strategies can achieve a high attack success rate, i.e., more than 97%, except GCN on the Cora dataset with MIAS. In addition, in most cases, the ASR of LIAS is slightly higher, around 1%, than that of MIAS. However, for the GCN model on the Cora dataset, the ASR of LIAS is significantly higher: more than 8%, than the MIAS. We can also see that the CAD for all datasets and models is unnoticeable, and the difference between the two strategies over CAD is negligible.

Analysis. Next, we investigate the reason why the backdoor attack performance of the LIAS is somewhat higher or significantly higher (for the GCN model on the Cora dataset) than the MIAS. As mentioned in Section 3.3.1, we evaluate the similarity between the ordering of the new representative features and the old ones by calculating the recall score of the target mask and the predicted mask. The histogram of recall scores over the poisoned testing dataset of all datasets and models is shown in Fig. 3.8. We can observe that most poisoned testing samples have a recall score of more than 0.5 in both MIAS and LIAS, which results in a high attack success rate for both strategies. To further investigate the slight advantage of the LIAS over the MIAS, we split the poisoned testing samples into two parts, one is misclassified into the target class successfully, and the other one is not, and compute the recall scores for these two parts, as shown in Fig. 3.9. We notice that, generally, the successfully misclassified nodes have significantly higher recall scores than those not misclassified into the target class. This phenomenon is consistent with the assumption mentioned in Section 3.3.1, i.e., the higher similarity between the



(a) Cora.

(b) CiteSeer.

Figure 3.7: Backdoor attack results of two trigger-injecting strategies.

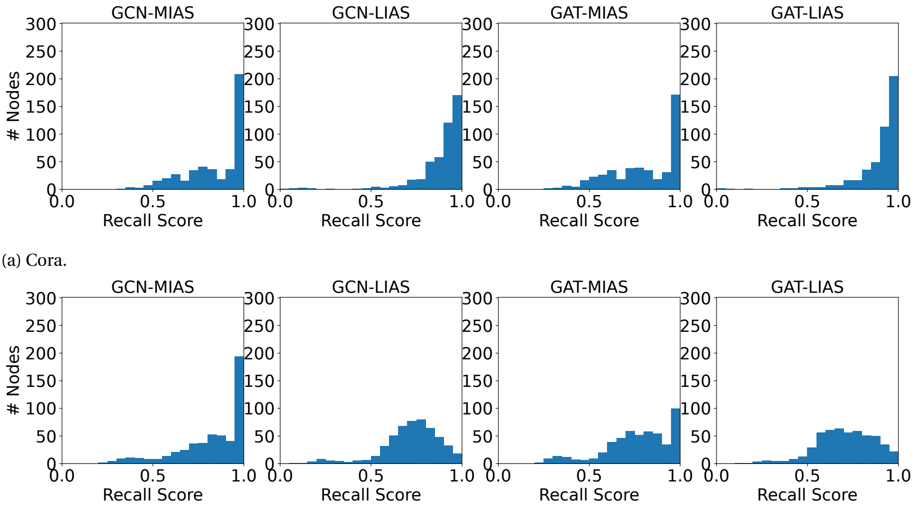
Table 3.4: Backdoor attack performance of MIAS and LIAS (SD: standard deviation).

	MIAS			
Dataset	GCN		GAT	
	ASR ± SD	CAD ± SD	ASR ± SD	CAD ± SD
Cora	90.08% ± 0.29%	0.32% ± 0.19%	97.91% ± 0.12%	0.34% ± 0.24%
CiteSeer	97.70% ± 0.10%	0.32% ± 0.17%	98.54% ± 0.09%	0.71% ± 0.20%
	LIAS			
Dataset	GCN		GAT	
	ASR ± SD	CAD ± SD	ASR ± SD	CAD ± SD
Cora	98.65% ± 0.06%	0.27% ± 0.21%	99.89% ± 0.03%	0.27% ± 0.21%
CiteSeer	98.96% ± 0.07%	0.15% ± 0.18%	99.88% ± 0.03%	0.80% ± 0.17%

ordering of the new representative features and that of the original ones indicates that the backdoored model can recognize the trigger pattern better. When comparing the second column and the last column of Fig. 3.9b, 3.9c, and 3.9d, we also see that LIAS has fewer nodes with low recall score than MIAS, which we believe is the reason of higher ASR of LIAS than MIAS.

In contrast, we surprisingly see that for the GCN model on the Cora dataset with

MIAS, the unsuccessfully misclassified nodes also have a high recall score as the successfully misclassified nodes. We assume that the main reason behind this is that, under the MIAS, the feature trigger is injected into the positions of the most representative features. Thus the backdoored model will recognize not only the trigger pattern but also the representative feature pattern for the original label. Therefore, for MIAS, it is possible that even the poisoned testing samples that are not successfully misclassified into the target class will have a high recall score. We verify this hypothesis by extending the target masks and predicted masks twice the feature trigger length, i.e., $2 * n$, and computing the recall scores again. [2] The histogram of the new recall scores of the GCN model on the Cora dataset is shown in Fig. 3.10. We also checked the prediction of the backdoored model over the unsuccessfully misclassified nodes. The output indicates that all these nodes are classified into their original classes. Comparing Fig. 3.9a and 3.10, we observe that the recall scores of the successfully misclassified nodes generally reduce to half of that without extended masks. We believe this is because, for these nodes, the backdoored model recognizes the trigger location exactly, and when we extended the masks twice the trigger length, only half of the features can be recalled. However, we can also see that for the MIAS, the recall scores of the unsuccessfully classified nodes are still as high as those without the extended masks. This is because the backdoored model recognizes the feature pattern for the original label (that is why these nodes are classified into the original class and the attack is not successful), so even if the masks are extended, the recall score is still high.

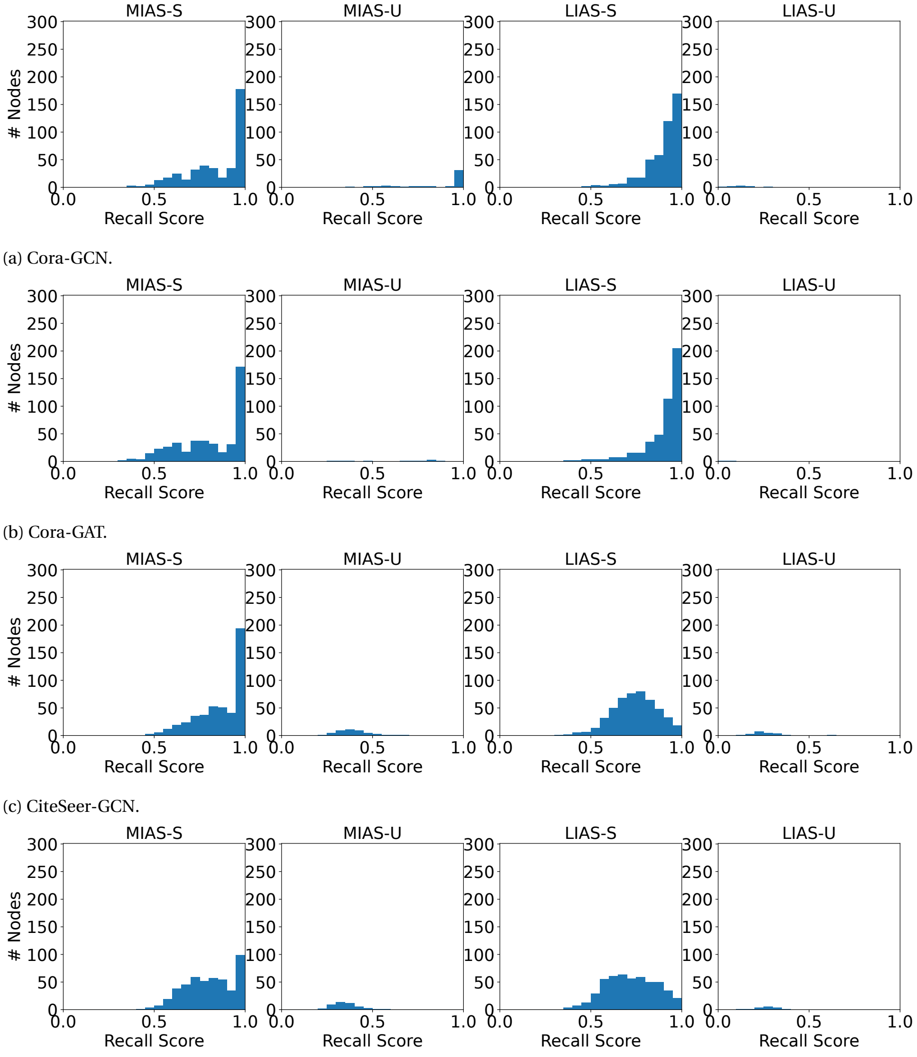


(a) Cora.

(b) CiteSeer.

Figure 3.8: Histogram of recall scores over the poisoned testing dataset.

[2] Here, we select an extension rate of 2. To verify the hypothesis, the extension rate can be set to $\gamma > 1$, and the recall scores of the successfully misclassified nodes are expected to reduce to $1/\gamma$ of that without extended masks.



(a) Cora-GCN.

(b) Cora-GAT.

(c) CiteSeer-GCN.

(d) CiteSeer-GAT.

Figure 3.9: Histogram of recall scores over two parts of the poisoned testing dataset (S means the nodes are successfully misclassified into the target label, U means not).

3.4. Clean-label Backdoor Attacks

3.4.1. Methodology

Problem Formulation

Given a pre-trained GNN model Φ_o and its training dataset $D_{train} = \{(G_1, y_1), (G_2, y_2), \ldots, (G_n, y_n)\}$ where G_i and y_i respectively are the i-th training graph and its true label, the clean-label

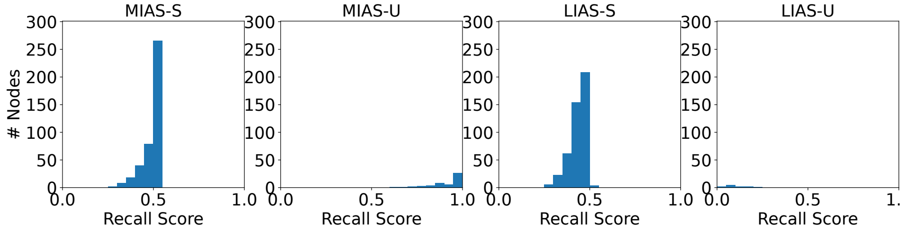


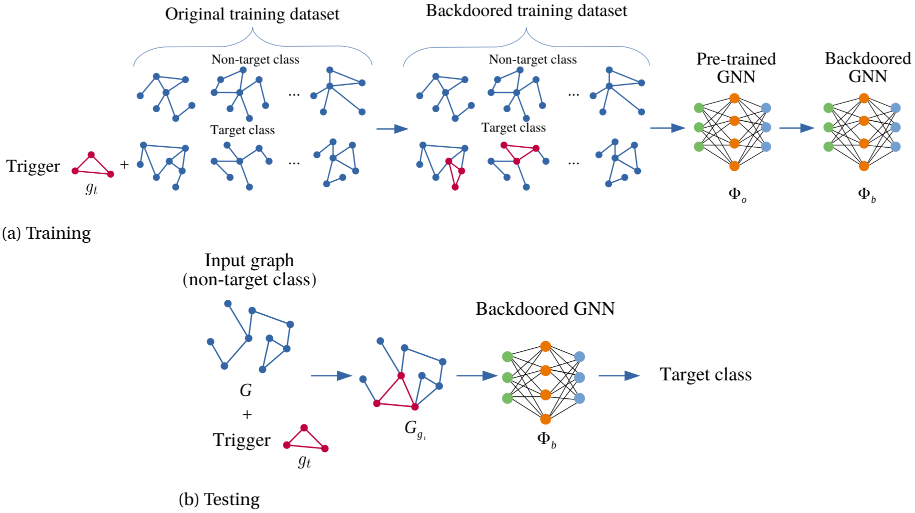
Figure 3.10: Histogram of recall scores of GCN model on Cora dataset with extended masks (S means the nodes are successfully misclassified into the target label, U means not).

backdoor attack aims to forge a backdoored GNN Φ_b that will misclassify the testing sample with a specific trigger into pre-determined labels (i.e., target label y_t) without affecting the performance on clean data. We assume the attacker can access the training dataset D_{train}. Unlike prior works [164, 143], which perform backdoor attacks on GNNs by injecting a trigger into a sampled training dataset and changing their labels to the target label, the attacker of clean-label backdoor attack samples a subset of training dataset with target label and injects trigger into them without changing their labels. Thus, the poisoned samples have plausible labels.

General Framework

The general framework of a clean-label backdoor attack on GNNs is shown in Figure 3.11. In the training phase, as presented in Figure 3.11a, the attacker samples data from the original training dataset in the target class and injects a specific trigger to generate poisoned samples. The resulting poisoned samples are then utilized to backdoor the pre-trained GNN model Φ_o to get the backdoored GNN model Φ_b. Here, we focus on the subgraph-based backdoor attacks on GNNs for the graph classification task since most graph classification tasks are implemented in GNNs by learning the network structure. The backdoored GNN model is assumed to predict any testing sample (which can be from an untarget class) with a specific trigger into the target class, as shown in Figure 3.11b.

Specifically, the implementation details of our attack are described in Algorithm 3. The key point is backdoored dataset generation. Here, we adopt the Erdős-Rényi (ER) model [44] to generate trigger g_t (line 3 in Algorithm 3) as it is fast and more effective than other random graph generation methods [164]. In particular, this model outputs a random graph of s nodes, and the probability of an edge between each pair of nodes in this graph is ρ. We sample subsets of the original training dataset (with target label) with proportion r, as shown as D_{tmp}, and the remaining is saved as clean training dataset D_{clean}. For each sampled graph, we inject a trigger (by the ER model) into it by sampling s nodes from the graph uniformly at random and replacing their connection with that in the trigger graph. Under the setting of a clean-label backdoor attack, the attacker does not re-label the sampled data. The backdoored dataset comprises the dataset with trigger $D_{trigger}$ and the remaining clean training dataset D_{clean}.



(a) Training

(b) Testing

Figure 3.11: Clean-label backdoor attack framework.

3.4.2. Experimental Results

Datasets. We perform experiments with two publicly available datasets: (1) MUTAG [33] - structure graphs of the mutagenic and non-mutagenic molecules and (2) NCI1 [115] - chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines. For each dataset, we randomly sample 80% of the data instances as the training dataset and the rest as the testing dataset.

Target Models. We choose GCN [67] and GIN [146] as our target models to be attacked, considering their excellent performance and widespread adoption [141, 142].

Evaluation. We use the *attack success rate (ASR)* to evaluate the attack effectiveness. Specifically, we embed each testing data with the specific trigger graph and calculate the ASR of the backdoored GNN model on the poisoned testing dataset. Here, we only embed the testing dataset of the non-target label with a trigger to avoid the influence of the original label. The ASR measures the proportion of trigger-embedded inputs misclassified by the backdoored GNN into the target class y_t chosen by the adversary. The trigger-embedded inputs are

$$D_{trigger} = \left\{(G_{1,g_t}, y_1), (G_{2,g_t}, y_2), \ldots, (G_{n,g_t}, y_n)\right\}.$$

Here, g_t is the graph trigger, $\left\{G_{1,g_t}, G_{2,g_t} \ldots, G_{n,g_t}\right\}$ is the testing dataset embedded with g_t, and $y_1, y_2, \ldots, y_n$ is the label set.

Formally, ASR is defined as:

$$Attack\,Success\,Rate = \frac{\sum_{i=1}^{n} \mathbb{I}(\Phi_b(G_{i,g_t}) = y_t)}{n},$$

Algorithm 3: Clean-label backdoor attack for the graph classification task

```
Input: Pre-trained GNN model $\Phi_o$, Training set $D_{train} = \{x_i, y_i\}_{i=1}^{S}$, Target label
       $y_t \in [0, C)$
Output: Backdoored GNN model $\Phi_b$
Function CLEAN_LABEL_BACKDOOR_ATTACK():
    $D_{trigger} \leftarrow \emptyset$
    $g_t \leftarrow ER(s, \rho)$
    $D_{tmp} \leftarrow sample(D_{train}, r, y = y_t)$
    $D_{clean} = \{data \in D_{train} : data \notin D_{tmp}\}$
    foreach $d \in D_{tmp}$ do
        $d[x] = INJECT(d[x], g_t)$
        $D_{trigger} = D_{trigger} \cup \{d[x], d[y]\}$
    end
End Function
$D_{backdoor} = D_{clean} \bigcup D_{trigger}$
$\Phi_b = Train(\Phi_o, D_{backdoor})$
return $\Phi_b$
```

where $\mathbb{I}$ is an indicator function and Φ_b refers to the backdoored GNN model.

Furthermore, we also use *clean accuracy drop (CAD)* to evaluate the attack evasiveness. CAD indicates the classification accuracy difference between the original GNN model Φ_o and the backdoored GNN model Φ_b over the clean testing dataset.

Results. We set the number of nodes in the trigger graph to be 20% of the average number of nodes in the dataset. Next, we set the poisoning rate r and edge existing probability ρ to be 10% and 80%, respectively. We set those parameters following the setting in prior backdoor attacks on GNNs [164]. Table 3.5 presents the attack results. We can observe that overall, a clean-label backdoor attack can achieve high attack effectiveness for both datasets and models, i.e., with an attack success rate over 84%, especially for the NCI1 dataset with up to 98.47% ASR. It can also be observed that in most cases, a clean-label backdoor attack has a low CAD, i.e., around 1%, which indicates that a clean-label backdoor attack has a negligible impact on the original task of the model.

The results in [124] indicate that only poisoning inputs of the target class (i.e., without changing the true labels) renders the attack ineffective. The authors argued the main reason for the attack's ineffectiveness is that the poisoned samples can be correctly classified by learning a standard classifier, so the backdoor attack is unlikely to be successful since relying on the backdoor trigger is not necessary to classify these inputs correctly. On the contrary, here, we find that without any other improvement, the clean-label backdoor attack is already successful on GNN models. This may be explained since the GNN model predicts the input graph by learning information of specific structure(s), i.e., explanation subgraph(s) [155], in the graph. If a graph is injected with a trigger graph, the GNN model will also try to learn the trigger pattern and add it to the explanation subgraphs. Once the backdoored GNN model is trained, it will output a target label if one of the explanation subgraphs, e.g., trigger graph, appears in the input graph.

Table 3.5: Attack performance (SD: standard deviation).

Dataset	ASR(%) | CAD(%) Mean (SD)	
	GCN	GIN
MUTAG	87.83(1.03) | 0.16(0.03)	84.86(1.68) | 2.24(0.15)
NCI1	98.47(1.30) | 0.88(0.01)	97.62(2.03) | 1.01(0.07)

3.5. CONCLUSIONS

This chapter designs and explores backdoor attacks in centralized GNNs. First, we take the first step toward the explainability of the impact of trigger injecting position on the performance of backdoor attacks on GNNs. We conduct research on two graph tasks - graph classification and node classification. For the graph classification task, we apply GNNExplainer to select the optimal subgraph in a graph to be replaced by the trigger graph. For the node classification task, we propose a new backdoor attack using a subset of node features as a trigger pattern, and then we apply GraphLIME to choose the optimal subset of node features to change their values to a fixed value as the feature trigger. Through empirical evaluation using benchmark datasets and state-of-the-art models, we verify that our approach can quickly select the optimal trigger-injecting position to implement a powerful backdoor attack on GNNs. Furthermore, we see that the attacker can select the least important parts of the graph to inject the trigger, thus reducing the chances of easy detection by the defender.

Next, we conduct a comprehensive analysis and explanation of graph backdoor attacks with two trigger-injecting strategies, i.e., MIAS and LIAS, proposed above. We investigate the node classification task and compare the attack performance for these two strategies. Our findings show that LIAS always achieves higher attack performance than MIAS. We further explain the difference with quantitative analysis, which contributes to a further understanding of the backdoor attack behavior in GNNs. Finally, in order to implement a more insidious backdoor attack, we design a new backdoor attack on GNNs that only poisons inputs of the target class without changing the true labels. Our method leverages the GNN model's redundant learning capability to learn the trigger pattern. Initial experimental evaluations showed that a clean-label backdoor attack could achieve a high attack success rate and low clean accuracy drop. We hope our study highlights the concern of clean-label backdoor attacks on GNNs, which are more insidious.

For future works, we aim to expand the technique of explaining backdoor attacks on GNNs for the node classification task to the graph classification task. More precisely, we would compute the similarity between the new representative subgraph and the old one by calculating the recall score of the target mask and the predicted mask. We also expect to explore the clean-label backdoor attack's effectiveness against filtering techniques mentioned in Section 3.1.

4

Backdoor Attacks on Federated GNNs

Previous studies have indicated that node-level GNNs are vulnerable to Membership Inference Attacks, which infer whether a node is in the training data of GNNs and leak the node's private information, like the patient's disease history. The implementation of previous MIAs takes advantage of the models' probability output, which is infeasible if GNNs only provide the prediction label (label-only) for the input. In addition to MIAs, there are also privacy concerns and regulation restrictions in GNNs when centrally trained, making it promising to apply federated learning to train GNNs. Although several research works have applied FL to train GNNs (Federated GNNs), there is no research on their robustness to backdoor attacks. Therefore, in this chapter, we aim to answer the second subquestion in Section 1.4. Specifically, we give solutions to the following research questions, i.e., Can we design a label-only membership inference attack on GNNs? and Is it possible to design backdoor attacks on Federated GNNs?

In this chapter, we propose a label-only MIA against GNNs for node classification. Our attacking method achieves around 60% accuracy, precision, and Area Under the Curve (AUC) for most datasets and GNN models. Also, this chapter conducts two types of backdoor attacks in Federated GNNs, i.e., CBA and DBA. We further explore the robustness of DBA and CBA against two state-of-the-art defenses. We find that both attacks are robust against the investigated defenses, necessitating the need to consider backdoor attacks in Federated GNNs as a novel threat that requires custom defenses.

This chapter is based on: 1) "Label-Only Membership Inference Attack against Node-Level Graph Neural Networks." *Conti, M., Li, J., Picek, S.,* ***Xu, J.*** *(2022). the 15th ACM Workshop on Artificial Intelligence and Security*, 2) "More is Better (Mostly): On the Backdoor Attacks in Federated Graph Neural Networks." ***Xu, J.****, Wang, R., Koffas, S., Liang, K., & Picek, S. (2022). The Annual Computer Security Applications Conference*, 3) "On Exploring Backdoor Attacks in Federated Graph Neural Networks." ***Xu, J.****, Koffas, S., & Picek, S. (2022). The Learning from Authoritative Security Experiment Results (LASER) workshop*, 4) "Unveiling the Threat: Investigating Distributed and Centralized Backdoor Attacks in Federated Graph Neural Networks." ***Xu, J.****, Koffas, S., & Picek, S.*

4.1. Introduction

Graph Neural Networks (GNNs) are gaining more and more attention for their broad application in analyzing social networks, recommender systems, and biological networks. Node classification, which predicts the label for nodes in the graph, is one of the popular tasks [41, 129]. Usually, GNNs are trained in centralized setting, which leads to privacy concerns, e.g., membership inference attack (MIA). MIA infers the existence of nodes in the training data of GNNs, which means the adversary can determine which nodes belong to the training data of GNNs with the implementation of MIAs. It is crucial to study MIAs as they can cause privacy leakage. For instance, we train a GNN model whose training data includes patients infected with COVID-19 and the goal of the GNN model is to investigate, recognize, and classify the infection factors of COVID-19 [97]. By attacking this GNN model, the adversary can utilize the probability vector output of the model to predict whether one patient is in the training data or not, which is a severe privacy leakage for the patient. The original intuition of MIA is that the overfitting model will assign a higher probability value to the training data than the testing data. Hence, previous MIAs against GNNs utilize the prediction probability vector to act as attack features or compute metric values [137, 58, 97]. However, they are ineffective when the model only outputs the input's label, which is a label-only condition. Furthermore, previous label-only MIAs against CNNs and semantic segmentation models are not effective or challenging to implement while being transferred to GNNs. To improve the attack performance under the label-only condition, we propose our label-only MIA against GNNs, which is more efficient and straightforward than previous methods.

In addition to MIAs, GNNs also face other challenges when centrally trained because of privacy concerns, regulatory restrictions, and commercial competition. For example, the financial institution may utilize GNN as a fraud detection model, but they can only have transaction data of its registered users (no data of other users because of privacy concerns). Thus, the model is not effective for other users. Similarly, in a drug discovery industry that applies GNNs, pharmaceutical research institutions can dramatically benefit from other institutions' data, but they cannot disclose their private data for commercial reasons [55].

Federated Learning is a distributed learning paradigm that works on isolated data. In FL, clients can collaboratively train a shared global model under the orchestration of a central server while keeping the data decentralized [66, 91]. As such, FL is a promising solution for training GNNs over isolated graph data, and there are already some works utilizing FL to train GNNs [55, 161, 72], which we denote as *Federated GNNs*. Although FL has been successfully applied in diverse domains, e.g., computer vision [83, 81] or language processing [168, 54], there could be malicious clients among millions of clients, leading to various adversarial attacks [5, 42]. In particular, limited access to local clients' data due to privacy concerns or regulatory constraints may facilitate backdoor attacks on the global model trained in FL.

Backdoor attacks on FL have been recently studied [5, 10, 144]. However, these attacks are applied in federated learning on Euclidean data, e.g., images and words. The backdoor trigger generation methods and injecting position are different between graph

(2023). Digital Threats: Research and Practice (DTRAP) journal.

data and images/words, as mentioned in Section 3.2. In particular, in [144], the authors split a square-shaped trigger placed in the top left corner of an image into four parts so that four malicious clients use each part in their poisoned datasets. When the training ends, the adversary concatenates these parts to form a global trigger in the image's upper left corner that activates the backdoor. This is impossible in GNNs as the data is not Euclidean, and there is no position that we can exploit. Also, defenses like FoolsGold [43] filter out clients that use similar updates as malicious. This can be effective for Euclidean data that use parts of the trigger in similar positions but may not be effective in GNNs. Indeed, the graph data is not Euclidean, and different partial triggers vary the graph structure resulting in non-aligned updates. Additionally, intensive research has been conducted on backdoor attacks in GNNs [164, 143]. However, these works focus on GNN models in centralized training. In federated learning, the malicious updates will be weakened in the aggregation function. Finally, there can be more than one malicious client, while in centralized GNNs, there is only one client. Thus, we should expect different behavior of backdoor attacks in Federated GNNs. Then, it is crucial to investigate if existing countermeasures that have been tested mostly with Euclidean data are still effective for backdoor attacks in Federated GNNs to understand how to deploy trustworthy AI systems.

This chapter offers backdoor attack strategies in the Federated GNNs. First, in Section 4.2, we propose a label-only MIA against GNNs for node classification with the help of GNNs' flexible prediction mechanism, e.g., obtaining the prediction label of one node even when neighbors' information is unavailable. Our attacking method achieves around 60% accuracy, precision, and Area Under the Curve (AUC) for most datasets and GNN models, some of which are competitive or even better than state-of-the-art probability-based MIAs implemented under our environment and settings. Additionally, we analyze the influence of the sampling method, model selection approach, and overfitting level on the attack performance of our label-only MIA. All of those three factors have an impact on the attack performance. Then, we consider scenarios where assumptions about the adversary's additional dataset (shadow dataset) and extra information about the target model are relaxed. Even in those scenarios, our label-only MIA achieves a better attack performance in most cases.

Second, in order to explore backdoor attacks in GNNs under distributed settings, we conduct two types of backdoor attacks in Federated GNNs: centralized backdoor attacks (CBA) and distributed backdoor attacks (DBA), in Section 4.3. CBA is conducted by embedding the same global trigger during training for every malicious party, while DBA is conducted by decomposing a global trigger into separate local triggers and embedding them into the training datasets of different malicious parties, respectively. Our experiments show that the DBA attack success rate is higher than CBA in almost all evaluated cases. For CBA, the attack success rate of all local triggers is similar to the global trigger even if the training set of the adversarial party is embedded with the global trigger. To further explore the properties of two backdoor attacks in Federated GNNs, we evaluate the attack performance for a different number of clients, trigger sizes, poisoning intensities, and trigger densities.

4.2. Label-only MIA to GNNs

The MIA aims to distinguish the training (members) and testing data (non-members) of the target model. Under the context of the node classification, the MIA determines which nodes are in the training data. Generally, there are two types of attack strategies. One is to train a classifier, with which we can predict the possibility of being a member or non-member for a specific data point [107, 116]. The other is to directly or indirectly compute the metric about the data point [78, 24, 86]. Then, the attacker determines the data point as a member or non-member by comparing the metric value with a threshold. Here, we focus on the classifier-based MIA, and the attack features are properties of one data point (or node) for feeding into the attack model. The key of the classifier-based MIA is to acquire the dataset (also called attack dataset) for training the classifier to distinguish the training and testing data. Under the label-only condition, the adversary cannot obtain the prediction probability vector from the target model for the attack feature extraction. Usually, the adversary has access to a shadow dataset with the same distribution as the target dataset. With this shadow dataset, the adversary trains a shadow model to mimic the behavior of the target model. Even though the adversary can only query the target model, the adversary has full knowledge of the shadow model. To improve the effect of imitation, the adversary could relabel the shadow dataset with the target model and train a relabelled shadow model. Then, the adversary utilizes the prediction of the shadow model to construct the attack dataset. Finally, the attacker will train an attack model based on the attack dataset for attacking the target model, i.e., distinguishing the training and testing data of the target model. One previous label-only work (not on GNNs) acquires the prediction probability vector of the relabelled shadow model as the attack features of the target dataset, called the transfer attack [78]. Besides, Li et al. [78] and Choquette-Choo et al. [24] proposed to use the distance between the original data point with its closest adversarial example to implement label-only MIA. However, finding the adversarial example of a graph is difficult under the label-only condition [139].

Unlike previous methods, we observe a flexible prediction mechanism of GNNs, which means that we can obtain the prediction label of a specific node with or without its neighbors' features and connection information. Besides, inspired by previous data augmentation methods for obtaining features [24] or reconstructing the prediction probability vector [105], we utilize feature masking and step-by-step edge dropping to measure the stability of the prediction label. Specifically, we obtain the attack features of the attack model from three aspects. The first one is the fixed properties of the node, including the number of neighbors and the ground truth of the node. The second one is the prediction correctness of the target model while only feeding the node's features which are masked with different rates and values, into the GNN. The last one is feeding the GNN with the node's features, features of 1-hop neighbors, and connection information. Specifically, the edges between the node and 1-hop neighbors are dropped step by step, and the node's features are masked with different rates and values to get the prediction correctness of the node and its 1-hop neighbors. We obtain the attack features from those three aspects to train the label-only attack model.

4.2.1. Our Label-only MIA

GNNs for node classification tasks have two learning settings, i.e., inductive and transductive. Under the inductive setting, the GNN cannot access the testing nodes during training. In the transductive environment, the GNN has knowledge of the whole graph structure, including the connection information of the unlabelled testing nodes. In this section, we do not consider the transductive setting because in the transductive setting, the connection information of the testing data is also used for the model training, which may influence the performance of MIA since the goal of MIA is to distinguish the training and testing samples. He et al. [58] also only focused on the inductive setting. While predicting the label of a specific node in the graph, the GNN learns its features, neighbors' features, and connection information. However, the GNN could also predict the node's label with only the property of a node, which is called the *flexible prediction mechanism*. The flexible way of prediction benefits our label-only MIA, which only gets the label from the model.

Problem Formulation

The goal of the MIA is to determine whether the target node V_t belongs to the training data (member) or testing data (non-member) of the target model F_t. To implement the MIA, the adversary has some external knowledge E. We formulate our label-only MIA A as the following function:

$$A: V_t, F_t, E \rightarrow \{1, 0\}. \tag{4.1}$$

Here, "1" means V_t is in the training data, and "0" otherwise (i.e., it is a binary classification task). In the experiments, we train a binary classifier to solve this task.

Threat Model

The target model F_t is open to the adversary, which means the attacker can query it for the label of the target node V_t. Considering flexible prediction mechanism and practical situation, we limit the query information to the target node's feature, features of its 1-hop neighbors $N(V_t)$, and connection information $C(V_t)$. There are two reasons for selecting 1-hop neighbors. The first one is that it requires less information than 2-hop neighbors [58] while predicting the membership of one node. The second reason is that the information of the target node is limited under the label-only condition. We cannot get enough helpful distinguishable signals without neighbors' information. Therefore, we choose 1-hop neighbors. Besides, we expose the target node's true label Y_{V_t} to the adversary.

The adversary has some external knowledge E. A shadow dataset D_s, extracted from the same distribution as the target dataset D_t used for training and testing the target model F_t, is in the external knowledge. We relax this assumption and sample the shadow dataset from the other distribution in experiments. Furthermore, the adversary knows the target model's architecture, training hyperparameters and algorithm. With that knowledge, the attacker can train a shadow model F_s to mimic the behavior of the target model. Similarly, we relax the assumption that the adversary knows the GNN architecture and type of the target model within exploration. The relaxation of those two assumptions tests the effectiveness of our label-only MIA under stricter conditions and makes our MIAs closer to attack in the real world. He et al. [58] and Olatunji et al. [97] also relaxed those assumptions.

Attack Methodology

The adversary obtains some external knowledge for the MIA implementation. Then the question is how to apply our label-only MIA on GNNs with that knowledge and still satisfy the threat model. To answer this question, we formulate the final attack model as a binary classifier to predict whether the node is in the training data of the GNN or not. Therefore, the acquisition of the attack features for training the attack model is the crucial point.

General Steps. The overall process is illustrated in Figure 4.1. First, the total dataset is split into target and shadow datasets. With a shadow dataset and knowledge about the training of the target model, the adversary can train a shadow model to mimic the behavior of the target model. The shadow model and membership situation of nodes in the shadow dataset are transparent to the adversary. Thus, the attacker generates the attack dataset via data points' fixed properties and multiple queries for each data point to the shadow model, which only outputs the prediction label. The features of the data point in the attack dataset are attack features. If the node is in the shadow model's training data, the adversary assigns its attack features with the label "1" and otherwise "0". Then, the adversary trains the attack model with the attack dataset extracted from the shadow dataset and model.

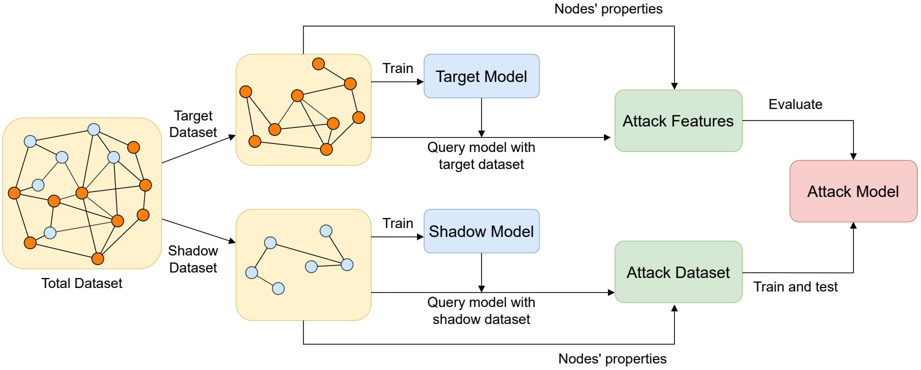


Figure 4.1: The general process of our label-only MIA.

When the attack model is trained completely, the adversary can querys the target model with target nodes from the target dataset to get each target node's attack features. Afterward, the attacker feeds the target nodes' attack features into the attack model to get the output which is a value between 0 and 1, indicating the probability of the target node being from training data. Notably, the acquisition of the attack features is interpreted in the next section.

Attack Features Acquisition. Under the label-only condition, the adversary only gets the prediction label of the target node from the target model. Therefore, obtaining the attack features of target nodes in the target dataset is crucial. Previous label-only MIAs [78] (not on GNNs) extract attack features of the target node from the prediction probability vector of relabelled shadow models, which are under the adversary's control. The relabelled shadow model utilizes the target model to relabel the shadow dataset, which is different from the shadow model. One previous method feeds the target node into relabelled shadow models to get the prediction probability vector as the attack features of this node under the label-only condition because the target model does not expose the prediction probability vector. For comparison, we present the result of this transfer attack under the label-only condition in Table 4.3. Unlike previous methods, we construct target nodes' attack features based on fixed properties of the node, data augmentation, flexible prediction mechanism, and practical situation. The application of data augmentation obtains inspiration from previous works [24, 35]. We consider the attack features from three aspects: fixed properties, 0-hop query, and 1-hop query. The specified properties, like the ground truth of the target node in the target model, act as a part of features in previous attack models [116, 62]. Besides, the adversary has only access to 1-hop neighbors, which means the attacker could query the target model with (1-hop query) or without (0-hop query) information of 1-hop neighbors. Therefore, we extract features from the three aforementioned aspects.

Acquisition with fixed properties. Olatunji et al. [97] mentioned that the connection between nodes increases the vulnerability of GNN models to privacy attacks. Inspired by this, we count the number of neighbors (n_num) and signal (w_i_node), which indicates whether the node is independent of other nodes, as part of the attack features of the target node V_t. Notably, the neighbors could be in the training or testing data of the dataset while counting the number of neighbors in the target or shadow dataset. Besides, we include the target node's ground truth (o_label) into the attack features, which is the same as the previous MIA [116].

Acquisition with the 0-hop query. The prediction of GNNs is flexible, which means we can get the prediction label of a node with or without the features and the connection information of its neighbors fed into GNNs. For each target node V_t, we randomly change different percentages of its feature values X_{V_t} to the largest and smallest value in the feature space. Then we only input the target node with the changed feature to the target model and record whether the prediction is the same as the ground truth of the target node. Here, $i\_none\_max\_rate$ and $i\_none\_min\_rate$ are the record results for each *rate* in $rate\_set$. The "1" means the prediction of the node with the changed feature is the same as the ground truth, and the "0" means otherwise.

Acquisition with the 1-hop query. Due to the flexible prediction mechanism of GNNs, we feed the features X_{V_t} of the target node V_t, the features $X_{N(V_t)}$ of its 1-hop neighbors $N(V_t)$, and connection information $C(V_t)$ between the target node and its 1-hop neighbors into the target model. At the same time, we apply two data augmentation strategies. One is randomly changing different rates of feature values to the largest and smallest value in the feature space. The other is removing the connected edges between the target node and its 1-hop neighbors one by one. Then, we record two metrics: one is the prediction accuracy of the 1-hop neighbors, and the other is whether the prediction of the target node is the same as its ground truth.

Specifically, we here explain the attack features' names while acquiring with the 1-hop query. Figure 4.2 consists of features with a high absolute SHAP value. The $n\_acc\_all\_max\_rate$ means the accuracy of the neighbors while keeping all the edges and altering a percentage (*rate*) of the target node's feature values to the maximum value in the feature space. The $n\_acc\_none\_max\_rate$ is similar except all the edges are removed. The $n\_acc\_avg\_max\_rate$ is the average accuracy of the neighbors during removing edges. The $i\_all\_max\_rate$ indicates whether the prediction of the target node with the changed feature (changing $rate$ percentage of features into to the maximum value) is the same as its ground truth while keeping all the edges. The $i\_step\_max\_rate$ presents whether the prediction of the target node with the changed feature is the same as its ground truth while reducing edges step by step. The names of attack features are slightly different while changing the feature value to the smallest value, i.e., replacing max with min. Besides, we record the real change percentage $change\_p\_rate$ of the feature values while randomly selecting a percentage (*rate*) of feature values for altering[1].

Our label-only MIA combines fixed properties, the result of the 0-hop query, and the result of the 1-hop query as the attack features. Previous studies [58, 97, 38] obtained the prediction of the target node with 2-hop neighbors or all nodes in the training or testing data as the input of the target model. They contain more information than our label-only MIA when predicting the label of the target node. Under the label-only condition, we cannot get the prediction probability vector from the target model. However, while dropping edges or changing its feature values, the prediction accuracy of the target model on the target node and its neighbors could also provide information for distinguishing the training and testing data.

4.2.2. Experiments

Datasets

We conduct experiments on four datasets: Cora_ML, CiteSeer, DBLP, and PubMed [14]. The main reason for selecting those datasets is that the number of nodes and classes of those datasets varies, which is beneficial for various experiments. Table 4.1 describes the statistics of those four datasets.

Table 4.1: Statistical information of datasets.

Dataset	Classes	# Edges	# Nodes	Length of node feature
Cora_ML	7	16,316	2,995	2,879
CiteSeer	6	10,674	4,230	602
DBLP	4	105,734	17,716	1,639
PubMed	3	88,648	19,717	500

Model Architectures and Training Settings

The GNN models we consider are GCN, GAT, GraphSAGE, and GIN. We vary the model

[1] After randomly selecting a percentage (*rate*) of feature values, it's possible that some selected feature values are already the minimum or maximum value. Thus, the real change percentage ($change\_p\_rate$) indicates the percentage of feature values which are actually changed.

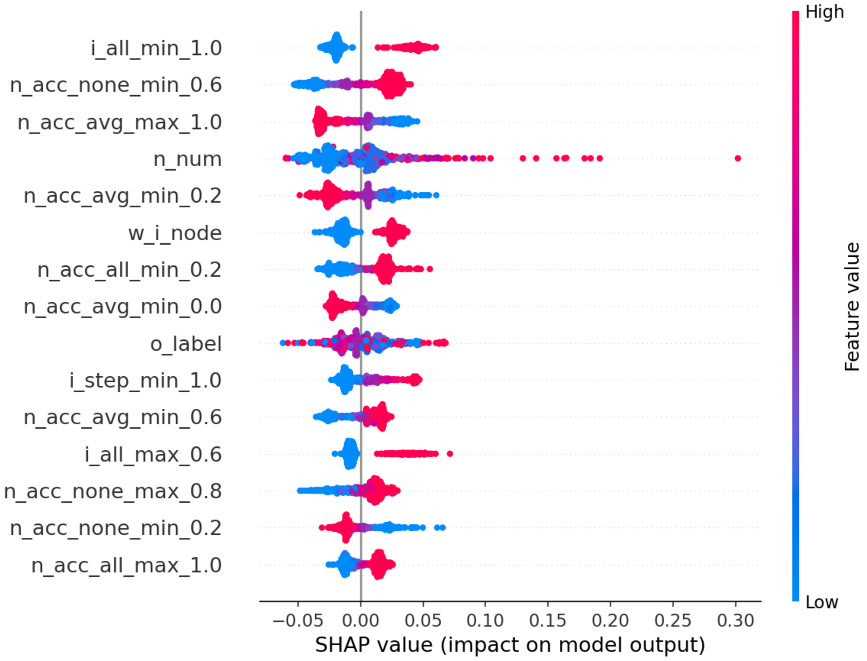


Figure 4.2: The SHAP values of attack features (top 15). The left column indicates the attack features ranked by the absolute SHAP value. The colorful right line represents the ruler of feature values. In the middle, the cluster of points reflects data points in the dataset. The x-axis shows the SHAP value of each feature within one data point.

architectures and training settings to explore the influence of the overfitting level on the attack performance. Here, we discuss high and low overfitting levels of the target models with various model architectures and training settings.

Low overfitting level. In this setting, we set four GNN models with three layers (input, hidden, and output layers) and 16 neurons in the hidden layer. The optimization algorithm of GNNs is Adam, with a learning rate of 6e-3 and a weight decay of 0.5. The number of training epochs is 400. Besides, we apply the BatchNorm, Dropout (0.5), and Jumping knowledge (concatenation) [147] to reduce the overfitting level. Jumping knowledge (concatenation) concatenates the output of each layer as the input of the last layer.

High overfitting level. Different from models with low overfitting level, we set the number of layers to 5 (3 hidden layers) and the number of neurons in the hidden layer to 64. The optimization algorithm of GNNs is Adam, with a learning rate of 0.001 and without weight decay. The number of training epochs is 200. In addition, we do not apply any strategies to reduce the overfitting level because we attempt to get the model with a high overfitting level for comparison.

We decide on the training hyperparameters of models with a high or low overfitting level by increasing or decreasing the overfitting level in a reasonable range. Besides, we

Table 4.2: The attack performance of our label-only MIA after ten repetitions (low overfitting level).

Dataset	GNN	Used Nodes	Test Acc (target model)	Train Acc (target model)	Our Attack (avg acc, pre, rec, auc, f1, low_fpr_0.01_tpr)
Cora_ML	GAT	1,344	0.736	0.883	[**0.604**, 0.605, 0.618, 0.658, 0.607, 0.059]
	GCN		0.763	0.951	[**0.613**, 0.621, 0.606, 0.666, 0.607, 0.065]
	GIN		0.664	0.875	[**0.59**, 0.624, 0.49, 0.631, 0.536, 0.041]
	GraphSAGE		0.73	0.956	[**0.602**, 0.618, 0.559, 0.65, 0.581, 0.061]
CiteSeer	GAT	3,336	0.793	0.879	[**0.598**, 0.611, 0.585, 0.647, 0.583, 0.032]
	GCN		0.81	0.942	[**0.599**, 0.622, 0.536, 0.645, 0.569, 0.031]
	GIN		0.777	0.921	[**0.592**, 0.605, 0.563, 0.635, 0.571, 0.032]
	GraphSAGE		0.795	0.937	[**0.581**, 0.571, 0.69, 0.617, 0.621, 0.032]
DBLP	GAT	7,920	0.732	0.83	[**0.593**, 0.604, 0.586, 0.645, 0.584, 0.048]
	GCN		0.746	0.884	[**0.593**, 0.598, 0.618, 0.644, 0.596, 0.043]
	GIN		0.715	0.865	[**0.579**, 0.59, 0.563, 0.621, 0.564, 0.026]
	GraphSAGE		0.739	0.891	[**0.579**, 0.578, 0.643, 0.618, 0.595, 0.032]
PubMed	GAT	12,300	0.861	0.884	[**0.583**, 0.573, 0.661, 0.636, 0.61, 0.033]
	GCN		0.867	0.907	[**0.596**, 0.601, 0.58, 0.657, 0.589, 0.059]
	GIN		0.852	0.899	[**0.568**, 0.572, 0.571, 0.611, 0.563, 0.034]
	GraphSAGE		0.866	0.923	[**0.557**, 0.548, 0.665, 0.597, 0.598, 0.033]

refer to the settings in previous works [58, 97].

The attack model is a multilayer perceptron (MLP) with 2 hidden layers, each of which has 64 neurons. The optimization algorithm is Adam, with a learning rate of 0.001. The number of training epochs is 300, and the batch size is 32. We use the Binary Cross Entropy to guide the training of the attack model. Our attack model's architecture is similar to attack models in previous works [58, 97].

Evaluation Metrics

We train the attack model with the objective function related to the Binary Cross Entropy. The output of the attack model is the probability of the input being a member of training data. We evaluate the attack performance with six metrics: precision, recall, F1 with the threshold of 0.5, accuracy, AUC, and True Positive Rate (TPR) with the False Positive Rate of 0.1, which is inspired by the work of Carlini et al. [19].

Experimental Steps

The implementation of our label-only MIA is composed of several steps. First, we divide the total dataset into two parts for target and shadow models, and each part dataset is continuously split into training and testing data, i.e., the total dataset is split into four sub-datasets. Then, we generate the attack dataset from the shadow model and shadow dataset for training and testing the attack model. After extracting the attack features from the target model and target dataset, we evaluate the attack performance of the attack model with the extracted attack features as the input. Apart from implementing our label-only MIA, we also implement previous probability-based MIAs under the same environment and settings. Those settings consist of the same models trained with the same sub-datasets, the training of the attack models, and the model selection strategy. Besides, we explore the impact of several factors on our label-only MIA. Those factors include the overfitting level, sampling methods, and attack model selection strategies. Furthermore, we attempt to relax two assumptions about the shadow dataset and the target model. Finally, we analyze the effectiveness of some defenses.

Previous Probability-based MIAs. We use eight probability-based MIAs proposed in previous papers [58, 97, 116, 107, 153, 118] as comparison with our label-only MIA. In those methods, the attacker can obtain the prediction probability vector from the target model and extract the attack features or metrics from the prediction probability vector, which is impossible under the label-only condition. Among these eight probability-based MIAs, four MIAs utilize the classifier to determine membership. They train the attack model based on the attack features. Specifically, the attack features of those four probability-based MIAs with classifiers are the output of the 0-hop query (top two probability values), the 2-hop query (top two probability values), the combination of the output of the 0-hop and 2-hop queries (four probability values in total), and the output of the training (if the target node is from the training data) or testing data (if the target node is from the testing data) (all probability values), respectively[2]. In our work, we name four probability-based MIAs with classifiers: 0-hop, 2-hop, the combination of 0-hop and 2-hop, and all probability methods. The other four MIAs calculate metrics based on the prediction probability vector. They leverage prediction correctness (Gap Attack), probability of ground truth, cross-entropy, and modified cross-entropy of the prediction to distinguish members and non-members. Notably, those four methods select thresholds with the help of shadow models.

Overfitting. As mentioned in the model architectures and settings, we attempt to explore the influence of the overfitting level on the attack performance. Thus, we train models of different architectures with various settings to explore the impact of the overfitting level on our label-only MIA.

Sampling Methods. We focus on the node-level GNN in this section. The dataset for training GNNs is composed of a graph with many nodes and edges. As we mentioned, we need to sample four sub-datasets with equal or approximately equal data points in each class. However, the number of nodes in each class is not equal and even has a large gap. Therefore, the sampling cannot guarantee that each class has an equal number of data points in each sub-dataset unless decreasing the number of data points in each sub-dataset. Based on this situation, we take three sampling methods into comparison. The first is a random sampling of four sub-datasets with maximum utilization of data points (called *the random sampling method*). The second one is a strict class-balanced sampling approach (called *the balanced sampling method*), which guarantees non-overlapping, class-balanced, and fewer data points in each sub-dataset. The third one is *the partially balanced sampling method*, which ensures the class balance in the training data of target and shadow models and randomly samples data points for the testing data.

Attack Model Selection Strategies. We train and test the attack model based on the attack dataset. Then, we evaluate the attack performance of the attack model with attack features from the target model. The selection of the attack model during the training process impacts the final attack model. Here, we choose the attack model based on five

[2] In the training phase of the attack model, the adversary knows the membership of shallow dataset thus it can use the total shallow training data (shallow testing data) if the target node is from the shallow training data (shallow testing data) to query the shadow model getting the attack features. However, in the evaluation phase, the adversary can only query the target model with the whole target dataset to get the attack features of the target node since the adversary has no knowledge about the membership of the target node

metrics, including training accuracy (*train_acc*), testing accuracy (*test_acc*), training loss (*train_loss*), testing loss (*test_loss*), and evaluation accuracy (*evaluate_acc*). Importantly, the adversary cannot compute evaluation accuracy in a practical scenario because the adversary cannot obtain the attack features of the target dataset for evaluation before selecting the final attack model. We here implement the selection strategy based on evaluation accuracy for comparison. Finally, we analyze the attack performance of those model selection strategies.

Assumptions Relaxation. The assumptions about the shadow dataset and the target model's information (GNN type and architecture) are not always available in real-world settings. Therefore, we relax those two assumptions alone and together to verify the effectiveness of our label-only MIA.

4.2.3. Results and Discussions

In this section, we provide the results of our experiments and discuss the findings from the results. We first compare the attack performance of our label-only MIA with eight previous probability-based MIAs. Then, we interpret the attack model with SHAP values. What's more, we explore the influence of different factors on the attack performance. Finally, we relax two assumptions about the shadow dataset and the target model's architecture.

Attack Performance Comparison

As mentioned, the adversary trains the shadow model, which has the same architecture as the target model, with the shadow dataset from the same distribution as the target dataset. Table 4.2 provides the attack performance of our label-only MIA after ten repetitions. Repeating each experiment ten times reduces the impact of the randomness, which is the same as the repetition times in the previous work [97].

In this table, the number of data points used in each experiment is related to the balanced sampling method. The overfitting level, measured by the gap between the training and testing accuracy of the target model, is relatively low. From the table, we can see that the attack accuracy, precision, and AUC value are around 0.6 in most experiments, indicating the effectiveness of our label-only MIA on GNNs.

Table 4.3 indicates the attack performance of four previous probability-based MIAs with classifiers, including 0-hop, 2-hop, the combination of 0-hop and 2-hop, and all probability methods. We implement those four MIAs under the same settings as the results of Table 4.2. The settings include the same target and shadow models (a low overfitting level) trained with the same dataset split, the training of the attack models, and the model selection strategy. Each result row in Table 4.3 corresponds with the result row under the same dataset and GNN model in Table 4.2. The "0-hop", "2-hop", "0-hop and 2-hop combination", and "all probability" in the table indicate the results of four probability-based MIAs with classifiers. Each result has six values: the average accuracy, precision, recall, AUC value, F1 score, and TPR under low FPR (0.01) after ten repetitions. We highlight the average accuracy of each result for a clear comparison in bold and red. Comparing the average attack accuracy of each row in two tables, we can see that our label-only MIA has higher average accuracy than previous probability-based MIA in most cases. For instance, on the GCN model and PubMed dataset, our label-only MIA achieves the average attack accuracy of 0.596, while four previous probability-based MIAs with classifiers obtain values of 0.506, 0.506, 0.504, and 0.507, respectively. Table 4.4 presents the attack performance of the transfer attack (Li et al. [78]) and four other previous probability-based MIAs with metrics. Four probability-based MIAs with metrics utilize prediction correctness (Gap Attack), probability of ground truth, cross-entropy, and modified cross-entropy to determine the membership of data by comparing the metrics with a threshold [153, 116, 107, 118]. The evaluation metrics are the same as Table 4.3. The "_" in the table means we cannot obtain this metric because no threshold is related to metric calculation. Comparing the average attack accuracy (bold and red) of Table 4.4 and Table 4.2, we can also find that our label-only MIA has a higher attack performance than those MIAs. It reflects that our label-only MIA has a competitive, even better performance than previous probability-based methods in most experiments under our environment and settings.

Table 4.3: The attack performance of four probability-based attacks with classifiers after ten repetitions.

Dataset	GNN	probability-based methods with classifiers (avg acc, pre, rec, auc, f1, low_fpr_0.01_tpr)			
		0-hop	2-hop	0-hop and 2-hop combination	all probability
Cora_ML	GAT	[**0.576**, 0.552, 0.634, 0.626, 0.546, 0.076]	[**0.549**, 0.508, 0.457, 0.61, 0.409, 0.033]	[**0.588**, 0.563, 0.62, 0.629, 0.548, 0.088]	[**0.571**, 0.511, 0.493, 0.652, 0.454, 0.078]
	GCN	[**0.691**, 0.67, 0.874, 0.765, 0.746, 0.157]	[**0.633**, 0.622, 0.807, 0.685, 0.687, 0.059]	[**0.693**, 0.675, 0.851, 0.765, 0.739, 0.142]	[**0.635**, 0.681, 0.639, 0.723, 0.591, 0.121]
	GIN	[**0.613**, 0.611, 0.761, 0.692, 0.655, 0.092]	[**0.575**, 0.577, 0.722, 0.604, 0.609, 0.009]	[**0.6**, 0.58, 0.724, 0.666, 0.629, 0.106]	[**0.55**, 0.587, 0.52, 0.575, 0.51, 0.026]
	GraphSAGE	[**0.627**, 0.68, 0.732, 0.747, 0.636, 0.136]	[**0.626**, 0.685, 0.73, 0.762, 0.625, 0.107]	[**0.639**, 0.66, 0.793, 0.74, 0.68, 0.124]	[**0.61**, 0.66, 0.324, 0.637, 0.372, 0.115]
CiteSeer	GAT	[**0.533**, 0.478, 0.329, 0.562, 0.338, 0.013]	[**0.518**, 0.422, 0.436, 0.539, 0.399, 0.009]	[**0.524**, 0.426, 0.467, 0.555, 0.414, 0.014]	[**0.515**, 0.38, 0.281, 0.531, 0.295, 0.01]
	GCN	[**0.555**, 0.456, 0.45, 0.567, 0.439, 0.011]	[**0.559**, 0.523, 0.475, 0.6, 0.463, 0.017]	[**0.551**, 0.526, 0.425, 0.575, 0.443, 0.009]	[**0.528**, 0.507, 0.366, 0.525, 0.388, 0.007]
	GIN	[**0.535**, 0.48, 0.618, 0.562, 0.508, 0.01]	[**0.541**, 0.535, 0.759, 0.567, 0.601, 0.011]	[**0.534**, 0.534, 0.728, 0.568, 0.581, 0.01]	[**0.515**, 0.498, 0.248, 0.519, 0.266, 0.01]
	GraphSAGE	[**0.522**, 0.476, 0.411, 0.534, 0.348, 0.014]	[**0.54**, 0.416, 0.381, 0.551, 0.354, 0.012]	[**0.53**, 0.417, 0.393, 0.569, 0.364, 0.011]	[**0.519**, 0.312, 0.26, 0.523, 0.268, 0.01]
DBLP	GAT	[**0.516**, 0.539, 0.175, 0.547, 0.231, 0.011]	[**0.514**, 0.471, 0.276, 0.539, 0.278, 0.011]	[**0.517**, 0.479, 0.274, 0.54, 0.299, 0.01]	[**0.503**, 0.52, 0.161, 0.521, 0.188, 0.015]
	GCN	[**0.532**, 0.545, 0.406, 0.561, 0.415, 0.011]	[**0.534**, 0.548, 0.424, 0.556, 0.419, 0.013]	[**0.532**, 0.543, 0.412, 0.562, 0.421, 0.011]	[**0.527**, 0.524, 0.416, 0.549, 0.432, 0.014]
	GIN	[**0.514**, 0.436, 0.325, 0.531, 0.299, 0.012]	[**0.517**, 0.499, 0.373, 0.529, 0.339, 0.006]	[**0.524**, 0.453, 0.358, 0.538, 0.318, 0.006]	[**0.507**, 0.372, 0.277, 0.527, 0.266, 0.006]
	GraphSAGE	[**0.531**, 0.53, 0.49, 0.553, 0.465, 0.011]	[**0.538**, 0.537, 0.474, 0.558, 0.463, 0.012]	[**0.539**, 0.548, 0.391, 0.562, 0.423, 0.015]	[**0.554**, 0.557, 0.576, 0.583, 0.536, 0.017]
PubMed	GAT	[**0.498**, 0.415, 0.185, 0.503, 0.198, 0.008]	[**0.5**, 0.341, 0.384, 0.497, 0.306, 0.009]	[**0.497**, 0.442, 0.365, 0.501, 0.352, 0.009]	[**0.505**, 0.464, 0.531, 0.511, 0.456, 0.01]
	GCN	[**0.506**, 0.353, 0.217, 0.515, 0.229, 0.009]	[**0.506**, 0.478, 0.291, 0.512, 0.286, 0.009]	[**0.504**, 0.446, 0.203, 0.508, 0.224, 0.008]	[**0.507**, 0.476, 0.342, 0.516, 0.353, 0.007]
	GIN	[**0.502**, 0.393, 0.649, 0.508, 0.455, 0.007]	[**0.502**, 0.398, 0.376, 0.508, 0.294, 0.007]	[**0.501**, 0.267, 0.375, 0.504, 0.262, 0.009]	[**0.499**, 0.239, 0.125, 0.497, 0.134, 0.006]
	GraphSAGE	[**0.504**, 0.404, 0.273, 0.513, 0.307, 0.008]	[**0.508**, 0.403, 0.295, 0.521, 0.302, 0.009]	[**0.504**, 0.451, 0.293, 0.52, 0.338, 0.009]	[**0.511**, 0.416, 0.334, 0.525, 0.351, 0.008]

Attack Model Explanation

We describe the details of acquiring the features for training the attack model in Section 4.2.1. To better understand the model's behavior and explain the model, we calculate the SHAP value [88] of each feature. The SHAP value implies the contribution or importance of each feature to the prediction of the model. Figure 4.2 gives the SHAP values of each feature in the attack model under the Cora_ML and GCN setting, which has a higher attack performance which helps to demonstrate the importance of each feature.

The left column in the figure displays features' names, ranked by the absolute SHAP values of each feature over total data points. In the middle, a large number of colorful points represent total data points with different feature values and SHAP values in each row. The x-axis means the SHAP value. The right line implies the feature values for data points in the middle. We can see that the feature named "i_all_min_1.0" has the top absolute SHAP value. This feature presents whether the prediction of the target model to the target node is the same as the ground truth while keeping all the edges between the target node with 1-hop neighbors and changing 100% of the target node's features to the minimum value. In addition, we can observe that most features have an apparent positive or negative impact on model output while being assigned high or low values. Besides, the features with the prefix "n_acc" have higher SHAP values than other features, which indicates that neighboring nodes' accuracy under various settings is beneficial for distinguishing the training and testing data.

Influence Factors

This section explores the influence factors of our label-only MIA's, including the overfitting level, sampling methods, and attack model selection strategies.

Overfitting. Table 4.5 shows the attack performance of our label-only MIA on target models with a high overfitting level. We compare the results in this table with Table 4.2 and highlight the average attack accuracy in bold and red. The average attack accuracy can increase with the increasing of the overfitting level. For example, on the DBLP dataset, the average attack accuracy increases from 0.593 and 0.593 (low overfitting level) to 0.6 and 0.599 (high overfitting level) for GAT and GCN models, respectively. On the contrary, the increase in the overfitting level could also degrade our label-only MIA's attack performance. For instance, the average attack accuracies reduce from high (0.604, 0.613, 0.59, and 0.602) to low values (0.597, 0.596, 0.536, and 0.593) under combinations of Cora_ML with GAT, GCN, GIN, and GraphSAGE, respectively. Therefore, the higher overfitting level impacts the attack performance, i.e., it could increase or decrease our label-only MIA's attack accuracy in specific cases. We use the target models with a low overfitting level for the following experiments.

Sampling Methods. As mentioned in the experimental steps, we explore the impact of three sampling methods, including the (0) random, (1) balanced, and (2) partially balanced sampling methods. Table 4.6 presents the number of data points in four sub-datasets, including the training and testing data of shadow and target models under different sampling methods. In the table, target_train, target_test, shadow_train, and shadow_test represent four sub-datasets. The "total" means the number of data points in this sub-dataset. The "one class" indicates the number of data points in each class of

Table 4.4: The attack performance of transfer attack and four probability-based attacks with metrics after ten repetitions.

Dataset	GNN	Test Acc (target model)	Train Acc (target model)	Attack performance (avg acc, pre, rec, auc, f1, low_fpr_0.01_tpr) '_' means that this setting does not have a corresponding metric				
				label-only method	probability-based methods with metrics			
				Transfer Attack	Gap Attack	Probability of Ground Truth	Cross-Entropy	Modified Cross-Entropy
Cora_ML	GAT	0.722	0.855	[**0.516**, 0.532, 0.288, 0.526, 0.367, 0.02]	**0.567**, 0.542, 0.855, _, 0.664, _	[**0.506**, 0.503, 0.994, 0.731, 0.668, 0.191]	[**0.502**, 0.4, 0.197, 0.274, 0.138, 0.014]	[**0.486**, 0.453, 0.813, 0.269, 0.556, 0.007]
	GCN	0.753	0.95	[**0.628**, 0.63, 0.625, 0.685, 0.625, 0.081]	[**0.598**, 0.558, 0.95, _, 0.703, _]	[**0.51**, 0.505, 0.997, 0.818, 0.671, 0.232]	[**0.57**, 0.623, 0.231, 0.196, 0.24, 0.018]	[**0.424**, 0.422, 0.774, 0.182, 0.54, 0.001]
	GIN	0.673	0.877	[**0.515**, 0.54, 0.234, 0.521, 0.32, 0.017]	[**0.602**, 0.566, 0.877, _, 0.687, _]	[**0.51**, 0.505, 0.993, 0.726, 0.669, 0.024]	[**0.517**, 0.259, 0.146, 0.301, 0.132, 0.009]	[**0.472**, 0.469, 0.861, 0.274, 0.598, 0.003]
	GraphSAGE	0.719	0.957	[**0.524**, 0.551, 0.259, 0.535, 0.35, 0.02]	[**0.619**, 0.571, 0.957, _, 0.715, _]	[**0.514**, 0.507, 0.999, 0.819, 0.673, 0.215]	[**0.557**, 0.586, 0.265, 0.21, 0.239, 0.021]	[**0.429**, 0.409, 0.737, 0.181, 0.516, 0.0]
CiteSeer	GAT	0.798	0.895	[**0.501**, 0.501, 0.492, 0.505, 0.495, 0.009]	[**0.549**, 0.529, 0.895, _, 0.665, _]	[**0.506**, 0.503, 0.996, 0.583, 0.668, 0.006]	[**0.513**, 0.157, 0.139, 0.434, 0.129, 0.005]	[**0.485**, 0.481, 0.865, 0.417, 0.606, 0.004]
	GCN	0.8	0.936	[**0.512**, 0.511, 0.521, 0.514, 0.515, 0.011]	[**0.568**, 0.539, 0.936, _, 0.684, _]	[**0.501**, 0.501, 1.0, 0.603, 0.667, 0.008]	[**0.498**, 0.113, 0.013, 0.422, 0.022, 0.006]	[**0.502**, 0.501, 0.987, 0.397, 0.665, 0.001]
	GIN	0.779	0.915	[**0.513**, 0.514, 0.521, 0.517, 0.516, 0.011]	[**0.568**, 0.54, 0.915, _, 0.679, _]	[**0.511**, 0.506, 0.998, 0.591, 0.671, 0.004]	[**0.502**, 0.101, 0.121, 0.437, 0.098, 0.006]	[**0.489**, 0.464, 0.882, 0.409, 0.601, 0.002]
	GraphSAGE	0.794	0.943	[**0.509**, 0.509, 0.53, 0.512, 0.517, 0.012]	[**0.574**, 0.543, 0.943, _, 0.689, _]	[**0.51**, 0.505, 0.999, 0.617, 0.671, 0.009]	[**0.523**, 0.227, 0.242, 0.406, 0.224, 0.005]	[**0.468**, 0.452, 0.762, 0.383, 0.551, 0.001]
DBLP	GAT	0.737	0.828	[**0.508**, 0.508, 0.523, 0.509, 0.515, 0.011]	[**0.546**, 0.529, 0.828, _, 0.646, _]	[**0.502**, 0.501, 0.996, 0.577, 0.667, 0.01]	[**0.505**, 0.121, 0.06, 0.44, 0.058, 0.007]	[**0.493**, 0.494, 0.945, 0.423, 0.644, 0.006]
	GCN	0.752	0.886	[**0.513**, 0.512, 0.552, 0.518, 0.531, 0.011]	[**0.567**, 0.541, 0.886, _, 0.672, _]	[**0.501**, 0.5, 1.0, 0.606, 0.667, 0.008]	[**0.499**, 0.111, 0.009, 0.423, 0.016, 0.007]	[**0.501**, 0.5, 0.991, 0.394, 0.665, 0.003]
	GIN	0.721	0.863	[**0.511**, 0.511, 0.507, 0.514, 0.509, 0.012]	[**0.571**, 0.545, 0.863, _, 0.668, _]	[**0.501**, 0.5, 1.0, 0.587, 0.667, 0.005]	[**0.498**, 0.048, 0.003, 0.457, 0.005, 0.004]	[**0.502**, 0.501, 0.997, 0.413, 0.667, 0.003]
	GraphSAGE	0.733	0.892	[**0.511**, 0.511, 0.543, 0.514, 0.526, 0.012]	[**0.579**, 0.549, 0.892, _, 0.68, _]	[**0.506**, 0.503, 0.997, 0.624, 0.669, 0.01]	[**0.502**, 0.24, 0.108, 0.411, 0.085, 0.005]	[**0.49**, 0.471, 0.898, 0.376, 0.609, 0.002]
PubMed	GAT	0.859	0.885	[**0.506**, 0.506, 0.539, 0.506, 0.517, 0.01]	[**0.513**, 0.508, 0.885, _, 0.645, _]	[**0.502**, 0.501, 0.987, 0.53, 0.665, 0.008]	[**0.503**, 0.251, 0.234, 0.477, 0.2, 0.009]	[**0.494**, 0.489, 0.782, 0.47, 0.571, 0.008]
	GCN	0.868	0.905	[**0.505**, 0.504, 0.61, 0.505, 0.551, 0.01]	[**0.519**, 0.51, 0.905, _, 0.653, _]	[**0.502**, 0.501, 0.994, 0.529, 0.666, 0.009]	[**0.503**, 0.252, 0.172, 0.482, 0.148, 0.008]	[**0.495**, 0.491, 0.837, 0.471, 0.596, 0.005]
	GIN	0.849	0.895	[**0.508**, 0.507, 0.575, 0.508, 0.537, 0.01]	[**0.523**, 0.513, 0.895, _, 0.652, _]	[**0.504**, 0.502, 0.989, 0.527, 0.666, 0.006]	[**0.502**, 0.102, 0.175, 0.488, 0.128, 0.007]	[**0.494**, 0.484, 0.837, 0.473, 0.583, 0.005]
	GraphSAGE	0.867	0.92	[**0.507**, 0.506, 0.599, 0.508, 0.548, 0.01]	[**0.526**, 0.515, 0.92, _, 0.66, _]	[**0.504**, 0.502, 0.994, 0.547, 0.667, 0.007]	[**0.507**, 0.234, 0.213, 0.467, 0.194, 0.008]	[**0.489**, 0.483, 0.794, 0.453, 0.579, 0.004]

this sub-dataset. The "-" means uncertainty indicating that the number of data points in each class is not fixed due to random selection. From the table, the uncertainty represents the class imbalance in the random and partially balanced sampling methods.

Table 4.7 provides the attack performance of different datasets and GNN models under three sampling methods. From the table, we can observe that the third sampling method achieves the best average attack accuracy in all datasets and GNN models except the GIN model on the CiteSeer dataset, which obtains the highest average attack accuracy with the first sampling method. From Table 4.6, we can see that the attack accuracy is higher under the first and third sampling methods, which implies class imbalance can improve the attack accuracy. Besides, the difference in average attack accuracy reaches 7% between the second and third sampling methods on the GAT model with the DBLP dataset. It indicates that the sampling method influences the attack performance via class imbalance and achieves a maximum gap of 7% for average attack accuracy. Olatunji et al. [97] used the partially balanced sampling method, while He et al. [58] leveraged the random sampling method. We use the balanced sampling method by default to avoid the disturbance brought by the class imbalance.

Attack Model Selection Strategies. In the experimental steps, we explain that we select the attack model during the training process based on five metrics, including training accuracy (*train_acc*), testing accuracy (*test_acc*), training loss (*train_loss*), testing loss (*test_loss*), and evaluation accuracy (*evaluate_acc*). Table 4.8 gives the average attack accuracy of the selected attack model. From the table, we could observe that selection

Table 4.5: The attack performance of our label-only MIA after ten repetitions (high overfitting level).

Dataset	GNN	Test Acc (target model)	Train Acc (target model)	Our Attack (avg acc, pre, rec, auc, f1, low_fpr_0.01_tpr)
Cora_ML	GAT	0.658	0.973	[**0.597**, 0.635, 0.504, 0.652, 0.545, 0.056]
	GCN	0.684	0.96	[**0.596**, 0.605, 0.586, 0.634, 0.586, 0.033]
	GIN	0.298	0.577	[**0.536**, 0.539, 0.525, 0.554, 0.525, 0.022]
	GraphSAGE	0.619	0.999	[**0.593**, 0.602, 0.595, 0.637, 0.59, 0.054]
CiteSeer	GAT	0.748	0.913	[**0.592**, 0.612, 0.546, 0.641, 0.567, 0.042]
	GCN	0.767	0.922	[**0.603**, 0.62, 0.572, 0.656, 0.586, 0.043]
	GIN	0.327	0.578	[**0.538**, 0.536, 0.569, 0.556, 0.549, 0.029]
	GraphSAGE	0.732	0.874	[**0.591**, 0.618, 0.532, 0.637, 0.557, 0.027]
DBLP	GAT	0.716	0.916	[**0.6**, 0.617, 0.585, 0.648, 0.582, 0.044]
	GCN	0.722	0.917	[**0.599**, 0.627, 0.549, 0.653, 0.572, 0.062]
	GIN	0.428	0.652	[**0.555**, 0.561, 0.509, 0.583, 0.522, 0.025]
	GraphSAGE	0.666	0.856	[**0.579**, 0.586, 0.593, 0.618, 0.577, 0.029]
PubMed	GAT	0.828	0.912	[**0.58**, 0.616, 0.477, 0.62, 0.524, 0.033]
	GCN	0.838	0.909	[**0.596**, 0.611, 0.561, 0.653, 0.575, 0.058]
	GIN	0.586	0.639	[**0.522**, 0.52, 0.607, 0.537, 0.542, 0.018]
	GraphSAGE	0.836	0.921	[**0.561**, 0.558, 0.672, 0.594, 0.603, 0.031]

strategies based on testing accuracy and loss have a slightly better attack accuracy than those based on training accuracy and loss with a maximum gap of 1% average attack accuracy. Therefore, we use testing accuracy for selecting the attack model in our MIAs.

Under our environment and settings, previous probability-based methods with classifiers might not achieve the attack performance reported in their papers [58, 97]. This kind of difference can be explained by three influence factors analyzed in this section and the change in the attack environment, including the model's architecture, training process, and hyperparameters.

Assumptions Relaxation

We design and conduct an ablation study on the relaxation of two assumptions about the shadow dataset and the target model's information. In the first experiment, we utilize shadow datasets from other distributions. Secondly, we train the shadow model with different types and architectures from the target model. Finally, we relax those two assumptions together.

Table 4.6: The number of data points in each sub-dataset under different sampling methods.

Dataset	Sampling Method	target_train		target_test		shadow_train		shadow_test	
		total	one class	total	one class	total	one class	total	one class
Cora_ML	0	749	-	749	-	748	-	749	-
	1	336	48	336	48	336	48	336	48
	2	630	90	630	-	630	90	630	-
CiteSeer	0	1,058	-	1,057	-	1,058	-	1,057	-
	1	834	139	834	139	834	139	834	139
	2	600	100	600	-	600	100	600	-
DBLP	0	4,429	-	4,429	-	4,429	-	4,429	-
	1	1,980	495	1,980	495	1,980	495	1,980	495
	2	3,200	800	3,200	-	3,200	800	3,200	-
PubMed	0	4,929	-	4,929	-	4,930	-	4,929	-
	1	3,075	1,025	3,075	1,025	3,075	1,025	3,075	1,025
	2	4,500	1,500	4,500	-	4,500	1,500	4,500	-

Table 4.7: The average attack accuracy of four datasets and GNN models under three sampling methods after ten repetitions.

Sampling Method	Dataset	GNN	Test Acc (target model)	Train Acc (target model)	Avg Acc
0	Cora_ML	GCN	0.68	0.851	0.621
	CiteSeer	GIN	0.742	0.956	0.613
	DBLP	GAT	0.653	0.682	0.614
	PubMed	GraphSAGE	0.871	0.919	0.55
1	Cora_ML	GCN	0.749	0.935	0.608
	CiteSeer	GIN	0.775	0.927	0.591
	DBLP	GAT	0.732	0.837	0.594
	PubMed	GraphSAGE	0.87	0.925	0.574
2	Cora_ML	GCN	0.777	0.941	0.651
	CiteSeer	GIN	0.728	0.946	0.568
	DBLP	GAT	0.771	0.82	0.664
	PubMed	GraphSAGE	0.866	0.919	0.581

Figure 4.3 shows the average attack accuracy comparison of the first experiment. We fix the target model and shadow model to GCN. And we train the target and shadow models with Cora_ML, CiteSeer, DBLP, and PubMed datasets, as shown in the x-axis and y-axis in the figure. While the target models are trained with Cora_ML, CiteSeer, DBLP, and PubMed (each row in the figure), we obtain the largest attack accuracy with shadow datasets trained with PubMed (0.679), DBLP (0.646), PubMed (0.653), and CiteSeer (0.62), which are not from the same datasets as the target datasets. Figure 4.4 shows the result of the second experiment. We fix the target dataset and shadow dataset to Cora_ML. The main reason for selecting Cora_ML is that the average attack accuracy of Cora_ML is relatively higher than other datasets, which means the change is evident while relaxing the second assumption. From the figure 4.4, we obtain the highest attack accuracy while the combination of target and shadow models are, GCN-GAT, GIN-GCN, GAT-GCN, and GraphSAGE-GraphSAGE, most of which do not have the same model type for target and shadow models. Figure 4.5 illustrates the average attack accuracy of the third experiment. Similarly, the maximum attack accuracy is not from the settings where the target model is trained with the same dataset and model type as the shadow

Table 4.8: The average attack accuracy of four datasets and GNN models with five model selection strategies after ten repetitions.

Dataset	GNN	Test Acc	Train Acc	Average accuracy of the attack model selected by different strategies				
		(target model)	(target model)	train_acc	test_acc	train_loss	test_loss	evaluate_acc
Cora_ML	GCN	0.744	0.943	0.607	0.616	0.603	0.617	0.644
CiteSeer	GIN	0.771	0.924	0.585	0.589	0.582	0.596	0.615
DBLP	GAT	0.734	0.837	0.582	0.592	0.583	0.589	0.604
PubMed	GraphSAGE	0.874	0.924	0.572	0.578	0.565	0.589	0.604

model. From those three figures, we surprisingly find that the relaxation of those two assumptions will not reduce the attack performance but increase the attack performance in most cases, which reflects on relatively light colors on the diagonal. Even when the attack performance decreases, the reduction degree is low. Therefore, the attack performance primarily increases while relaxing the assumptions about the shadow dataset and target model's information. The possible reason for this phenomenon is that the attack features of members (or non-members) are similar or are converted to be alike in the attack model under settings where the shadow dataset's distribution is different from the target dataset, and the shadow model is different from the target model. This phenomenon is also discussed in previous works [107, 58].

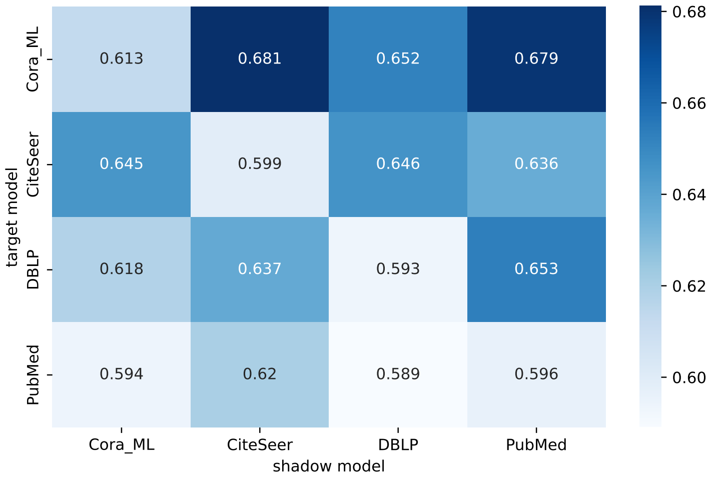


Figure 4.3: The average attack accuracy after ten repetitions while the shadow dataset is from the other distribution. The fixed GNN for shadow and target models is GCN. The x-axis means the setting of the shadow model. The y-axis represents the setting of the target model.

Defenses

Table 4.9 shows the average attack accuracy of ten repetitions against four different defenses. The defenses include Normalization (the BatchNorm), Dropout (0.5), Regularization (the Adam with weight decay of 0.5), and Jumping knowledge (concatenation). Here, we select the GCN model on the Cora_ML dataset with a high overfitting level to evaluate the robustness of our MIAs against these defenses. From the table, we can ob-

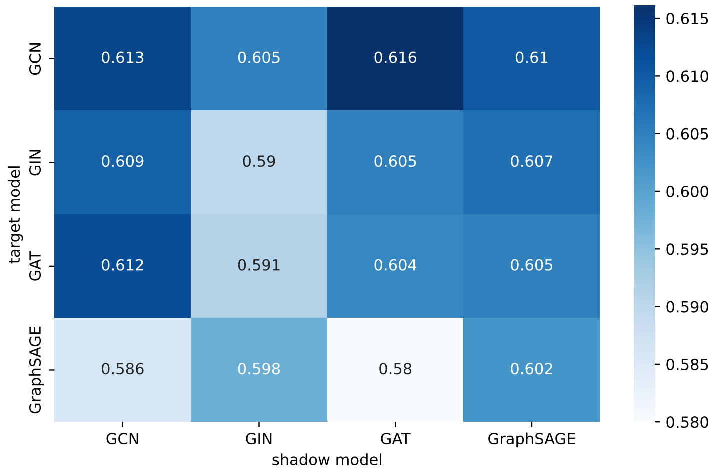


Figure 4.4: The average attack accuracy after ten repetitions while the target model's information is relaxed. The shadow and target datasets are from Cora_ML.

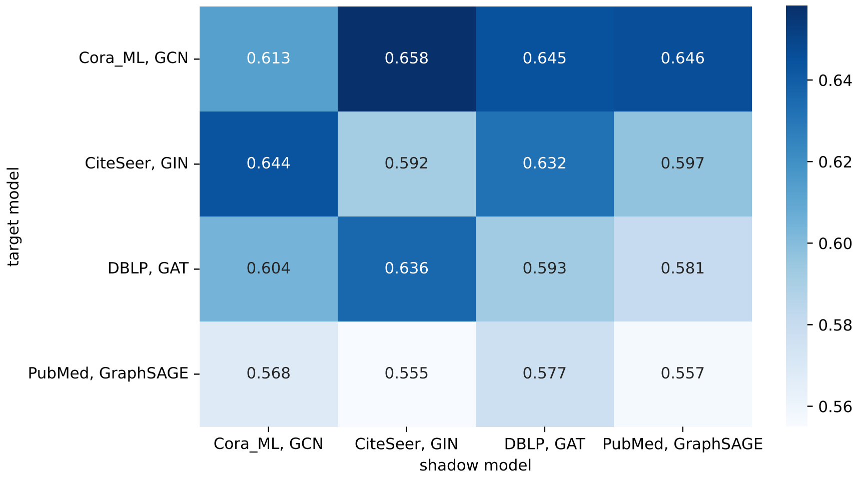


Figure 4.5: The average attack accuracy after ten repetitions while the shadow dataset and the target model's information are relaxed together.

serve that the average accuracy decreases when applying Normalization (row 1, 0.586) or Regularization (row 3, 0.573) compared with no defenses (row 0, 0.596). However, the average accuracy slightly increases when only using the Dropout (row 2, 0.602) or Jumping knowledge (row 4, 0.606). Besides, the combinations between four defenses could raise (row 5, 7, 9, 11, 12, 13, 15) or lower (row 6, 8, 10, 14) the average accuracy. Although the application of Regularization decreases the average accuracy, e.g., in four rows (6, 8, 10, 14), the combination of Regularization and other defenses can increase the average accuracy, like rows 11 and 13. The results show that Regularization and Normalization could slightly decrease the average attack accuracy while applying alone. Moreover, those four defenses cannot prevent our label-only MIA completely with average accuracy of less than 0.5 apart from row 14.

We apply and compare four regular defense mechanisms, all of which cannot prevent our label-only MIA completely. One possible defense against our MIAs is to explore a defense mechanism that takes the GNNs' particular properties into consideration, e.g., flexible prediction strategy and unique connection between nodes and their neighbors. Currently, we do not find an efficient defense mechanism for our label-only MIA and leave it as future work.

Table 4.9: The average attack accuracy after ten repetitions under different defenses (Cora_ML, GCN).

Row	Normalization	Dropout	Regularization	Jumping Knowledge	Avg Acc
0	×	×	×	×	0.596
1	√	×	×	×	0.586
2	×	√	×	×	0.602
3	×	×	√	×	0.573
4	×	×	×	√	0.606
5	√	√	×	×	0.609
6	√	×	√	×	0.581
7	√	×	×	√	0.615
8	×	√	√	×	0.55
9	×	√	×	√	0.607
10	×	×	√	√	0.52
11	√	√	√	×	0.609
12	√	√	×	√	0.614
13	√	×	√	√	0.618
14	×	√	√	√	0.497
15	√	√	√	√	0.613

4.3. Backdoor Horizontal Federated GNNs

Federated Learning enables n clients to train a global model $\boldsymbol{w}$ collaboratively without revealing local datasets. Unlike centralized learning, where local datasets must be collected by a central server before training, FL performs training by uploading the weights of local models ($\{\boldsymbol{w}^i \mid i \in n\}$) to a parametric server. Specifically, FL aims to optimize a loss function:

$$\min_{\boldsymbol{w}} \ell(\boldsymbol{w}) = \sum_{i=1}^{n} \frac{k_i}{n} L_i(\boldsymbol{w}), L_i(\boldsymbol{w}) = \frac{1}{k_i} \sum_{j \in P_i} \ell_j(\boldsymbol{w}, x_j), \tag{4.2}$$

where $L_i(\boldsymbol{w})$ and k_i are the loss function and local data size of i-th client, and P_i refers to the set of data indices with size k_i.

At the t-th iteration, the training can be divided into three steps:

- *Global model download.* All clients download the global model $\boldsymbol{w}_t$ from the server.
- *Local training.* Each client updates the global model by training with their datasets: $\boldsymbol{w}_t^i \leftarrow \boldsymbol{w}_t^i - \eta \frac{\partial L(\boldsymbol{w}_t, b)}{\partial \boldsymbol{w}_t^i}$, where η and b refer to learning rate and local batch, respectively.
- *Aggregation.* After the clients upload their local models $\{\boldsymbol{w}_t^i \mid i \in n\}$, the server updates the global model by aggregating the local models. In this section, we use the averaging aggregation function: $\boldsymbol{w}_{t+1} \leftarrow \sum_{i=1}^{n} \frac{1}{n} \boldsymbol{w}_t^i$.

Backdoor attacks are common in FL systems involving multiple training dataset owners. In such attacks, the adversary $\mathscr{A}$ manipulates one or more local models to generate poisoned models, denoted as $\tilde{W}^i$, which are then aggregated into the global model G_t, thereby compromising its properties. There are two common techniques used in backdoor attacks in FL, as shown in Figure 4.6: 1) data poisoning, where $\mathscr{A}$ manipulates local training dataset(s) D^i_{local} used to train the local model [96, 144], and 2) model poisoning, where $\mathscr{A}$ manipulates the local training process or the trained local models themselves [5]. Regarding data poisoning backdoor attacks in FL, during the local training phase, one or more malicious clients can inject triggers into local benign datasets to produce backdoored datasets. By training on the backdoored datasets, malicious updates can be obtained. Consequently, if the server aggregates with these malicious updates, the global model will exhibit misclassification on the samples with the injected triggers. In the model poisoning backdoor attacks in FL, to enhance the effect of the attacks, the adversaries can also use the method of scaling [5] to increase their weight.

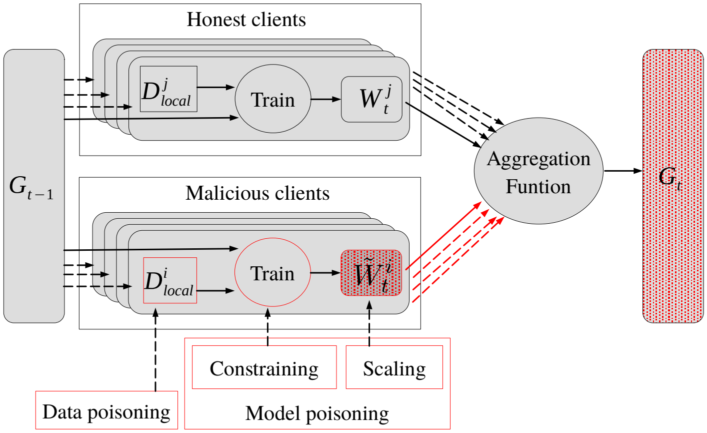


Figure 4.6: Overview of backdoor attacks on FL.

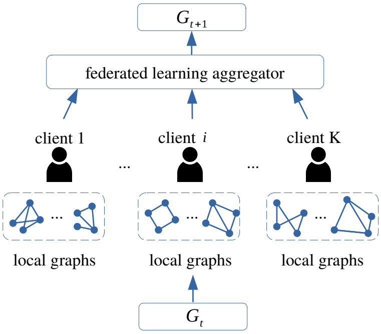


Figure 4.7: Framework of federated GNNs for graph-level task. Each client trains its local GNN model based on local graphs, and the federated learning aggregator aggregates the local models to obtain the global model. G_t and G_{t+1} define the global models at iteration t and $t+1$ respectively. K is the number of clients.

FL has gained increasing attention as a training paradigm where data is distributed at remote devices and models are collaboratively trained in a central server. While FL has been widely studied in Euclidean data, e.g., images, texts, and sound, there are increasing studies about FL in graph data. Figure 4.7 illustrates the framework of federated GNNs for a graph-level task. FL on graph data was introduced in [72], where each client is regarded as a node in a graph. When it comes to detecting financial crimes (e.g., fraud or money laundering), traditional machine learning tends to lead to severe overreporting of suspicious activities. Thanks to the reasoning ability of the graph neural network, its advantages can be well-reflected. Considering the need for privacy, [121] proposed the framework for Federated GNNs to optimize the machine learning model. Besides, other research works [64, 21, 138] have been dedicated to enhancing the security of Federated GNNs. By using secure aggregation, [64] proposed a method to predict the trajectories of objects via aggregating both spatial and dynamic information without information leakage. With differential privacy, [21] and [138] put forward a framework to train Federated GNNs for vertical FL and recommendation system, respectively. Moreover, SpreadGNN was proposed in [56] to perform FL without a server. *Although there is an increasing number of works on FL for graph data, the vulnerability of Federated GNNs to backdoor attacks is still underexplored.* In this section, we focus on data poisoning for our attacks in Federated GNNs as model poisoning requires multiplying large factors to model weights when conducting attacks, which can be detected by traditional byzantine-robust aggregation rules such as Median [154] and Krum [12].

4.3.1. PROBLEM FORMULATION

Overview

FL is a practical choice to push machine learning to users' devices, e.g., smart speakers, cars, and phones. Usually, federated learning is designed to work with thousands or even millions of users without restrictions on eligibility [5], opening up new attack vectors. As stated in [15], training with multiple malicious clients is now considered a practical threat by the designers of federated learning. Because of the data privacy guarantee among the clients in the federated learning, local clients can modify their local training dataset without being noticed. Furthermore, existing federated learning frameworks do not provide a functionality to verify whether the training on local clients has been finished correctly. Consequently, one or more clients can submit their malicious models trained for the assigned task and backdoor functionality.

Threat Model

Unlike traditional machine learning benchmarking datasets, graph datasets, and real-world graphs may exhibit non-independent and identical distribution (non-i.i.d) due to factors like structure and feature heterogeneity [55]. Therefore, following the FL assumptions, we assume that graphs among K clients are non-i.i.d. distributed. The clients engaging in training can be divided into honest and malicious clients. In Table 4.10,[3] we summarize the settings of different experiments shown in Section 4.3.3. Molecular machine learning is a paramount application in the Federated GNNs, where many small graphs are distributed between multiple institutions [55]. Therefore, we run experiments (Exp. I and II) on two molecular datasets, i.e., NCI1 and PROTEINS_full. For these experiments, we set 5 clients in total because, with more clients, the local dataset of each client becomes very small, resulting in severe overfitting for the local models. Similar settings and phenomena can also be found in prior works on Federated GNNs [55]. The choice of small datasets may be a limitation of our work, but real-world cross-silo settings could involve only a few different organizations (from two to one hundred) [66]. Besides the molecular domain, substantial attention has also been given to Federated GNNs in real-world financial scenarios [161, 130]. In such scenarios, clients can be different organizations, e.g., banks, and a GNN model is trained on siloed data, leading to a cross-silo federated learning setting [66]. As shown in Exp. III and IV, we assume 10, 20, and 100 clients for a synthetic dataset, i.e., TRIANGLES, which is a realistic real-world cross-silo scenario [114].

Table 4.10: Summary of the experimental setting (K: number of clients, M: number of malicious clients).

Experiment	Dataset	K	M
Exp. I	NCI1, PROTEINS_full, TRIANGLES	5	2
Exp. II	NCI1, PROTEINS_full, TRIANGLES	5	3
Exp. III	TRIANGLES	10	4, 6
		20	8, 12
Exp. IV	TRIANGLES	100	5, 10, 15, 20
Prior work [55]	Molecules	4	0

All clients strictly follow the FL training process, but the malicious client(s) will inject graph trigger(s) into their training graphs. We also assume the server is conducting model aggregation correctly. Our primary focus is to investigate backdoor attack effectiveness on Federated GNNs, so we adopt two backdoor attack methods as defined below (the definitions of the local trigger and global trigger used in these two attacks are also given).

[3] Exp. I, Exp. II, Exp. III, and Exp. IV represent the experiments of the honest majority attack scenario, malicious majority attack scenario, the impact of the number of clients, and the impact of the percentage of malicious clients, respectively.

Definition 7 (Local Trigger & Global Trigger.) *The local trigger is the specific graph trigger for each malicious client in DBA. The global trigger is the combination of all local triggers.*[4]

Definition 8 (Distributed Backdoor Attack (DBA).) *There are multiple malicious clients, and each of them has its local trigger. Each malicious client injects its local trigger into its training dataset. All malicious clients have the same backdoor task. An adversary $\mathscr{A}$ conducts DBA by compromising at least two clients in FL.*

Definition 9 (Centralized Backdoor Attack (CBA).) *A global trigger consisting of local triggers is injected into one client's local training dataset. An adversary $\mathscr{A}$ conducts CBA by usually compromising only one client in FL.*

Adversary's capability. We assume the adversary $\mathscr{A}$ can corrupt M ($M \leq K$) clients to perform DBA. We perform a complete attack in every round, i.e., a poisoned local dataset is used by malicious clients in every round, following the attack setting in [144]. The adversary cannot impact the aggregation process on the central server nor the training or model updates of other clients.

Adversary's knowledge. We assume that the adversary $\mathscr{A}$ knows the compromised clients' training dataset. In this context, the adversary can generate local triggers as described in Section 4.3.2. Additionally, we follow the original assumptions of FL. The number of clients participating in training, model structure, aggregation strategy, and a global model for each iteration is revealed to all clients, including malicious clients.

Adversary's goal. Unlike some non-targeted attacks [103] aiming to deteriorate the accuracy of the model, the backdoor attacks studied in this section aim to make the global model misclassify the backdoored data samples into specific pre-determined labels (i.e., target label y_t) without affecting the accuracy on clean data.

In distributed backdoor attacks, each malicious client injects its local trigger into its local training dataset to poison the local model. Therefore, DBA can fully leverage the power of FL in aggregating dispersed information from local models to train a poisoned global model. Assuming there are M malicious clients in DBA, each has its local trigger. Each malicious client i in DBA independently implements a backdoor attack on its local model. The adversarial objective for each malicious client i is:

$$\begin{aligned} w_t^{i*} = \underset{w_t^i}{\operatorname{argmin}}(& \sum_{j \in D_{trigger}^i} \ell(w_{t-1}^i(\Phi(x_j^i, \kappa^i), y_t)) \\ & + \sum_{j \in D_{clean}^i} \ell(w_{t-1}^i(x_j^i), y_j^i)), \forall i \in [M], \end{aligned} \tag{4.3}$$

where the poisoned training dataset $D_{trigger}^i$ and clean training dataset D_{clean}^i satisfy $D_{trigger}^i \cup D_{clean}^i = D_{local}^i$ and $D_{trigger}^i \cap D_{clean}^i = \varnothing$. D_{local}^i is the local training dataset of client i. Φ is the function that transforms the clean data with a non-target label into

[4] Since it is an NP-hard problem to decompose a graph into subgraphs [32], we first generate local triggers and then compose them to get the global trigger used in CBA.

poisoned data using a set of trigger generation parameters κ^i. In this section, κ^i consists of trigger size s, trigger density ρ, and poisoning intensity r: $\kappa = \{s, \rho, r\}$.

Trigger Size s: the number of nodes of a local graph trigger. Here, we set the trigger size s as the γ fraction of the graph dataset's average number of nodes. Note that this does not violate our threat model (the adversary does not have access to the whole dataset), as the average number of nodes in the local dataset is similar to that of the whole dataset.
Trigger Density ρ: the complexity of a local graph trigger, which ranges from 0 to 1, and is used in the Erdős-Rényi (ER) model to generate the graph trigger.
Poisoning Intensity r: the ratio that controls the percentage of backdoored training dataset among the local training dataset.

Unlike DBA with multiple malicious clients, there is only one malicious client in CBA.[5] CBA is conducted by embedding a global trigger into a malicious client's training dataset. The global trigger is a graph consisting of local trigger graphs used in DBA, as explained further in Section 4.3.2. Thus, the adversarial objective of the attacker k in round t in CBA is:

$$\begin{aligned} w_t^{k*} = \underset{w_t^k}{\operatorname{argmin}}(& \sum_{j \in D^k_{trigger}} \ell(w^k_{t-1}(\Phi(x^k_j, \kappa), y_t)) \\ & + \sum_{j \in D^k_{clean}} \ell(w^k_{t-1}(x^k_j), y^k_j)), \end{aligned} \tag{4.4}$$

where κ is the combination of κ^i. Utilizing the power of FL in message passing from local models to the global model, the global model is supposed to inherit the backdoor functionality.

4.3.2. Backdoor Attacks against Federated GNNs

General Framework

We focus on subgraph-based (data poisoning) backdoor attacks and the graph classification task. Attackers can perform DBA or CBA as shown in Figure 4.8. In DBA, multiple malicious clients engage in attacking, and they inject local triggers into corresponding malicious clients' local training datasets. CBA is conducted with one malicious client, whose training data is poisoned with the global trigger that consists of the local triggers used in DBA. We describe the notations used throughout the section in Table 4.11.

Distributed Backdoor Attack. For DBA in Federated GNNs, we assume there are M ($M \leq K$) malicious clients among K clients, as shown in Figure 4.8(a). Each malicious client embeds its local training dataset with a specific graph trigger to poison its local model. For instance, in Figure 4.8(a), each malicious client has a local trigger highlighted by a specific color (i.e., orange, green, red, yellow).[6] In this section, we did not use the same local trigger for different malicious clients in DBA as it would mean poisoning intensity for this specific local trigger is increasing, but simultaneously, the total trigger

[5] In practice, the centralized attack can poison more than one client with the same global trigger, as mentioned in [5]. Here, we assume there is one malicious client

[6] Although we use the triangle as the graph trigger for each malicious client, in practice, the local triggers are more complex and different from each other.

Table 4.11: Notations used in this section.

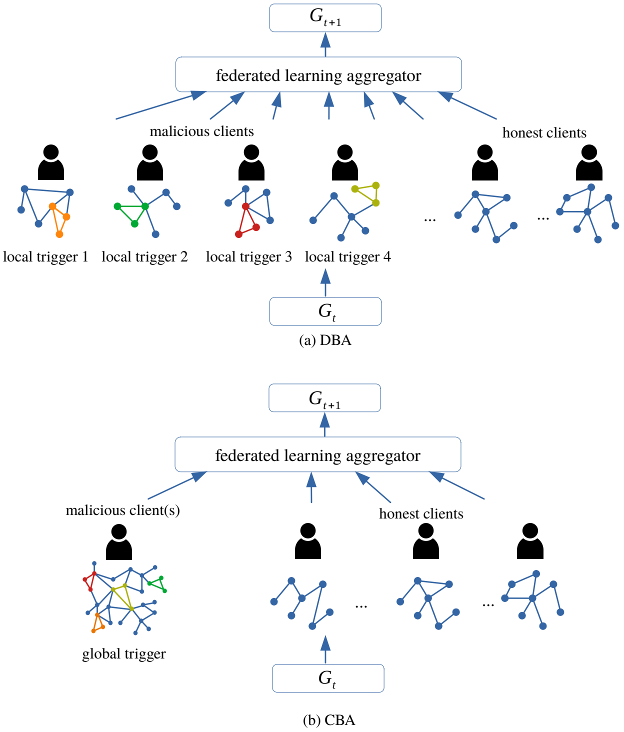
Notations	Descriptions
y_t	target label
G_t	joint global model at round t
E	local epochs
K	number of clients
M	number of malicious clients
C_h, C_m	honest clients, malicious clients
D_{local}	client's local training dataset splitted from dataset D_{train}
D_{test}	testing dataset splitted from dataset D
t_{global}	global trigger
t_{local}	local trigger
w_t^k	client k's local trained model at round t
r	poisoning ratio
s	number of nodes in graph trigger
ρ	edge existence probability in graph trigger
$D_{trigger}$	dataset with trigger embedded
D_{clean}	clean training dataset
$D_{backdoor}$	backdoored training dataset
B	local minibatch size
η	learning rate

pattern activating the backdoor is reduced. We evaluated this setting by running some additional experiments, and we found the attack under this setting is not stronger than the current setting (i.e., different local triggers). Through training with these poisoned training datasets, the poisoned local models are uploaded to the server to update the global model. The final adversarial goal is to use the global trigger to attack the global model. Algorithms 4 and 5 illustrate the distributed backdoor attack in Federated GNNs. We first split the clients into two groups, the honest (C_h) and the malicious one (C_m) (line 2, Algorithm 4). In each round, each client updates its weights through local training (line 13, Algorithm 4), and finally, the global server aggregates local models' weights to update the global model through averaging (line 15, Algorithm 4).

The local training for every client is described in Algorithm 5. If the client is malicious (line 2, Algorithm 5), the local training dataset will be backdoored (line 4, Algorithm 5) with the local trigger (line 3, Algorithm 5. As mentioned in Section 4.3.1, all the local triggers form the global trigger (line 5, Algorithm 5).

We conduct experiments for the malicious majority and honest majority settings to explore the impact of different percentages of malicious clients on the attack success rate. We provide additional motivation for the malicious majority setting in Section 4.3.3.

Centralized Backdoor Attack. Unlike DBA conducted with multiple malicious clients, CBA performs the attack with only one malicious client. CBA is a general approach in a centralized learning scenario. For example, in image classification, the attacker poisons the training dataset with a trigger so that the model misclassifies the data sample with the same trigger into the attacker-chosen label. As shown in Figure 4.8(b), the malicious client embeds its training dataset with the global trigger highlighted by four colors. This global trigger consists of local triggers used in DBA, as shown in Line 5 of Algorithm 5. Specifically, the attacker in CBA embeds its training data with four local patterns, to-



(a) DBA

(b) CBA

Figure 4.8: Attack Framework.

gether constituting a complete global pattern as the backdoor trigger.[7]

To compare the attack performance between the distributed backdoor attack and centralized backdoor attack in Federated GNNs, we need to make sure the trigger pattern in CBA is the union set of local trigger patterns in DBA. We can use two strategies: 1) first generate local triggers in DBA and then combine them to get the global trigger, or 2) first generate a global trigger in CBA and then divide it into M local triggers. We utilize the first strategy as it is an NP-hard problem to divide a graph into several subgraphs [32]. Thus, in different attack scenarios (i.e., honest majority or malicious majority attack scenarios), the CBA performance is different since the global trigger has been changed due to the different number of malicious clients.

Backdoored Data Generation

We adopt the Erdős-Rényi (ER) model [44] to generate triggers (function *GenerateTrigger* in Algorithm 5) as it is more effective than the other methods (e.g., Small World model [134] or Preferential Attachment model [7]) [164]. In particular, *GenerateTrigger* (line 3 in Algorithm 5), creates a random graph of s nodes. An edge between a pair of nodes in this graph is generated with probability ρ.

[7] Here, the four colors are only used to denote four trigger patterns.

Algorithm 4: Distributed Backdoor Attacks in Federated GNNs

```
Input: Dataset $D$, Target label $y_t$
Output: Backdoored Global model $G_{t+1}$, global trigger $t_{global}$
Function DBA():
    $C_h, C_m \leftarrow ClientSplit(Clients)$
    $D_{local}, D_{test} \leftarrow DataSplit(D)$
    $t_{global} \leftarrow \varnothing$
    Server executes:
    initialize $G_0$, $f = False$
    foreach round $t = 0, 1, 2, \ldots$ do
        foreach client $k \in (C_h \cup C_m)$ do
            $w_t^k = G_t$
            if $k \in C_m$ then
                $f = True$
            end
            $w_{t+1}^k \leftarrow ClientUpdate(k, w_t^k, f, t_{global})$
        end
        $G_{t+1} \leftarrow \sum_{k=1}^{K} \frac{w_{t+1}^k}{K}$
    end
End Function
return $G_{t+1}, t_{global}$
```

Algorithm 5: ClientUpdate

```
   Input: Client $k$, Local training dataset $D_{local}$, Current global model $w$, flag $f$, global trigger $t_{global}$
   Output: Updated model $w$
1  Function ClientUpdate():
2      if $f$ is $True$ then
3          $t_{local} \leftarrow GenerateTrigger(s, \rho)$
4          $D_{local} \leftarrow BackdoorDataset(D_{local}, t_{local}, y_t)$
5          $t_{global} = t_{global} \cup t_{local}$
6      end
7      $\mathscr{B} \leftarrow$ (split $D_{local}$ into batches of size B)
8      foreach local epoch $i$ from 1 to $E$ do
9          foreach $b \in \mathscr{B}$ do
10             $w \leftarrow w - \eta \bigtriangledown l(w, b)$
11         end
12     end
13 End Function
14 return $w$
```

Backdoored data is generated (line 4 in Algorithm 5) through the following process, as illustrated in Algorithm 6. We sample subsets of the local training datasets (with non-target labels) with proportion r, and the rest are saved as clean datasets. For each sampled data, we inject a trigger into it by sampling s (trigger size) nodes from the graph uniformly at random and replacing their connection with that in the trigger graph, as shown in Figure 4.9. The node features of the trigger graph are the nodes' one-hot degree. Additionally, the attacker re-labels the sampled data with an attacker-chosen target label. The backdoored data is composed of the dataset with trigger and the original clean

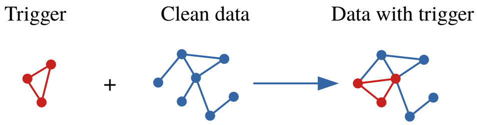


Figure 4.9: Adding trigger into sampled data.

dataset.

Algorithm 6: BackdoorDataset

```
Input: Local Training Dataset $D_{local} = \{x_i, y_i\}_{i=1}^{S}$, Target label $y_t \in [0, C)$, local
       trigger $t_{local}$
Output: Backdoored Training Dataset $D_{backdoor}$
Function BackdoorDataset():
    $D_{trigger} \leftarrow \varnothing$
    $D_{tmp} \leftarrow sample(D_{local}, r, y \neq y_t)$
    $D_{clean} = \{data \in D_{local} : data \notin D_{tmp}\}$
    foreach $d \in D_{tmp}$ do
        $x = AddTrigger(d[x], t_{local})$
        $y = y_t$
        $D_{trigger} = D_{trigger} \cup \{x, y\}$
    end
End Function
$D_{backdoor} = D_{clean} \bigcup D_{trigger}$
return $D_{backdoor}$
```

4.3.3. Experiments

Experimental Setting

We implemented FL algorithms using the PyTorch framework. All experiments were run on a server with 2 Intel Xeon CPUs, one NVIDIA 1080 Ti GPU with 32GB RAM. Each experiment was repeated ten times to obtain the average result. Our code is blinded for review but will be made public.

Datasets. We run experiments on three publicly available datasets: two molecular structure datasets - NCI1 [92], PROTEINS_full [16], and one synthetic dataset - TRIANGLES [68], which is a multi-class dataset. Table 4.12 provides more information about these datasets.

Dataset splits. For each dataset, we randomly sample 80% of the data instances as the training dataset and the rest as the test dataset. To simulate non-i.i.d. training data and supply each participant with an unbalanced sample from each class, we further split the training dataset into K parts following the strategy in [42] with hyperparameter 0.5 for TRIANGLES (10 classes) and hyperparameter 0.7 for other datasets (2 classes). In this section, apart from experiments where we analyze the effect of trigger factors, we set

Table 4.12: Datasets statistics.

Dataset	# Graphs	Avg. # nodes	Avg. # edges	Classes	Class Distribution
NCI1	4,110	29.87	32.30	2	2,053[0], 2,057[1]
PROTEINS_full	1,113	39.06	72.82	2	663[0], 450[1]
TRIANGLES	45,000	20.85	32.74	10	4,500[0 – 9]

trigger factors as follows: $\gamma = 0.2$, $\rho = 0.8$, and $r = 0.2$. As we show in later experiments, these hyperparameters yield an effective attack. By choosing them, we model a strong adversary that helps in evaluating the attack's behavior in the worst-case scenario.

Models and metrics. In our experiments, we use three state-of-the-art GNN models: GCN [67], GAT [127], and GraphSAGE [53].

We use the *attack success rate (ASR)* to evaluate the attack effectiveness, as shown in Algorithm 7. We embed the testing dataset with local triggers or the global trigger and then calculate the ASR of the global model on the poisoned testing dataset. We only embed the testing dataset of the non-target label with triggers to avoid the influence of the original label. The ASR measures the proportion of trigger-embedded inputs that are misclassified by the backdoored GNN into the target class y_t chosen by the adversary. The trigger-embedded inputs are

$$D_{g_t} = \left\{(G_{1,g_t}, y_1), (G_{2,g_t}, y_2), \ldots, (G_{n,g_t}, y_n)\right\}.$$

Here, g_t is the graph trigger, $\left\{G_{1,g_t}, G_{2,g_t} \ldots, G_{n,g_t}\right\}$ is the test dataset embedded with graph trigger g_t, and $y_1, y_2, \ldots, y_n$ is the label set. Formally, ASR is defined as:

$$Attack\,Success\,Rate = \frac{\sum_{i=1}^{n} \mathbb{I}(G_{backdoor}(G_{i,g_t}) = y_t)}{n},$$

where $\mathbb{I}$ is an indicator function and $G_{backdoor}$ refers to the backdoored global model. Here, the graph trigger g_t can be local triggers or a global trigger.

We use the *clean accuracy drop (CAD)* to measure the effect of the backdoor attack on the original task. It is calculated by comparing the performance of the backdoored and clean models on a clean testing set. The accuracy drop should generally be small to keep the attack stealthy. Given the clean inputs

$$D_{clean} = \left\{(G_1, y_1), (G_2, y_2), \ldots, (G_n, y_n)\right\},$$

CAD is defined as:

$$Clean\,Accuracy\,Drop = \frac{\sum_{i=1}^{n} \mathbb{I}(G_{clean}(G_i) = y_i)}{n} - \frac{\sum_{i=1}^{n} \mathbb{I}(G_{backdoor}(G_i) = y_i)}{n}, \quad (4.5)$$

where G_{clean} refers to the clean global model.

Backdoor Attack Results

We evaluate multiple-shot attack [5], which means that the attackers perform attacks in multiple rounds, and the malicious updates are accumulated to achieve a successful backdoor attack. We do not evaluate the single-shot attack [5] because the multi-shot

Algorithm 7: Evaluate Backdoor Attack

```
Input: Global Model $G_t$, Testing Dataset $D_{test}$, Target label $y_t$, graph trigger $g_t$
Output: Attack Success Rate $ASR$
Function Evaluation():
    $D_{tmp} \leftarrow d \in D_{test}$ if $d[y] \neq y_t$
    foreach $d \in D_{tmp}$ do
        $x = AddTrigger(d[x], g_t)$
        $y = y_t$
        $D_{tmp} = D_{tmp} \cup \{x, y\}$
    end
End Function
$ASR = accuracy(G_t, D_{tmp})$
return $ASR$
```

is stealthier [102]. The multi-shot attack does not require multiplying large factors to model weights when conducting the attack, while the single-shot needs to multiply large factors to maintain the effectiveness of backdoor attacks, which can be filtered out or detected by traditional anomaly detection-based approaches such as Krum [12]. Since our main goal is conducting backdoor attacks on FL, we chose a multiple-shot attack with a high attack success rate and stealthiness. As mentioned in Section 4.3.1, we perform a complete attack in every round, showing the difference between DBA and CBA in a shorter time [144].

Cao et al. took into account the situation of backdoor attacks in the malicious majority of clients and proposed a method of defense-*FLTrust* [18]. Before training begins, an honest server collects and trains on a small dataset. The server takes the updates obtained by training on a small dataset as the root of trust in each iteration. It is then compared to the updates uploaded by the clients. If the cosine similarity between them is too small, the updates will be filtered out. With this approach, the accuracy of the global model remains equivalent to that of the baseline. Based on *FLTrust*, Dong et al. considered the setting of two semi-honest servers and malicious majority clients and proposed *FLOD* to ensure that gradients are not leaked on the server side [37]. To explore the impact of different percentages of malicious clients on the attack performance, we evaluate the honest majority and malicious majority attack scenarios according to the percentage of malicious clients among all clients. Specifically, we set two and three malicious clients among five clients for the honest majority and malicious majority attack scenarios, respectively.

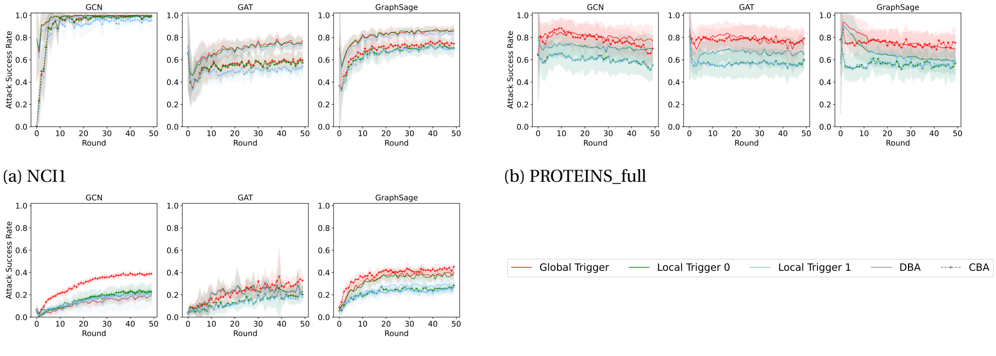
In our experiments, we evaluate the ASR of CBA and DBA with the global trigger and local triggers. The goal is to explore:

- In CBA, whether the ASR of local triggers can achieve similar performance to the global trigger even if the centralized attacker would embed a global trigger into the model.

- In DBA, whether the ASR of the global trigger is higher than all local triggers even if the global trigger never actually appears in any local training dataset, as mentioned in [144].

Honest Majority Attack Scenario. The attack results of CBA and DBA in the honest majority attack scenario are shown in Figure 4.10. Notice that the DBA ASR with a specific trigger is always higher than or at least similar to that of CBA with the corresponding trigger. For example, in Figure 4.10a (the result for the GAT model), the DBA ASR with the global trigger is higher than CBA with a global trigger. The only exception happens for GCN on TRIANGLES. We also find that the ASR of the two attacks in TRIANGLES is significantly lower than the other two datasets but still higher than random guessing. The TRIANGLES is a multi-class dataset containing complex data relations. Thus, more information needs to be encoded in each model's weights for the class features compared to the other datasets. As a result, there is not enough remaining space to learn our triggers easily. In most results on NCI1 and PROTEINS_full, there is an initial drop in the attack success rate for both DBA and CBA, resulting from the high local learning rate of honest clients [5]. *Based on the result for CBA, surprisingly, the ASR of all local triggers can be as high as the global trigger even if the centralized attacker embeds the global trigger into the model, which is inconsistent with the behavior in [144].* We analyze it through further experiments shown in Figure 4.18.

Moreover, the results for the PROTEINS_full dataset show that *in DBA, the attack success rate of the global trigger is higher than (or at least similar to) any local trigger, even if the global trigger never actually appears in any local training dataset.* This indicates that the high attack success rate of the global trigger does not require the same high attack success rate of local triggers. However, for the other two datasets (NCI1 and TRIANGLES), the attack success rate of the global trigger is close to all local triggers (except the result of GraphSage on TRIANGLES). This indicates that in some cases, the local trigger embedded in local models can successfully transfer to the global model so that once any local trigger is activated, the global model will misclassify the data sample into the attacker-chosen target label. This phenomenon is not consistent with the observations in [144] as in Euclidean data, most locally triggered images are similar to the clean image, but any (small) change in the structure of a graph will result in a significant dissimilarity.



(a) NCI1

(b) PROTEINS_full

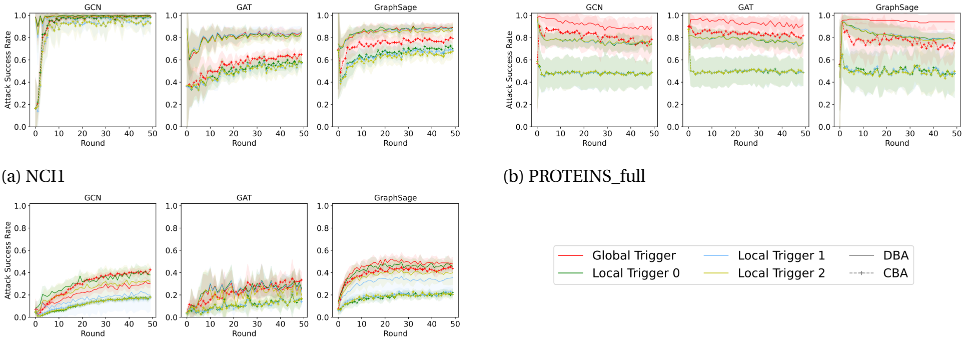
(c) TRIANGLES

(d) Legend

Figure 4.10: Backdoor attack results in the honest majority attack scenario.

Malicious Majority Attack Scenario Figure 4.11 illustrates the attack results in the malicious majority attack scenario. Compared with the honest majority attack scenario, in most cases, the attack success rate of DBA and CBA increases as with more malicious clients, more malicious updates are uploaded to the global model, making the attack more effective and persistent. Moreover, the increase in DBA is more significant than in CBA. For instance, based on the NCI1 dataset and GAT model, the DBA ASR with the global trigger in the honest majority attack scenario is 17.54% higher than CBA, while in the malicious majority attack scenario, the ASR difference is 20.65%. Thus, increasing the number of malicious clients is more beneficial for DBA than CBA. With more malicious clients, more local models are used to learn the trigger patterns in DBA, while there is only one malicious local model in CBA.

For CBA, the ASR with the global trigger is higher while the attack performance with local triggers stays at a similar level or even decreases. One possible reason is that more malicious clients mean a larger global trigger, requiring more learning capacity of the model. If there is not enough learning capacity for every local trigger in the global trigger, the backdoored model can have poor attack performance with a specific local trigger but will behave well with the union set of the local triggers, i.e., the global trigger.



(a) NCI1

(b) PROTEINS_full

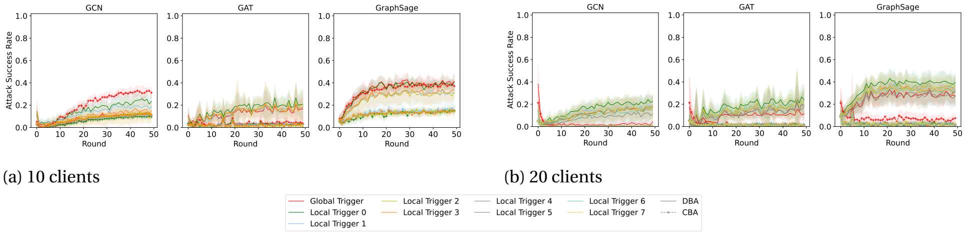
(c) TRIANGLES

(d) Legend

Figure 4.11: Backdoor attack results in the malicious majority attack scenario.

Impact of the Number of Clients We only set the number of clients as 5 for these graph datasets because some of these datasets, i.e., NCI1 and PROTEINS_full, are small (less than 5,000 graphs). However, to explore the impact of the number of clients on DBA and CBA, we also conduct experiments with more clients on the largest dataset - TRIANGLES. We set the number of clients as 10 and 20 and keep the ratio of malicious clients among the total clients the same as before, i.e., 0.4 and 0.6 for the honest majority and malicious majority attack scenarios, respectively. Here, we provide the results of the honest majority attack scenario, as shown in Figure 4.12. The results of the malicious majority attack scenario are given in Figure 4.13, and the phenomenon between the two attack scenarios with 10 and 20 clients is similar to that with 5 clients.

It is obvious that with the increase in the number of clients, the attack success rate of CBA decreases dramatically while the attack performance of DBA keeps steady. This is reasonable because, in CBA, there is only one malicious client whose malicious up-

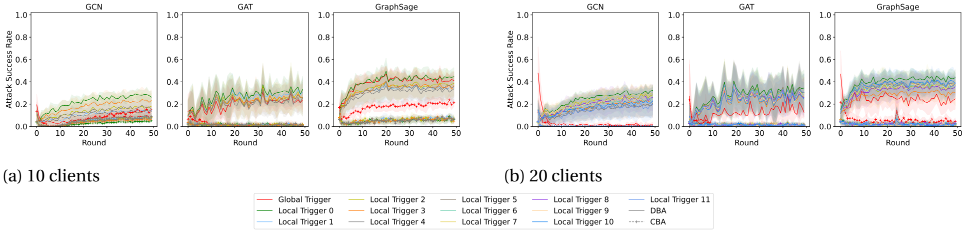


(a) 10 clients

(b) 20 clients

(c) Legend

Figure 4.12: Backdoor attack results of TRIANGLES with more clients in the honest majority attack scenario.



(a) 10 clients

(b) 20 clients

(c) Legend

Figure 4.13: Backdoor attack results of TRIANGLES with more clients in the malicious majority attack scenario.

dates contribute less to the global model with more clients in total. On the contrary, in DBA, the proportion of malicious clients among total clients is the same, meaning the malicious updates contribute the same to the global model regarding the different number of clients. Therefore, as shown in Figures 4.12a and 4.12b, the number of clients has negligible impact to the DBA.

Impact of the Percentage of Malicious Clients Although we have analyzed the experiments with the honest majority and malicious majority scenarios, we further explore the impact of the percentage of malicious clients on the attack performance by calculating their Pearson Correlation Coefficient (PCC), as shown in Figure 4.14, (we provide the results for the GraphSage model as the example as they are more stable, and the results of other models are aligned). Recall that M represents the number of malicious clients, and each number over the line is the corresponding PCC. As we can see, for all datasets, PCC in DBA is larger than CBA, meaning the increase in M has a more positive impact on DBA than CBA. This is intuitive as more malicious clients in DBA lead to more local models embedded with local triggers, while in CBA, it means a larger global trigger due to more local triggers. Specifically, in DBA, more malicious clients mean more model weights to learn the trigger. In CBA, there is only one attacker, and learning a larger global trigger can be out of the model's representation capability. Additionally, as we keep the poisoning intensity of DBA and CBA the same for each malicious client, there are more poisoned training data in DBA than in CBA as more malicious clients are used.

We also explore the attack performance with more clients and less percentage of malicious clients on the large dataset - TRIANGLES. Figure 4.15 shows the attack results

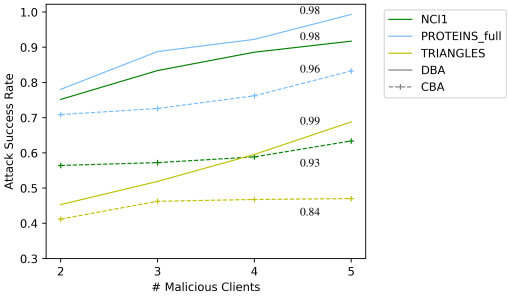


Figure 4.14: Correlation between ASR and M.

on TRIANGLES with 100 clients and fewer malicious clients, ranging from 5% to 20% (here, we also take the results of the GraphSage model as the example, the results of other models are presented in Figure 4.16 and 4.17). Table 4.13 illustrates the specific attack results. We can see from Figure 4.15 that DBA's ASR gradually increases with more malicious clients while CBA's ASR stays below 10%, further verifying that the increase in M has a more positive impact on DBA than CBA. From Figure 4.15, we can still observe that with 20% malicious clients, DBA can also achieve ASR of more than 20%, which means with less percentage (e.g., 20%) of malicious clients, the DBA is still effective. With more clients in total, the attack performance of CBA decreases, consistent with the observation in Figure 4.12. Thus, adding more clients does not change our previous conclusions (with 5 clients).

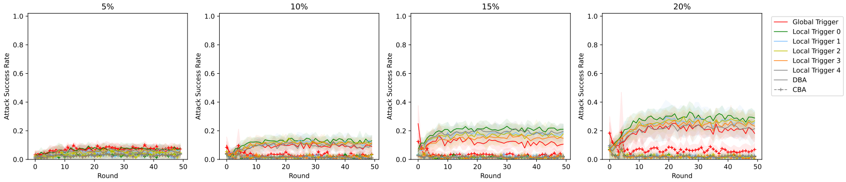


Figure 4.15: Backdoor attack results of TRIANGLES with less percentage of malicious clients ($K = 100$, GraphSage).

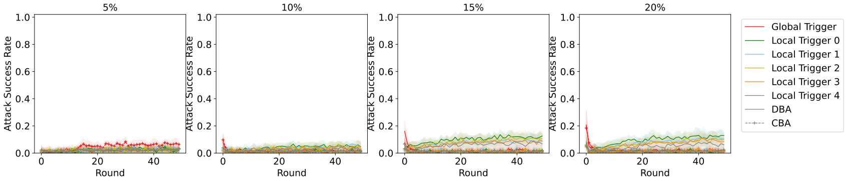


Figure 4.16: Backdoor attack results of TRIANGLES with less percentage of malicious clients ($K = 100$, GCN).

Analysis of CBA results In Figure 4.10, for CBA, the attack success rate of all local triggers can be as high as the global trigger, which is counterintuitive as the centralized

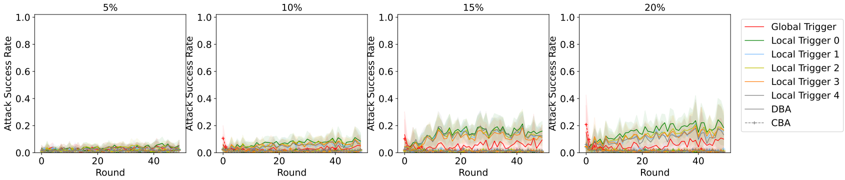


Figure 4.17: Backdoor attack results of TRIANGLES with less percentage of malicious clients (K = 100, GAT).

Table 4.13: Attack success rate of CBA and DBA with less percentage of malicious clients in TRIANGLES (K=100, GraphSage).

Model	Attack Success Rate (CBA% | DBA%)			
	5%	10%	15%	20%
GCN	6.50 | 2.03	1.36 | 1.39	1.28 | 1.06	1.52 | 1.95
GAT	1.48 | 3.91	1.11 | 3.65	1.22 | 6.97	1.51 | 8.01
GraphSage	7.64 | 6.89	2.85 | 8.80	2.87 | 10.93	5.53 | 20.49

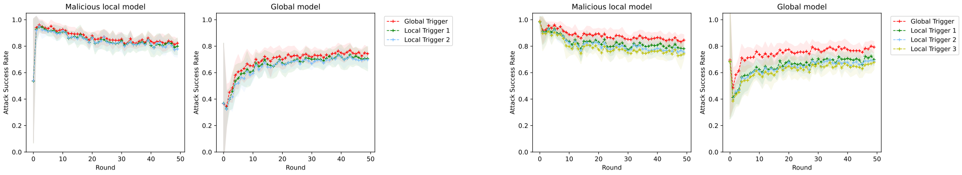
attack only embeds the global trigger into the model. To explain these results, we further implement an experiment (NCI1 on GraphSage model) where we evaluate the attack success rate of the global trigger and local triggers in both the malicious local model [8] and the global model. As shown in Figure 4.18, in the malicious local model, the ASR of all local triggers is already close to the global trigger, which means that the malicious local model has learned the pattern of each local trigger. After aggregation, the global model inherits the capacity of local models. Once any local trigger exists, the global model will misclassify the data sample into the attacker-chosen target label.

Still, in [144], for the CBA, the attack success rate of all local triggers is significantly lower than the global trigger. There, the malicious local model learns the global trigger instead of each local trigger, so the poisoned model can only misclassify the data sample once there is a global trigger in the data. The different results in CBA between [144] and our work can be explained since there, the local triggers composing the global trigger are located close to each other (i.e., less than three pixels distance). In our work, the location of local triggers is random since a graph is non-Euclidean data where we cannot put nodes in some order. When the local trigger graphs are further away from each other, the malicious local model in CBA can only learn the local trigger instead of the global trigger.

Analysis of Backdoor Hyperparameters

This section studies the backdoor hyperparameters discussed in Section 4.3.1. We only modify one factor for each experiment and keep other factors as in Section 4.3.3. We provide results for TRIANGLES and the GraphSage model as an example as those results are more stable, i.e., have the smallest standard error, and the results of other models are aligned. For each factor, we evaluate the global trigger's ASR and the test accuracy on the clean test dataset. We illustrate the results on TRIANGLES in two attack scenarios to analyze the effects of each factor for DBA and CBA. The results are shown in Figure 4.19.

[8] For the CBA, we assume there is one centralized attacker, so there is only one local model that will be poisoned and we define this model as the malicious local model



(a) Honest majority attack scenario

(b) Malicious majority attack scenario

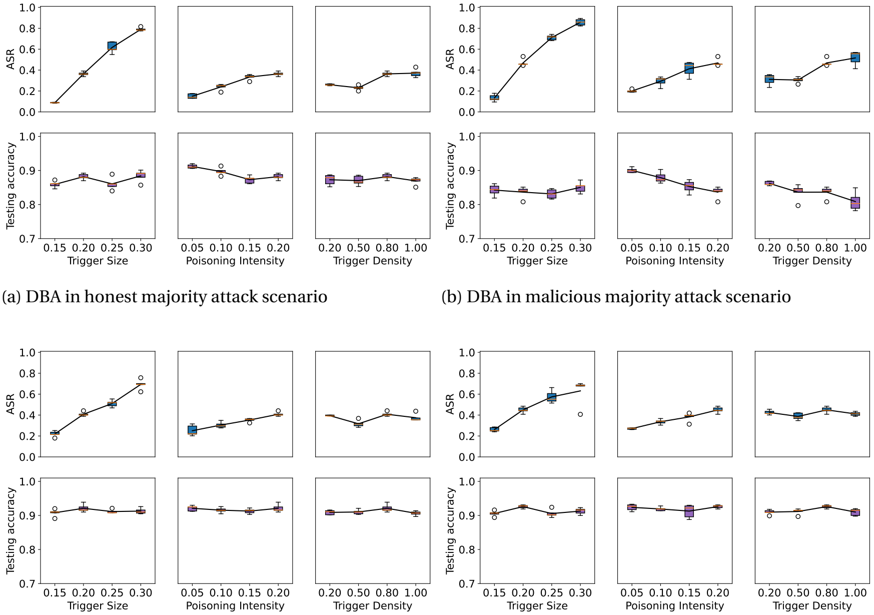
Figure 4.18: Centralized backdoor attack results on the malicious local model and global model with different triggers.

Effects of Trigger Size From the ASR results in Figure 4.19, for both attacks and attack scenarios, with the increase of trigger size, the attack success rate rises significantly, e.g., the DBA's ASR increases from 0.09 to 0.80 with trigger size rising from 0.15 to 0.30 (honest majority attack scenario). There is no significant effect of trigger size on the test accuracy of the global model, implying that the trigger size has little impact on the original main task.

Effects of Poisoning Intensity Similar to the impact of trigger size on the attack success rate, a higher poisoning intensity gives a higher attack success rate. Intuitively, a backdoor attack can perform better with more poisoned data. Nevertheless, the increase is less significant than that of different trigger sizes. Specifically, in comparison with [143], where there is no obvious difference between the impact of poisoning intensity and trigger size, here, a larger trigger size has a more positive influence on ASR than a larger poisoning intensity. We consider this an interesting observation and plan to investigate it in future work. Moreover, in DBA, the test accuracy decreases with the increasing poisoning intensity, and with more malicious clients, the drop is more significant, as shown in Figures 4.19a and 4.19b. This can be explained as with higher poisoning intensity, and more malicious clients, more model weights (including some for the original task) are influenced by the malicious trigger patterns, and the performance on the main task degrades more. We can also observe that with higher poisoning intensity, there is no obvious drop in the testing accuracy for CBA, as presented in Figures 4.19c and 4.19d. Although more local data is poisoned, the other honest clients (the majority part) still guarantee the performance on the main task.

Effects of Trigger Density From Figure 4.19b, DBA's ASR improves from 30.10% to 47.96% when the trigger density increases from 0.50 to 0.80. This is because the average complexity of the TRIANGLES dataset is 0.16 [104]. Thus, when the trigger density is set close to this value, the difference between the original graph and the trigger graph is harder to distinguish. However, the effect of the trigger density in CBA's ASR is not strong. We see a slight fluctuation as the trigger density increases, but its range is very small to be considered a trend. In CBA, we use only one malicious client, and the weak effect of the trigger density is smoothed by the averaging operation.

In Figure 4.19, in most cases, there is no significant drop in the test accuracy with an increase in the trigger size and trigger density. On the contrary, in the backdoor attacks in centralized GNNs [145], as trigger size increases, the test accuracy decreases. This can be explained as, in FL, the influence of backdoor functionality on the main task is weakened by the aggregation of local models.



(a) DBA in honest majority attack scenario

(b) DBA in malicious majority attack scenario

(c) CBA in honest majority attack scenario

(d) CBA in malicious majority attack scenario

Figure 4.19: Results on TRIANGLES with different trigger parameters.

Clean Accuracy Drop

The goal of the backdoor attack is to make the backdoored model simultaneously fit the main task and backdoor task. Therefore, it is critical that the trained model still behaves normally on untampered data samples after training with the poisoned data. Here, we use *clean accuracy drop (CAD)* to evaluate if the backdoored model can still fit the original main task. CAD is the classification accuracy difference between global models with and without malicious clients over the clean testing dataset. CBA's and DBA's final clean accuracy drop results in the honest and malicious majority attack scenarios are given in Tables 4.14 and 4.15, respectively. In most cases, both attacks have a low CAD, i.e., around 2%, and only in a few cases is there a significant CAD. These results imply that, in most cases, both attacks have a negligible impact on the original task of the model. Additionally, in some cases, DBA's CAD is significantly higher than CBA's, e.g., DBA's CAD is 5.74% in the GraphSage model on TRIANGLES while CBA's is 3.24%, as shown in Table 4.15. At the same poisoning intensity for each client, there are more poisoned data in DBA than in CBA, leading to worse performance in the main task. The substantial clean accuracy drop in DBA can also be observed in [144].

Attack performance in Real-world Social Network Datasets

Two real-world social network datasets We already used two molecular structure datasets

Table 4.14: Clean accuracy drop of CBA and DBA in the honest majority attack scenario.

Dataset	Clean Accuracy Drop (CBA% | DBA%)		
	GCN	GAT	GraphSage
NCI1	2.54 | 2.01	0.84 | 1.42	0.93 | 0.16
PROTEINS_full	1.81 | 4.06	0.49 | 0.46	2.31 | 2.82
TRIANGLES	0.01 | 1.32	3.71 | 2.87	3.31 | 4.45

Table 4.15: Clean accuracy drop of CBA and DBA in the malicious majority attack scenario.

Dataset	Clean Accuracy Drop (CBA% | DBA%)		
	GCN	GAT	GraphSage
NCI1	4.45 | 2.74	1.03 | 1.07	1.29 | 2.22
PROTEINS_full	2.78 | 1.30	0.03 | 2.65	2.72 | 4.59
TRIANGLES	0.14 | 0.30	5.10 | 5.84	3.24 | 5.74

and one synthetic dataset to explore the distributed backdoor attacks and centralized backdoor attacks in the Federated GNNs. However, in order to further explore the DBA and CBA in real-world applications and scenarios, we in this section extend our analysis of DBA and CBA to two additional real-world social network datasets. In recent years, with the rapid development of internet technology, social network graphs have played an increasingly important role in people's lives [51]. These graphs contain valuable information regarding individuals' behavior and interactions, making them applicable in various domains such as sentiment analysis, recommendation systems, and spam detection. Federated GNNs can also be employed in these areas. Nonetheless, the vulnerability of Federated GNNs to backdoor attacks poses significant risks. For instance, in the scenario of applying the Federated GNNs on spam detection, if the attacker executes a backdoor attack on the system, the model will detect any input graph with a specific trigger as benign, potentially leading to severe safety issues. Consequently, it is crucial to also explore the attack performance of the Federated GNNs on social network datasets. What is more, these two social network datasets have different graph characteristics from the aforementioned two molecular structure datasets and one synthetic dataset. Specifically, these two datasets have a larger average number of edges, e.g., the average number of edges of COLLAB is 2,457.78 while that of NCI, PROTEINS_full, and TRIANGLES datasets are 32.30, 72.82, and 32.74 respectively. Through experiments on these two social network datasets, it can be verified that our attacks are effective not only on simple datasets but also on complex datasets.

- COLLAB. This scientific-collaboration dataset is derived from three public collaboration datasets, namely, *High Energy Physics, Condensed Mater Physics*, and *Astro Physics*. It consists of ego-networks [9] of different researchers from each field, with each graph labeled according to the researcher's field. The task is to determine whether the ego-collaboration graph of a researcher belongs to High Energy, Condensed Matter, or Astro Physics field.

- IMDB-BINARY. This dataset is a movie-collaboration dataset where actor/actress

[9] An ego network is defined as a portion of a social network formed for a given individual, termed *ego*, and the other persons with whom the user has a social relationship, termed *alters* [4].

Table 4.16: Datasets statistics of two social network datasets.

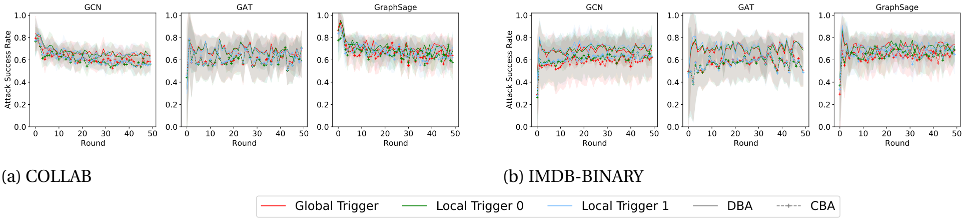
Dataset	# Graphs	Avg. # nodes	Avg. # edges	Classes	Class Distribution
COLLAB	5,000	74.49	2,457.78	3	2600[0], 775[1], 1625[2]
IMDB-BINARY	1,000	19.77	96.53	2	500[0], 500[1]

and genre information of different movies on IMDB was collected. Collaboration graphs are generated based on *Action* and *Romance* genres, and ego-networks are derived for each actor/actress. Similar to the COLLAB dataset, each ego-network is labeled with the genre graph it belongs to. The task is to identify to which genre an ego-network graph belongs. Table 4.16 describes the information about these two social network datasets.

Attack results The attack results of DBA and CBA on two social network datasets in the honest majority attack scenario are presented in Figure 4.20. As seen in the figure, we can observe that for both GCN and GAT models, the ASR of DBA with a specific trigger is consistently higher than that of CBA with the corresponding trigger. For instance, for GAT on IMDB-BINARY, the ASR of DBA with the global trigger is typically 10% higher than that of CBA with the global trigger. However, for the GraphSage model, the ASR of the two attacks is similar. It can be explained that the GraphSage model has a more redundant learning capacity, making it easier for CBA to learn the global trigger pattern. As a result, the ASR of CBA is higher than the other two models and is similar to that of DBA. Similar to the results obtained from the prior datasets, i.e., NCI1, PROTEINS_full, and TRIANGLES, for CBA on the two social network datasets, the ASR of all local triggers can be as high as the global trigger even if the centralized attacker embeds the global trigger into the model. This observation highlights the vulnerability of the GNN models to even the slight structure manipulation in the graph. Furthermore, in DBA, the ASR of the global trigger is also close to all the local triggers, which further indicates that in DBA, the local trigger embedded in local models can successfully transfer to the global model. It implies that the attacker can compromise the global trigger by embedding a simple local trigger into the local models, leading to significant security concerns.

Figure 4.21 shows the attack results of two social network datasets in the malicious attack scenario. In comparison to the honest majority attack scenario, the ASR of DBA increases for GCN and GraphSage models as more malicious clients participate in the attack, more model capacity is used to learn the trigger pattern, and more malicious updates are uploaded into the global model, making it easier to achieve the attack. For example, for the GCN model on COLLAB, the ASR of DBA in the malicious majority attack scenario is around 15% higher than that in the honest majority attack scenario. On the other hand, the ASR of CBA notably decreases in the malicious majority attack scenario, especially for the global trigger in GCN and GraphSage models. One possible explanation for this observation is that in the CBA, there is only one malicious local model and the global trigger is larger in the malicious majority attack scenario than in the honest majority attack scenario, which requires more learning capacity of the model to learn every local trigger. It is likely that the backdoored model cannot learn the local trigger patterns clearly, and therefore, it cannot recognize the global trigger effectively. These findings indicate that DBA is generally more effective than CBA in the malicious majority attack scenario. Moreover, the results demonstrate that the number of malicious clients participating in the attack significantly impacts the effectiveness of the backdoor attacks in the Federated GNNs. It is essential to develop more robust and secure models that can resist these backdoor attacks, especially when the attacker has access to more clients in the federated learning setting.

Clean accuracy drop: Here, we also use CAD to evaluate the impact of backdoor attacks on the original mask task on the social network datasets. The testing accuracy of DBA and CBA on the two social network datasets are shown in Figure 4.22 (honest majority attack scenario) and 4.23 (malicious majority attack scenario). Specifically, the clean accuracy drop results in the honest and malicious majority attack scenarios are given in Table 4.17 and 4.18, respectively. In most cases, the CAD of both attacks is less than 5% for both datasets. Additionally, we observe that the clean accuracy drop of DBA is generally higher than that of CBA, with a difference of more than 3% in most cases. This observation is consistent with the results obtained in the prior experiments.

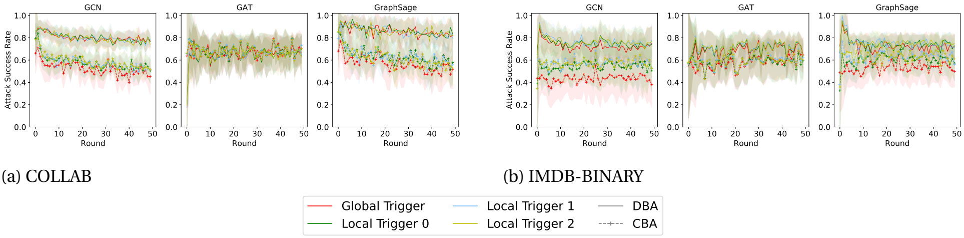


(a) COLLAB

(b) IMDB-BINARY

(c) Legend

Figure 4.20: Backdoor attack results in the honest majority attack scenario (social network datasets).



(a) COLLAB

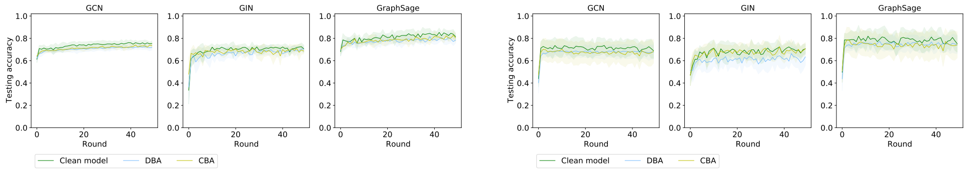
(b) IMDB-BINARY

(c) Legend

Figure 4.21: Backdoor attack results in the malicious majority attack scenario (social network datasets).

Table 4.17: Clean accuracy drop of CBA and DBA in the honest majority attack scenario (social network datasets).

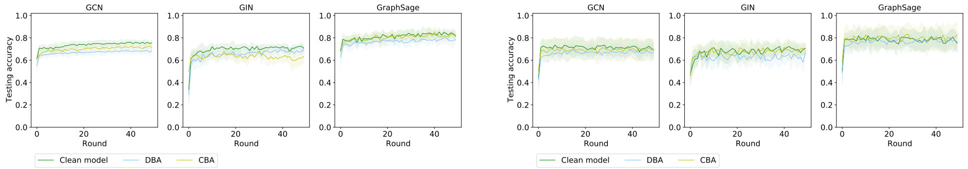
Dataset	Clean Accuracy Drop (CBA% | DBA%)		
	GCN	GAT	GraphSage
COLLAB	2.44 | 3.46	1.75 | 2.27	2.52 | 4.40
IMDB-BINARY	3.75 | 2.89	1.38 | 8.06	4.35 | 2.97



(a) COLLAB

(b) IMDB-BINARY

Figure 4.22: Testing accuracy in the honest majority attack scenario (social network datasets).



(a) COLLAB

(b) IMDB-BINARY

Figure 4.23: Testing accuracy in the malicious majority attack scenario (social network datasets).

Table 4.18: Clean accuracy drop of CBA and DBA in the malicious majority attack scenario (social network datasets).

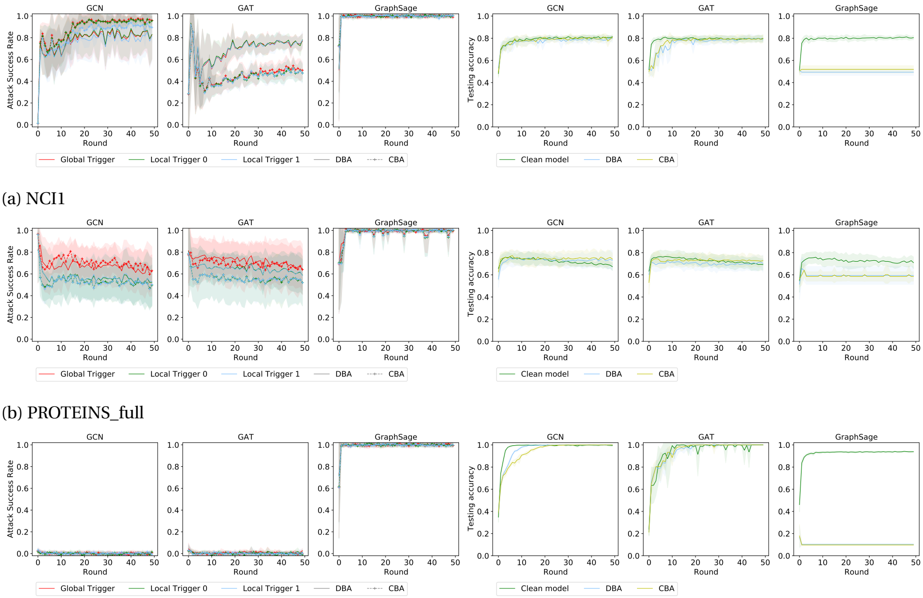
Dataset	Clean Accuracy Drop (CBA% | DBA%)		
	GCN	GAT	GraphSage
COLLAB	3.83 | 7.37	9.07 | 3.16	1.14 | 5.09
IMDB-BINARY	1.22 | 3.58	0.74 | 6.72	2.37 | 0.61

4.3.4. DEFENSES

Potential Countermeasures

FLAME [95] is one of the state-of-the-art defenses against backdoor attacks in FL, combining the benefits of both defense types (Byzantine-robust aggregation mechanisms and differential privacy techniques) to eliminate the impact of backdoor attacks while maintaining the performance of the aggregated model on the main task. FoolsGold [43] is a robust FL aggregation algorithm that can identify attackers in federated learning based on the diversity of client updates. It reduces the aggregation weights of detected malicious clients while retaining the weights of other clients. One of the assumptions in this defense is that each client's training data is non-i.i.d and has a unique distribution, which fits the non-i.i.d data distribution setting in this section. Ozdayi et al. also proposed a defense mechanism against backdoor attacks in federated learning [99]. The idea is to adaptively adjust the learning rate of the server's aggregation function, per dimension and per round, based on the sign information of agents' updates. In this work, we focus on evaluating the attack effectiveness of DBA and CBA against FoolsGold and RLR (Robust Learning Rate).

Results against FLMAE



(a) NCI1

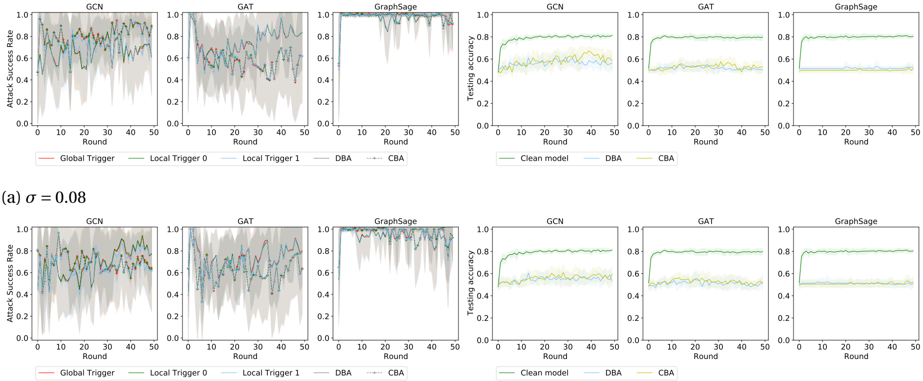
(b) PROTEINS_full

(c) TRIANGLES

Figure 4.24: Backdoor attack results against FLAME (in the honest majority attack scenario).

Figure 4.24 shows the attack performance of the NCI1, PROTEINS_full, and TRIANGLES datasets under FLAME in the honest majority attack scenario (the results in the malicious majority attack scenario are similar). One important parameter in FLAME is the Gaussian noise level σ, and σ is set to be 0.01 for Figure 4.24 as default. Compared with the attack performance before FLAME (Figure 4.10), for GCN and GAT models, the ASR of both DBA and CBA decreases around 10% for NCI1 and PROTEINS_full datasets while for TRIANGLES dataset, the ASR even decreases to nearly 0. However, we can also observe that for the GraphSage model, the ASR under FLAME increases for all datasets, and the testing accuracy decreases dramatically. Given that for PROTEINS_full and NCI1 datasets, the attack performance under FLAME does not degrade obviously, we further evaluate the attack performance under FLAME on PROTEINS_full and NCI1 datasets with different Gaussian noise levels, as shown in Figure 4.25 and 4.26. It can be observed that with a higher Gaussian noise level, there is more fluctuation in the attack performance under FLAME, i.e., more standard deviation. Generally, with a larger Gaussian noise level, the ASR under FLAME for these two datasets decreases while the testing accuracy also reduces significantly. Based on the experimental results against FLAME, we find that FLAME can indeed degrade the attack performance of DBA and CBA, especially for TRIANGLES datasets with GCN and GAT models. However, generally, it can only reduce the ASR of both attacks for less than 10% for PROTEINS_full and NCI1 datasets, and it does not work for the GraphSage model.

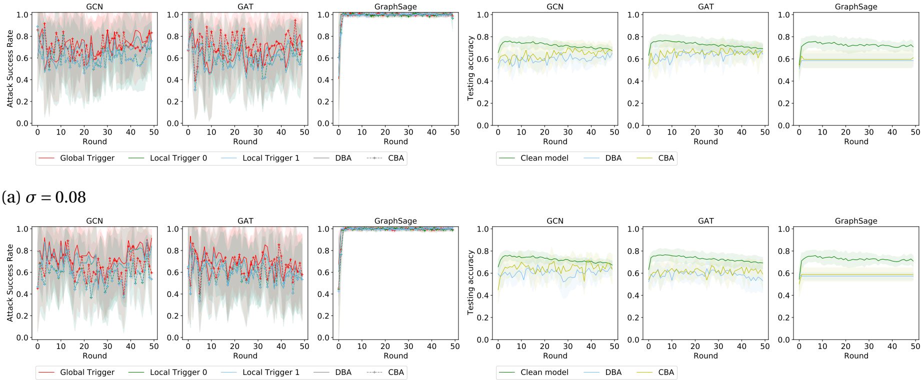
Results against FoolsGold



(a) $\sigma = 0.08$

(b) $\sigma = 0.10$

Figure 4.25: Backdoor attack results against FLAME on NCI1 with different σ (in the honest majority attack scenario).



(a) $\sigma = 0.08$

(b) $\sigma = 0.10$

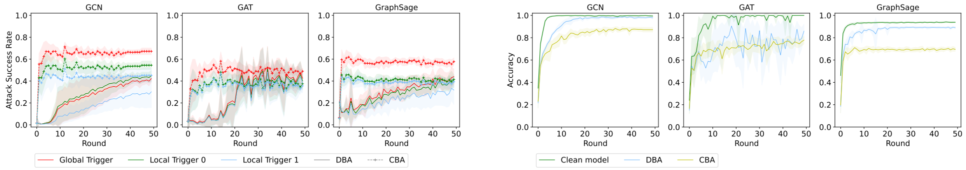
Figure 4.26: Backdoor attack results against FLAME on PROTEINS_full with different σ (in the honest majority attack scenario).

Figure 4.27 shows the attack performance for the TRIANGLES dataset under FoolsGold in the honest majority attack scenario (the results in the malicious majority attack scenario are similar). As illustrated in Figure 4.27, we can observe that for DBA, the ASR under FoolsGold remains consistent for the GraphSage model, whereas it rises along with a decrease in testing accuracy for the GAT model. Moreover, generally, under FoolsGold, there is a significant increase in CBA's ASR in all models, but the testing accuracy of CBA reduces significantly at the same time. For example, the CBA's ASR increases by about 20% for the GraphSage model. However, the testing accuracy of CBA on GraphSage has a drop of more than 20% (Figure 4.27b). Our hypothesis for this situation is that under FoolsGold, the malicious client in CBA is assigned a higher weight (recall the description of the FoolsGold mechanism from the paragraph above) than other clients, so malicious updates contribute more to the aggregated model. Simultaneously, the low weights on the honest clients' updates lead to the failure of the performance on the original task. To verify this hypothesis, we reported FoolsGold's weights on every client in DBA and CBA in Table 4.19. Here, the FoolsGold weight for each client ranges from 0 to 1. As we can see, in CBA, the weight of the malicious client is 1, and the weights of other clients are 0, which means only the malicious updates are aggregated into the global model. Therefore, the attack success rate of CBA increases significantly under FoolsGold. On the other hand, in DBA, the weights of the malicious clients are similar to the honest clients, indicating that the malicious and honest updates contribute equally to the aggregated model, as in the aggregation function without the defense, i.e., the average aggregation function. Therefore, there is no obvious difference between the attack performance of DBA before and after the defense. One possible reason is that in CBA, there is only one malicious client whose updates are likely to appear dissimilar from those of other honest clients, so FoolsGold cannot identify the malicious updates successfully.

The ASR under FoolsGold on NCI1 and PROTEINS_full datasets (honest majority attack scenario) are shown in Figures 4.28 and 4.29, respectively. There is a slight increase in the attack success rate of DBA and CBA under FoolsGold, which indicates that this defense fails to identify the malicious updates and misclassifies them as benign. The graph data are not Euclidean data, e.g., images, so the slightly different subgraphs used as triggers do not induce aligned updates. As a result, the cosine similarity cannot be used to detect malicious clients based on their updates. Even though there are more malicious clients in the malicious majority scenario and the probability of detecting the malicious updates should be higher, we observe the same behavior. This further verifies our hypothesis that the defense based on cosine similarity between updates is not very effective in the graph domain. The clean accuracy drop under FoolsGold on these two datasets is similar to that without the defense.

Based on the experimental results against FoolsGold, we find that the tested defense cannot detect malicious updates successfully. One reason may be that this defense applies *cosine distance* to try to identify malicious models, i.e., the distance between malicious updates is smaller between honest updates. Still, in our attacks, the malicious clients' updates could already be very dissimilar to each other, so the malicious updates are likely to be clustered into honest updates.

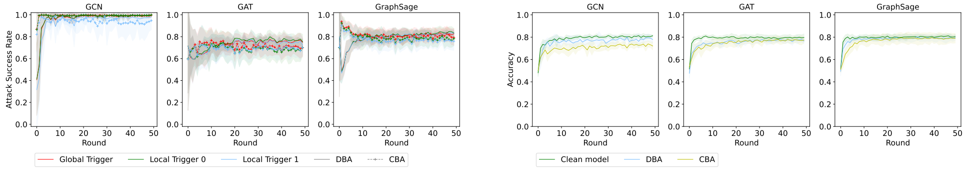
Results against RLR



(a) Attack success rate

(b) Testing accuracy

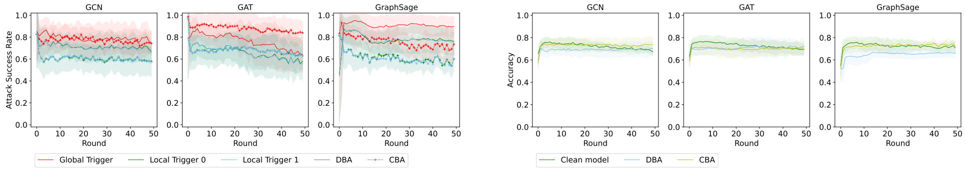
Figure 4.27: Backdoor attack results of TRIANGLES on FoolsGold for the honest majority.



(a) Attack success rate

(b) Testing accuracy

Figure 4.28: Attack success rate on NCI1 on FoolsGold (in the honest majority attack scenario).



(a) Attack success rate

(b) Testing accuracy

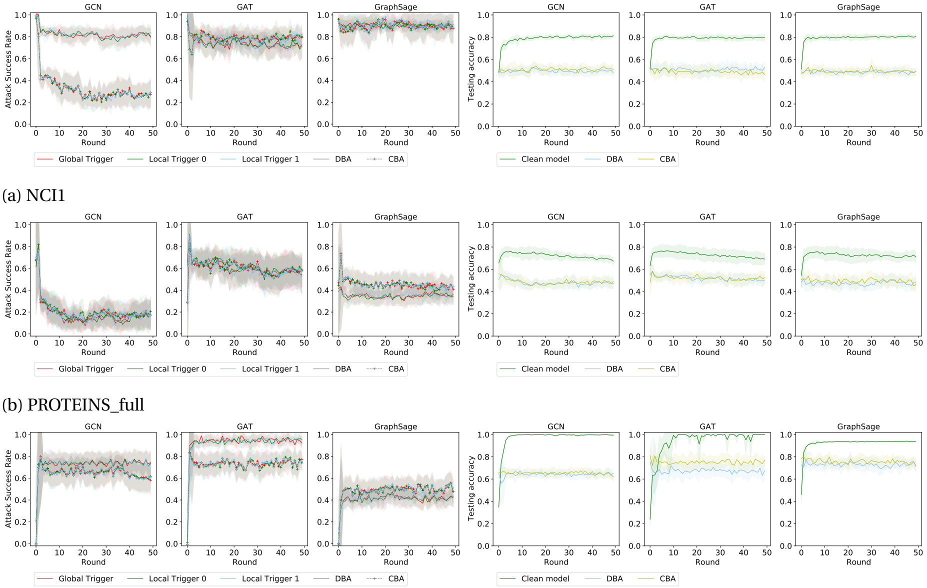
Figure 4.29: Attack success rate on PROTEINS_full on FoolsGold (in the honest majority attack scenario).

Table 4.19: FoolsGold weight in DBA and CBA on TRIANGLES (honest majority attack scenario).

Attacks	Attacker 1	Attacker 2 (client 2 in CBA)	Client 3	Client 4	Client 5	Attackers (sum)
DBA	1.00 ± 0.00	1.00 ± 0.00	0.82 ± 0.15	0.81 ± 0.09	0.78 ± 0.12	2.00 ± 0.00
CBA	1.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	1.00 ± 0.00

The attack performance of NCI1, PROTEINS_full, TRIANGLES against RLR is shown in Figure 4.30 (honest majority attack scenario) and 4.31 (malicious majority attack scenario) including the attack success rate and testing accuracy. The results reveal some interesting insights, and the ASR of DBA and CBA against RLR varies significantly across different datasets and models. In the honest majority attack scenario, compared to the attack performance before the defense, as shown in Figure 4.10, we can observe that the ASR of DBA and CBA decreases for the NCI1 dataset in the GCN model. For example, the ASR of CBA decreases from around 100% to 20%, while it increases for GAT and GraphSage models. On the other hand, for the PROTEINS_full dataset, the ASR of both attacks decreases dramatically, i.e., the decrease is up to 60% compared to Figure 4.10b, for all models. In contrast, for the TRIANGLES dataset, the ASR of DBA and CBA surprisingly increases after RLR. Compared to the NCI1 and PROTEINS_full datasets, the TRIANGLES dataset has fewer average nodes, which makes it more challenging to create a backdoor trigger pattern. Thus, as we can see from Figures 4.10 and 4.11, TRIANGLES has lower ASR than other datasets. Moreover, the undistinguished backdoor trigger pattern in the TRIANGLES dataset leads to the inefficiency of RLR. Another intriguing discovery is that the testing accuracy of all datasets and models reduces significantly under RLR, dropping to around 50% for NCI1 and PROTEINS_full, which is random guessing for these two datasets. The large testing accuracy drop here is not consistent with that in [99]. One possible reason behind this observation is that in the RLR defense, the malicious updates are generally detected as benign and then assigned a positive learning rate. In contrast, benign updates are detected as malicious ones and assigned a negative learning rate. Therefore, the malicious updates contribute more to the aggregated global model, leading to a higher attack success rate and poorer performance on the original task. Moreover, RLR only works under the assumption that the number of malicious clients is sufficiently below θ (learning threshold) [99]. However, in our threat model, the number of malicious clients can be higher than θ.

When there are more malicious clients (Figure 4.31), the attack success rate of DBA decreases dramatically in most cases compared to the honest majority attack scenario, while the ASR of CBA remains relatively stable. For instance, the ASR of DBA in the TRIANGLES in the malicious majority attack scenario is around 36% while that in the honest majority attack scenario is around 75%. It may be explained that in CBA, there is only one malicious local model, while in DBA, there are multiple malicious local models. Thus with an increase in the number of malicious clients, the malicious updates in DBA increase more than in CBA. This makes it easier for RLR to detect malicious updates in DBA, resulting in a decrease in DBA's ASR. Despite the decrease of ASR in DBA under RLR in the malicious majority attack scenario, the clean accuracy drop under RLR is still very high in both attacks, i.e., more than 25% in most cases. Our experimental results against RLR indicate that this defense mechanism is not effective in detecting malicious



(a) NCI1

(b) PROTEINS_full

(c) TRIANGLES

Figure 4.30: Backdoor attack results against RLR (in the honest majority attack scenario).

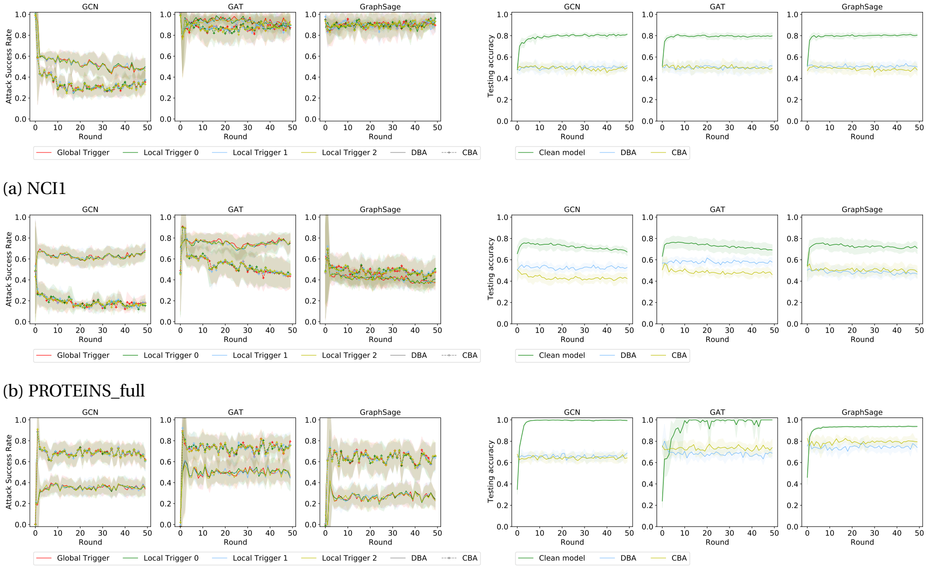
updates, and it can significantly degrade the original main task.

4.3.5. GENERAL TAKEAWAYS ON THE EXPERIMENTAL ASPECTS

Here, we detail and reflect on the experimental aspects of our two backdoor attacks, i.e., DBA and CBA, in the federated GNNs, and draw general takeaways. Specifically, we introduce technical details, present how we have designed the experiments to comprehensively evaluate and compare the attack performance of these two attacks, and especially how we designed the global and local triggers' generation. We also present the implementation details for our two attacks, along with general discussions on *availability* of artifacts and *reproducibility* of the results. We finally report our failures and unexpected results when running or reproducing our experiments, together with solutions we devised to solve the problem and lessons learned from the unexpected results. During our exposition, we aimed to provide the reader with a summary of the lessons learned when implementing the DBA and CBA in the Federated GNNs. We expect that other researchers and practitioners may use those lessons to learn from our experimental methodologies, execution, and results.

Experiments Design

Experimental Setting The settings of different experiments on backdoor attacks in the federated GNNs are summarized in Table 4.10. To eliminate our implementation mistakes, we followed several steps for both the training and backdoor attack perspectives. First, during the model training, we selected to implement the small dataset first and



(a) NCI1

(b) PROTEINS_full

(c) TRIANGLES

Figure 4.31: Backdoor attack results against RLR (in the malicious majority attack scenario).

then the large dataset as it takes less time to load the small dataset and train it. Once the training for a small dataset had no errors, we continued training for the large dataset, e.g., TRIANGLES. Second, in the implementation of a backdoor attack, we first run experiments with extremes (large poisoning rate and large trigger size) to make sure that the poisoning pipeline works and then use more realistic poisoning hyperparameters. We also implemented backdoor attacks in the centralized setting first and then moved to the distributed setup because training in the distributed setting is more complex.

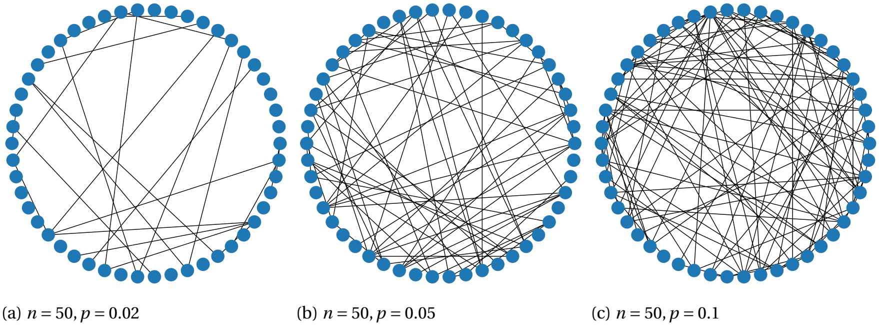
Non-i.i.d. Distribution We followed [18] to distribute the training dataset among clients. Assuming there are M classes in a dataset, we randomly split the clients into M groups. A training example with label l was assigned to group l with probability $p > 0$ and to any other groups with probability $\frac{1-p}{M-1}$. Within the same group, data was uniformly distributed to each client. p controlled the distribution difference of the clients' local training dataset. If $p = \frac{1}{M}$, then the clients' local training datasets are independent and identically distributed (i.i.d.). Otherwise, the clients' local training datasets are non-i.i.d. Moreover, a larger p indicates a higher degree of non-i.i.d. among the clients' local training dataset. In our work, we set $p > \frac{1}{M}$ to simulate the non-i.i.d. setting.

Trigger Generation To compare the attack performance between the distributed backdoor attack and centralized backdoor attack in Federated GNNs, we need to ensure the trigger pattern in CBA is the union set of local trigger patterns in DBA. We can use two strategies: 1) first generate local triggers in DBA and then combine them to get the global trigger, or 2) first generate a global trigger in CBA and then divide it into M local triggers. We utilize the first strategy as it is an NP-hard problem to divide a graph into several subgraphs [32]. Thus, in different attack scenarios (i.e., honest majority or malicious majority attack scenarios), the CBA performance is different since the global trigger has been changed due to the different number of malicious clients. Moreover, DBA should be implemented before CBA so that the local triggers used in DBA can be saved and combined as the global trigger in CBA.

To obtain the local triggers, we adopted the Erdős-Rényi (ER) model [44]. There are two closely related variants of the ER model, one of which is the $G(n, p)$ model. In the $G(n, p)$ model, a graph with n nodes is constructed by randomly connecting nodes. Each edge is included in the graph with probability p, independently from every other edge. An example of ER graph with 50 nodes at different edge probabilities, $p = 0.02, 0.05, 0.1$ is shown in Figure 4.32. A trigger graph generated using `networkx` [52] is shown in Listing 4.1.

```
import networkx as nx
n = 50
p = 0.05
t_g = nx.erdos_renyi_graph(n, p, directed=False)
```

Listing 4.1: Generate trigger graph with networkx


(a) $n = 50, p = 0.02$

(b) $n = 50, p = 0.05$

(c) $n = 50, p = 0.1$

Figure 4.32: Examples of graphs generated by ER model with 50 nodes and different values of p.

Implementation Details

Running Environment We implemented DBA and CBA in the Federated GNNs on PyTorch framework with the version of $1.9.0 + cu111$. All the experiments are run in TU Delft's cluster consisting of CentOS-based worker machines. The architecture of the cluster we used is illustrated in Figure 4.33. There are multiple NVIDIA GPUs in the cluster, such as Tesla P100 and V100, GeForce GTX 1080 Ti and RTX 2080 Ti, and A40. We created a virtual environment with python>=3.6 and installed the dependencies (`torch>=1.9.0`, `torchvision>=0.10.0`, `numpy>=1.23.2`, `dgl>=0.9.1`, `networkx>=2.4`, `hdbscan==0.8.28`, `joblib==1.1.0`). As shown in Figure 4.33 we first need to connect through ssh to a virtual machine in the cluster network (login node). From this machine we can submit our experiments to the cluster's worker nodes. The cluster is shared among different members of the university and for that reason everything is managed by slurm workload manager.[10] Each researcher creates tasks by declaring the resources

[10] https://slurm.schedmd.com/documentation.html

needed (amount of memory, number of CPUs, type of GPU, estimated execution time etc.) and the executable that is going to be run in a bash-like file. This file is used by `sbatch` which is a tool provided by slurm to submit jobs in the cluster. Slurm keeps track of all the pending tasks and according to the resources they need it assigns them to the corresponding worker nodes. The results are saved in a network file system (nfs) that the worker nodes have mounted which is also accessible from the login node.

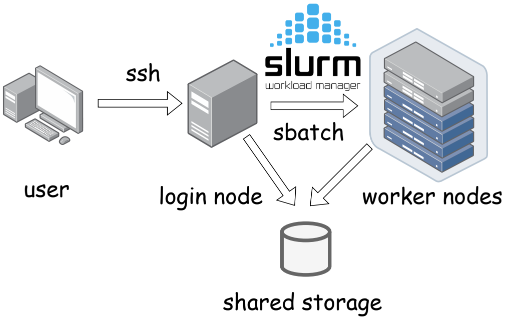


Figure 4.33: Cluster's architecture.

Experimental Setup Our experiments were run on graph datasets from TUDataset [92]. TUDataset is a collection of benchmark datasets for graph classification and regression. We also adopted Deep Graph Library (DGL) [131], which is one of the most popular deep learning libraries for graph data processing to conduct experiments on graph data. DGL provides a powerful graph object that can reside on either CPU or GPU. It provides various functions, from loading graph datasets, processing graph datasets, loading GNN models, to computing with graph objects. For instance, we used three graph datasets (NCI1, PROTEINS_full, TRIANGLES), and they can be loaded as shown in Listing 4.2:

```
import dgl.data
dataset = dgl.data.TUDataset('DataName')
```

Listing 4.2: Import data with dgl

We implemented our GNN models on top of a popular open-source framework used for benchmarking graph neural networks [39]. In addition to evaluating the effectiveness of our attacks in the Federated GNNs, we also evaluated the robustness of our attacks against state-of-the-art defenses. We selected two defenses against backdoor attacks in federated learning. For one defense, we could not find a public release of the code, which means we needed to reproduce it ourselves. Additionally, in our past experience in experimental evaluation, re-implementation is often required, representing extra effort. Moreover, we may fail to respect original implementation decisions because some design choices may be needed to obtain the original results. Thus, researchers in the security field should be encouraged to make their code available and provide documentation for reproduction.

Observation 1

It should be encouraged to make the code available and provide documentation for reproduction.

For the other selected defense, there is open-source code, and the public code is intended for the image domain. Although the selected defenses proved to be effective against the backdoor attacks in the federated learning in the image domain, our experimental results verified that our attacks could not be fully eliminated without influencing the original task. Therefore, what we observed is that the methods proven to be effective in one domain may not work in other domains.

Observation 2

The methods proven to be effective in one domain may not work well in other domains.

Availability and Reproducibility The source code for DBA and CBA in the Federated GNNs has been released on GitHub[11], including the material to reproduce the experimental evaluation. The README file consists of 1) small graph datasets used in our experiments (TRIANGLES dataset is not uploaded but can be downloaded directly from the TUDataset), 2) requirements.txt file to create the virtual environment, 3) python files to run DBA and CBA, 4) pkl files generated during our evaluation, and 5) instructions to reproduce experiments. We also provided python scripts in the GitHub repository, which set the appropriate parameters, e.g., dataset, model config, number of clients, number of malicious clients, results saving path, etc. For example, to train a clean Federated GCN model over the NCI1 dataset, we run a python script:

```
$ python clean_fedgnn.py --dataset NCI1 \\
   --config ./configs/TUS/TUs_graph_classification_GCN_NCI1_100k.json \\
   --num_workers 5 \\
   --num_mali 0  \\
   --filename ./Results/Clean \\
```

Listing 4.3: Script to train a clean Federated GNNs

The results are saved in the folder `./Results/Clean`, which is used to calculate the clean accuracy drop.

To run DBA or CBA, the value of some arguments in the python script is changed, e.g., the number of malicious clients and the results saving path. The argument "config" defines the GNN architecture and training details, such as the number of layers in the GNN model, dropout rate, etc.

During running, the generated local triggers will be saved and used in later experiments of CBA. From each script used for our attacks, we can get the training and testing loss and accuracy, the attack success rate with the global trigger, and the attack success rate with each local trigger for each client separately and for the global model. For each experimental setting, we repeated the experiment 10 times to eliminate randomness introduced from our algorithms. We used `numpy.average` and `numpy.std` to cal-

[11] https://github.com/xujing1994/bkd_fedgnn

culate the average value and standard deviation. Finally, we used `matplotlib.plot` and `matplotlib.fill_between` to plot the figures, such as `ax.plot(x, y)` and `ax.fill_between(x, y-std, y+std)`.

Based on the comments from the artifact evaluation, our GitHub repository documentation is well-detailed and easy to follow. All scripts are runnable and cover all components relevant to the experimental Section 4.3.3. In addition, the obtained results are reproducible.

Takeaways During the experimental evaluation, commonly, running machine learning (ML) experiments is resource intensive and time-consuming. Thus, it is better to record too much information than not enough. For example, we had to rerun all experiments of DBA because we did not record the generated local triggers, which we realized later should be used in CBA to make sure the trigger pattern in CBA is the union set of local trigger patterns in DBA. In addition, diverse metrics are useful for debugging and can give insights into how the model training is progressing. For instance, we calculated the attack success rate with the global trigger, each local trigger for the global model, as well as the attack success rate with each local trigger for each client. Therefore, we recorded plenty of information during our analysis while only part of it was used in the Section 4.3.

It is important to motivate the required experiments and design them carefully prior to the coding part. In this way, we can avoid running unnecessary experiments that can be very time-consuming. Additionally, careful planning could eliminate mistakes in our code, which is crucial as our cluster is shared and code that crushes may affect the cluster's job scheduler. We learned the hard way that we need to document everything that we already run to avoid wasting time in the same experiments. Moreover, there are many implementation choices in our experiments. Thus, it is very useful to comment on the code and use descriptive commit messages because it is easy to forget the reason behind some of the decisions, especially if the project takes longer time to complete. Additionally, we realized that it would be useful to add unit testing in our pipeline to decrease the debugging time and avoid trivial mistakes. Finally, we believe that security conferences should follow the example of software engineering and programming language conferences and include an artifact evaluation committee to encourage authors to share artifacts.

Failure and Unexpected Results

Failures with Experiments In our experiments, we found the backdoor attack performance for a specific case, i.e., the TRIANGLES dataset and the GCN model, was obviously lower than other cases. After checking the classification accuracy of GCN over TRIANGLES, we realized the performance was poor, which can be the cause of the bad attack performance. Considering that the classification task of the TRIANGLES dataset is counting the number of triangles in a graph, and also the concept of message passing in the GNN model, a solution we devised was to change the `Aggregation` function in the GCN model from the default `mean` to `sum`. Then, the classification performance of GCN over TRIANGLES was improved significantly, as well as the attack performance. From this failure, we learned that we need to take into account the property of each dataset when configuring the experiments.

Unexpected Results In our experiments, we found for CBA, the attack success rate of all local triggers for the global model can be as high as the global trigger, which is counterintuitive as the centralized attack only embeds the global trigger into the model in the training phase. This phenomenon was not consistent with observations in [144]. For these unexpected results, the first step we took was to check the code to make sure that there is no error. Next, we proposed a hypothesis, i.e., the attack success rate of all local triggers is already high for the malicious client in the CBA. To verify this hypothesis, we designed additional experiments and finally explained this unexpected phenomenon in the Section 4.3. Thus, the results of the experiment are not always as expected and additional experiments are required to explain the current experimental results. Sometimes, obtaining unexpected results is not a problem as it can lead to new findings. What is more, we also learned that in machine learning, it is very useful to save the models after training to avoid spending time again when we evaluate them against defenses or run analysis experiments with them.

4.4. CONCLUSIONS

This chapter investigates backdoor attacks on Federated GNNs. In Section 4.2, we propose a method of implementing MIA against GNNs under the label-only condition. The average attack accuracy, precision, and AUC values of our label-only MIA are around 0.6 in most experiments, which are competitive or even better than previous probability-based MIAs implemented in the same environment and settings. In addition, we explore the influence factors of our label-only MIA. The higher overfitting level impacts the attack performance, i.e., it could increase or decrease our label-only MIA's attack accuracy. The sampling method also influences the attack performance and achieves a maximum gap of 7% on average attack accuracy. Model selection strategies with testing accuracy and loss have a slightly better attack accuracy than training accuracy and loss with a maximum gap of 1% on average attack accuracy. Then, we consider scenarios where two assumptions about the shadow dataset and the target model's information are relaxed. Surprisingly, the relaxation of those two assumptions will increase the attack performance in most experiments. Finally, we explore label-only MIA against four defense methods and their combinations. The results show that those four defenses cannot prevent our label-only MIA completely. This section only focuses on node classification tasks. We leave label-only MIA against graph-level GNNs as feature work.

In Section 4.3, we explore how Centralized and Distributed Backdoor attacks behave in Federated GNNs. Through extensive experiments on two molecular structure datasets, one synthetic dataset and two real-world social network datasets, and three popular GNN models, we showed that generally, DBA achieves a higher attack success rate than CBA. We showed that in CBA, the ASR of local triggers could be as high as the global trigger even if, during training, only the global trigger is embedded in the model. The impact of the percentage of malicious clients on DBA's ASR is analyzed with correlation, where we confirm the intuition that more malicious clients lead to more successful attacks. We analyzed the critical backdoor hyperparameters to explore their impact on the attack performance and the main task. We also demonstrated that DBA and CBA are robust against two defenses for the backdoor attack in FL. Interestingly, the ASR of DBA and CBA can be even higher under the defenses. We consider our work to provide new challenges when exploring adversarial attacks in Federated GNNs, a domain unexplored before our work. Furthermore, a powerful defense targeting backdoor attacks in Federated GNNs is required. The experimental setting in this work verifies the effectiveness of our method in a cross-silo federated learning setting and motivates further research in exploring backdoor attacks in Federated GNNs considering cross-device FL [114]. Future work will include exploring backdoor attacks in Federated GNNs for the node classification task. For example, in a social media app where each user has a local social network G^k and $\{G^k\}$ constitutes the latent entire human social network G, the developers can train a fraud detection GNN model through FL. In such a case, an attacker can conduct a backdoor attack to force the trained global model to classify a fraud node as benign.

Furthermore, in Section 4.3, we also focus on the experimental aspects of two backdoor attacks on Federated GNNs and also present the lessons we learned. In particular, artifact evaluation and sharing are necessary for reproducible research, and they also greatly help the progress of research, making it possible for other researchers to reuse benchmarks, compare their work, build upon the artifacts, etc. Thus, we believe that security conferences should include an artifact evaluation track to encourage researchers to share their artifacts. Regarding the failures in the experiments, we argue that there can be many possible causes for them, and it may be useful to analyze the datasets' properties and the model's learning task. We also argue that experimental results are not always as expected, and additional experiments are often required to support the current results. Finally, it is not trivial to run or reproduce results. We suggest that all the experiments be documented carefully so there is a clear motivation for the deployed experiments.

5

Protecting Ownership of GNNs

Graph Neural Networks (GNNs) have achieved promising performance in various real-world applications. Building a powerful GNN model is not a trivial task, as it requires a large amount of training data, powerful computing resources, and human expertise. Moreover, with the development of adversarial attacks, e.g., model stealing attacks, GNNs raise challenges to model authentication. To avoid copyright infringement on GNNs, verifying the ownership of the GNN models is necessary.

This chapter presents a watermarking framework for GNNs for both graph and node classification tasks. We 1) design two strategies to generate watermarked data for the graph classification task and one for the node classification task, 2) embed the watermark into the host model through training to obtain the watermarked GNN model, and 3) verify the ownership of the suspicious model in a black-box setting. The experiments show that our framework can verify the ownership of GNN models with a very high probability (up to 99%*) for both tasks. We also explore our watermarking mechanism against an adaptive attacker with access to partial knowledge of the watermarked data. Finally, we experimentally show that our watermarking approach is robust against a state-of-the-art model extraction technique and four state-of-the-art defenses against backdoor attacks.*

This chapter has been published as "Watermarking Graph Neural Networks based on Backdoor Attacks." ***Xu, J.****, Koffas, S., Ersoy, O., & Picek, S. (2023). The IEEE European Symposium on Security and Privacy*

5.1. Introduction

Many real-world data can be modeled as graphs, e.g., social networks, gene interactions, and transport networks. Similar to the great success of deep learning algorithms on, e.g., image recognition [57, 69, 117], speech recognition [49, 59], and natural language processing [46], deep graph models such as graph neural networks (GNNs) [67, 127, 53] have also achieved promising performance in processing graph data. Such successful results can be attributed to their superior ability to incorporate information from neighboring nodes in the graph recursively [142]. Still, building and training a well-performed graph neural network is not a trivial task, as it usually requires a large amount of training data, effort in designing and fine-tuning a model, and powerful computing resources, making the trained model have a monetary value. For instance, the cost of training a machine learning model can be more than one million USD [120].

As graph neural networks are more widely developed and used, their security also becomes a serious concern. For instance, the adversary can steal the model through a model stealing attack. Recent works have shown the high effectiveness of model stealing attacks on complex models even without knowledge of the victim's architecture or the training data distribution [122, 98, 101], which leads to model copyright infringement. Moreover, if the model is intended to be released for commercial purposes, the stolen model would even lead to financial loss. Therefore, it is crucial to verify the ownership of a GNN model.

Digital watermarking is typically used to identify ownership of the copyright of media signals, e.g., audio, video, and image data [73]. The information to be embedded in a signal is called a digital watermark. The signal where the watermark is embedded is called the host signal. A digital watermarking system is usually divided into two steps: embedding and verification. Figure 5.1 shows a typical digital watermarking life cycle. In the embedding step, an embedding algorithm E embeds the watermark into the host signal to generate the watermarked data S_w. After embedding, the watermarked data is transferred or modified (dashed frame, this part is optional). During the watermark verification step, a verification algorithm is applied to attempt to extract the watermark from the watermarked signal. If the extracted watermark is equal to or within the acceptable distance of the original watermark, we can confirm that the signal is the protected signal. More recently, with the development of deep neural networks, new watermarking methods were designed to protect the DNN models by embedding watermarks into DNN models [125, 22, 2]. The idea of watermarking neural networks is similar to traditional digital watermarking in multimedia data. To implement digital watermarking in neural networks, we can assume the multimedia data that we want to protect is the model, and the embedding and verification steps of the watermarking correspond to the training and inference phase of the protected model. A neural network watermarking model should satisfy the following requirements [75, 13]: robustness, fidelity, capacity, integrity, generality, efficiency, and secrecy (see Section 5.3.2).

Multiple works discuss embedding watermarks into DNN models to protect the IP of the models [125, 2, 162, 151, 63, 28, 87]. For instance, Uchida et al. [125] presented a framework to embed watermarks into the parameters of DNNs via the parameter regularizer during training leading to its white-box setting. In such a white-box scenario, the internal details of the suspicious model, including the network structure and parame-

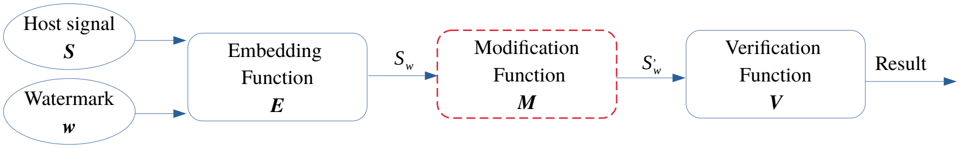


Figure 5.1: Digital watermarking life cycle.

ters, must be accessible to the model owners, which significantly limits the application of this watermarking method since the suspicious model (a model which is suspiciously stolen from the host model) are usually developed as a cloud service, so only the output predictions of the model can be acquired. To address the limitations of watermarking DNNs in the white-box setting, Adi et al. used random training instances and random labels to watermark a neural network in a black-box way [2]. The authors based their approach on backdoor attacks. Additionally, Zhang et al. [162] extended the threat model to support black-box setting verification for DNN models. To enhance the robustness of the watermarking, Yang et al. proposed to leverage the concept of fault attack to embed a watermark into a DNN model for IP protection [151], and Jia et al. presented an approach to force the model to entangle representations for legitimate task data and watermarks [63]. A robust black-box watermarking scheme for self-supervised learning pre-trained encoders was proposed by Cong et al. [28]. The general idea of watermarking a model in a black-box setting is to train the model using specific samples so that the model can memorize the watermark information and be verified when predicting on these samples. The discussed works focus on the image and audio domains, not graph data, making the selection of datasets and neural network models different and direct comparisons difficult.

The watermark generation methods and injecting position differ between image and graph data [167, 145]. Specifically, as non-Euclidean data, the graph has rich structural information that can be used to generate the watermark. In the image domain, the watermark injecting position can be defined, which is impossible in graph data as there is no position information one can exploit in a graph. Thus, the discussed watermarking mechanism cannot generate watermarks for graph data.

To the best of our knowledge, only one work considers watermarking graph data and GNNs. Zhao et al. presented a watermarking framework for GNNs by generating a random graph associated with features and labels as the watermark [167]. However, this work only studies the watermarking in GNNs for the node classification task while neglecting other relevant settings, e.g., the graph classification task. The applications of graph classification are numerous and range from determining enzymes in bioinformatics, categorizing documents in NLP, and analyzing social networks. Furthermore, the presented method only works for the GNN models trained through inductive learning. If the owner's model is trained by transductive learning, the proposed method in [167] is not feasible as the graph structure of the training graph has been changed. GNNs can take advantage of transductive learning, thanks to the natural way they make information flow and spread across the graph, using relationships among patterns [25]. Thus, various transductive approaches have been widely applied and implemented in many domains, such as natural language processing (NLP), surveillance, graph reconstruction, and ECG classification [106]. In this chapter, we present the first watermarking framework for GNNs suitable for both graph and node classification tasks as well as models trained by both inductive and transductive learning. Besides broadening the applicability of watermarking in GNN settings, our framework has significant improvements compared to state-of-the-art [167]. First, we present a statistical analysis for evaluating our watermarking framework. Specifically, we apply Welch's t-test to verify the effectiveness of our watermarking mechanism. We also calculate a threshold to ensure a low false positive rate (FPR) and false negative rate (FNR), i.e., less than 0.0001. Second, together with the two backdoor defenses (model pruning and fine-tuning) evaluated against the proposed watermarking mechanism in [167], we show that our framework is robust against a model extraction attack (knowledge distillation), two more backdoor defenses (randomized subsampling, and fine-pruning) and an adaptive attacker. Finally, in [167], the experiments were conducted on only two node classification datasets and one model. In our work, we test our framework with three datasets and three GNN models for graph classification and two datasets and three GNN models for node classification. The extensive experiments show that our method is not limited to specific GNN models. Our watermarking method can achieve higher watermark accuracy than [167], i.e., up to 100%.

Following the idea of [2], our watermarking method utilizes backdoor attacks. Backdoor attacks in GNNs aim to misclassify graph data embedded with a trigger. In our work, instead of considering backdoor attacks in GNNs [135, 143, 164] for offensive purposes, we use them to protect the IP of the GNN models. More precisely, we use the backdoor triggers as digital watermarks to identify the ownership of a GNN model. Our watermarking framework includes three phases:

- **Watermarked data generation**. We designed two strategies to generate watermarked data for the graph classification and one for the node classification.
- **Watermark embedding**. We train the host model with the watermarked data. The intuition is to explore the memorization capabilities of GNNs to learn the trigger pattern of the watermarked data automatically.
- **Ownership verification**. Once the watermark is embedded into the model, we can verify the ownership of remote suspicious models by sending watermarked data generated in the first phase. Only the models protected by the watermarks are assumed to output matched predictions. To address the limitations of [167], we use the feature trigger as the watermark pattern by modifying the feature information of the graph instead of changing the graph's structure.

We evaluate our watermarking framework with five benchmark datasets: two for the node classification task and three for the graph classification task. The results show that our watermarking framework can verify the ownership of suspicious models with high probability. At the same time, the performance of the watermarked GNN on its original task can be preserved. Our main contributions can be summarized as follows:

- We propose a watermarking framework to verify the ownership of GNN models for both the node and graph classification tasks. It is the first watermarking framework for GNNs on the graph classification task.
- We use hypothesis testing in our watermarking mechanism to provide statistical analysis for the model ownership verification results.

- We propose two watermark generation mechanisms to generate watermarked data for the graph classification task. One strategy is based on classical backdoor attacks, and the other is based on embedding the watermark into random graphs, which experimentally shows superior performance.
- For the node classification task, we propose a training-agnostic framework that also applies to a model trained by transductive learning. Specifically, we only modify the feature information of the graph in the watermarked data generation phase.
- We explore our watermarking mechanism against an adaptive attacker who has knowledge of partial watermarked data. The experiments show that it is difficult to unlearn the watermark functionality without influencing the main task for graph classification.
- We investigate the robustness of our method against a model extraction attack and four defenses against backdoor attacks. Experimental results show our watermarked model is robust against these mechanisms.

We evaluate our watermarking framework with several benchmark datasets and popular GNN models. Experimental results show that the proposed method achieves excellent performance, i.e., up to 99% accuracy, in IP protection of the models while having a negligible impact on the original task (less than 1% clean accuracy drop).

5.2. GNN WATERMARKING

In this section, we propose a framework to generate watermarked data, embed a watermark into GNNs, and verify the ownership of GNNs by extracting a watermark from them. The framework's purpose is to protect the IP of the graph neural networks by verifying the ownership of suspicious GNNs with an embedded watermark. The framework first generates watermarked data and trains the host GNNs with the watermarked data. The GNNs automatically learn and memorize the connection between the watermark pattern and the target label through training. As a result, only the model protected with our watermark can output predefined predictions, i.e., the assigned target label, when the watermark pattern is observed in the queries sent to the suspicious model. Figure 5.2 illustrates the workflow of our GNN watermarking framework.

5.2.1. THREAT MODEL

In our threat model, we model two parties, a model owner, who owns a graph neural network model m for a certain task t, and a suspect, who sets up a similar service $t^{'}$ from the model $m^{'}$, where two services have a simillar purpose $t \approx t^{'}$. In practice, there are multiple ways for a suspect to get the model m. For example, it could be an insider attack from the owner's organization that leaks the model, or it could be stolen by various model stealing attacks, e.g., [122, 9]. Although the exact mechanism of how a suspect obtains the model m is out of the scope of this chapter, we still evaluate our watermarking method against a model extraction attack in Section 5.4. Our goal is to help the model owner protect his/her model m, an intellectual property with a concrete value. Intuitively, if model m is equivalent to $m^{'}$, we can confirm that the suspect is a plagiarizer

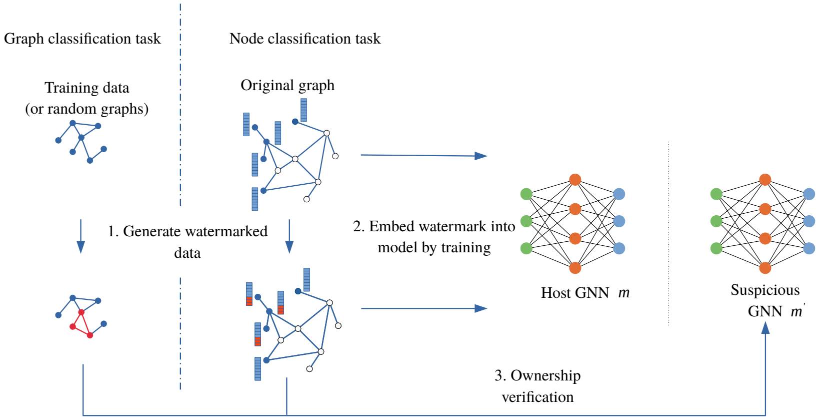

Graph classification task
Node classification task
Training data
(or random graphs)
Original graph
1. Generate watermarked data
2. Embed watermark into model by training
Host GNN m
Suspicious GNN m'
3. Ownership verification

Figure 5.2: GNN watermarking framework.

and m' is a plagiarized model of m.[1] We define the owner's model m as the host model and the model m', which is likely to be stolen from m as the suspicious model. In this work, we assume that as an IP protector, we can only query the suspicious model m' in a black-box manner. As the model owners, we have full access to the model m, including its architecture, training data, and the training process.

5.2.2. Watermarked Data Generation

Graph classification. Since most graph classification tasks are implemented using GNNs to learn the network structure, we focus on utilizing the subgraph-based backdoor attacks [164, 143] to verify the ownership of GNNs for the graph classification task. Here, we investigate two watermarking data generation strategies for GNNs on the graph classification task.

Embedding watermark into original training data. Specifically, we select a subset of samples in training data and embed a generated random watermark (i.e., a random graph) into it. A random graph including n nodes and e edges is generally generated by starting with n isolated nodes and inserting e edges between nodes at random. A classical method to generate random graphs is called Erdos-Renyi (ER) random graph model [44] in which each possible edge occurs independently with a probability $p \in (0,1)$. This chapter utilizes ER random graphs as the watermark graph for watermarking GNNs on the graph classification task since ER model is commonly used in the graph domain and has shown to be more effective than the other models like Small World (SW) model [134] and Preferential Attachment (PA) model [7]. Once the watermark graph is generated, we embed it into each graph of the selected subset of training data by randomly choosing n nodes in the graph and changing their connections to be the same as the watermark graph. Since we only change the graph's structure, we do not modify the node's features. The watermark information is also carried by the label of the watermark

[1] The chances that various entities created the same model independently are small, especially when considering real-world applications.

embedded graphs. We assume the value of the label of the watermarked graphs is in the range $[0, C)$, where C is the number of classes, and the label for watermarked graphs can be determined in advance. We emphasize that the labels of the sampled training data are different from the label of the watermark-embedded graphs. In this way, three parameters - r (proportion of training data selected to be injected with the watermark graph), n (number of nodes in the watermark graph), and p (the probability of the edge existence in the watermark graph) would have a significant impact on the watermark generation, then affecting the watermark embedding and verification later.

Embedding watermark into generated random graphs. In addition to embedding a watermark graph into the original training data, we propose first generating random graphs and then embedding a generated watermark into these random graphs. The intuition here is that by embedding a watermark graph into the original training data, as discussed in the previous paragraph, the watermark will have some side effects on the original functionality of our watermarked graph neural networks. We design this strategy to decrease the impact of watermarking in the original task. First, we generate a number of random graphs with the ER method, where we define that number as a specific proportion (r) of the training data. The number of nodes and edge existence probability are the same as the average number of nodes and edges of the training data. Then, we generate the watermark graph in the same way described in the previous approach and embed the watermark graph into the random graphs generated in the first step. We use the node degree as the node feature for the generated random graphs. We also assign the label for the watermark-embedded graphs in advance, similar to the first strategy. There are also three parameters r, n, and p in this strategy.

The detailed comparison and analysis of these parameters in the above two strategies are given in Section 5.3. Concerning the adversary capability, the first watermarked data generation strategy requires access to a partial training dataset, and the second strategy requires none as it uses extra random graphs to generate the watermarked data.

Node classification. We apply the backdoor attack as proposed in [145] to implement watermarking GNNs for the node classification task, which can be applied in not only inductive learning but also transductive learning-based models. Specifically, we randomly select a proportion r of the total number of nodes in the graph as the watermark carrier nodes and change their subset node features[2] into a predefined fixed value[3] to generate the watermarked data. Given an arbitrary node in the graph, by changing the value of a subset of its features as a feature trigger and assigning a target label to it, the host model m aims to learn and memorize the watermark pattern.

5.2.3. WATERMARK EMBEDDING

Graph classification. Once the watermarked data are generated, the next step in the framework is to embed the watermark into the host GNN model m. Here, we explore the intrinsic learning capability of graph neural networks to embed the watermark. We first train a clean model m_c based on the original training data D_{train} and then continue training the model using the watermarked data. The detailed GNN watermark embedding process is shown in Algorithm 8. The inputs are the pre-trained clean model m_c,

[2] Here, the number of node features whose values are changed is defined as watermark length l.

[3] The fixed value is uniformly selected between 0 and 1.

original training data D_{train} and target label for the watermarked data, and the outputs are the watermarked GNN model m_w and watermarked data D_{wm}. The model owner defines the target label of the watermarked data. In the main function, we sample data D_{tmp} from the original training data uniformly at random. The data we sample has a label that is different from the target label (Line 3 in Algorithm 8) so that we can avoid the influence of the original label in the ownership verification phase. For the second watermarking strategy for the graph classification task, which is based on generating random graphs as watermark carrier data, we utilize the *ER* method to generate random graphs with an average number of nodes and edges of the training data and proportion r (Line 5 in Algorithm 8). Then, for each data in D_{tmp}, we add the generated watermark to x and relabel it with y_t (Lines 7-11 in Algorithm 8). Therefore, we obtain the watermarked data D_{wm}, which is later used in the verification process. We train the pre-trained clean GNN model with both the sampled original data D_{tmp} and D_{wm} (or just D_{wm} for the second strategy).[4] We assume that the GNN model will learn the watermark pattern during the training process and, thus, be protected against the model stealing attacks.

Algorithm 8: Watermark embedding for graph classification task

```
   Input: Pre-trained clean model $m_c$, Training set $D_{train} = \{x_i, y_i\}_{i=1}^S$, Target label $y_t \in [0, C)$
   Output: Watermarked GNN model $m_w$, Watermark data $D_{wm}$
1  Function WATERMARK_EMBEDDING():
2      $D_{wm} \leftarrow \emptyset$
3      $D_{tmp} \leftarrow sample(D_{train}, r, y \neq y_t)$ // strategy 1
4      // or
5      $D_{tmp} \leftarrow GRAPH\_GENERATE(n_{avg}, p_{avg}, r))$ // strategy 2
6      foreach $d \in D_{tmp}$ do
7          $x_{wm} = ADD\_WATERMARK(d[x], watermark)$
8          $y_{wm} = y_t$
9          $D_{wm} = D_{wm} \cup \{x_{wm}, y_{wm}\}$
10     end
11 End Function
12 $m_w = Train(m_c, D_{wm}, D_{tmp})$
13 (or $m_w = Train(m_c, D_{wm})$)
14 return $m_w, D_{wm}$
```

The watermarking embedding process for the node classification is the same as the graph classification task.

5.2.4. OWNERSHIP VERIFICATION

After training our model with watermarked data, if adversaries steal and further fine-tune the watermarked model, they will likely set up an online service to provide the AI service of the stolen model. Then, it is difficult to access the architecture and parameters of the suspicious model directly. As we have explained in Section 5.2.1, to verify the ownership of the suspicious model $m^{'}$ in a black-box manner, we can send D_{wm}, which is returned in the previous watermark embedding process to the suspicious model. If, for part of samples in D_{wm}, the suspicious model outputs the target label y_{wm}, we can assume that $m^{'}$ is stolen (developed) from our watermarked model m_w. However, the premise of this assumption is that our watermarked model has statistically different be-

[4] We use both the sampled original data and watermarked data for the first strategy to decrease the impact of watermarking on the model's original main task.

havior from the clean model, leading to different watermark accuracy between these two models. To provide a statistical guarantee with the model ownership verification results, we can adopt statistical testing with the ability to estimate the level of confidence to determine whether the watermark accuracy of our watermarked model is significantly different from the clean model. We define the null hypothesis $\mathscr{H}_0$ as follows:

$$\mathscr{H}_0 : P_r(m_w(x_{wm}) = y_{wm}) \cong P_r(m_c(x_{wm}) = y_{wm}),$$

where $P_r(m_w(x_{wm}) = y_{wm})$ represents the watermark success probability of the watermarked model and $P_r(m_c(x_{wm}) = y_{wm})$ represents the watermark success probability of a clean model.

The null hypothesis $\mathscr{H}_0$ states that the watermark success probability of the watermarked model is equal or approximate to the clean model, i.e., there are no significant differences between the watermark accuracy of the watermarked model and a clean model. On the contrary, the alternative hypothesis $\mathscr{H}_1$ states that the watermarked model has significantly different watermark accuracy from the clean model, which verifies the effectiveness of our watermarking mechanism. If we can reject the null hypothesis $\mathscr{H}_0$ with statistical guarantees, we can claim that our watermarking method successfully verifies the ownership of the suspicious models.

Through querying a series of watermarked and clean models with q watermarked samples, we can obtain their prediction results for each watermarked model and clean model, which can be used to calculate the watermark accuracy, denoted as α_k and β_k, respectively:

$$\alpha_k = \frac{\sum_{i=1}^{q} \mathbb{I}(y_i^{w_k} = y_{wm})}{q}, k \in [1, n] \tag{5.1}$$

$$\beta_k = \frac{\sum_{i=1}^{q} \mathbb{I}(y_i^{c_k} = y_{wm})}{q}, k \in [1, n] \tag{5.2}$$

where n is the number of watermarked and clean models. We set n to 10 in our experiments.

The value of watermark accuracy can be considered as an estimation of the watermark success probability. We apply the Welch's t-test [136][5] to test the hypothesis. According to the watermark accuracy of n watermarked models and clean models, denoted as $\{\alpha_1, \cdots \alpha_n\}$ and $\{\beta_1, \cdots \beta_n\}$ respectively, we can calculate the t statistic:

$$t = \frac{\bar{\alpha} - \bar{\beta}}{\sqrt{\frac{s_\alpha^2}{n} + \frac{s_\beta^2}{n}}}, \tag{5.3}$$

where s_α^2 and s_β^2 are the unbiased estimators of the population variance.

The degrees of freedom ν associated with variance estimate is approximated using the Welch-Satterthwaite equation [108, 136]:

$$\nu \approx \frac{(N-1)(s_\alpha^2 + s_\beta^2)^2}{s_\alpha^4 + s_\beta^4}. \tag{5.4}$$

[5] We use the Welch's t-test since the watermark accuracy of clean and watermarked models can be treated as normal distributions according to a Shapiro-Wilk Test [112], and they may have different variances.

According to the theoretical analysis above, we can formally state under what conditions the model owner can reject the null hypothesis $\mathscr{H}_0$ at the significance level $1-\tau$ (i.e., with τ confidence) with watermark accuracy of watermarked and clean models. Specifically, we take the watermark accuracy results of NCI1 and DiffPool as an example, as shown in Table 5.4. According to a Shapiro-Wilk Test [112], the p-values [133] of these two populations, i.e., watermark accuracy of clean and watermarked models, are 0.77 and 0.16, respectively. Given a significance level of 0.05, these p-values indicate these two populations can be assumed to be normally distributed, and a Welch's t-test is applicable. With significance level $1-\tau = 0.05$, and the degree of freedom calculated with Eq. (5.4), i.e., $\nu \approx 16$, the t critical value t_τ is 2.120. Based on Eq. (5.3), we calculate t statistic $t = 45.82$, which is significantly larger than $t_\tau = 2.120$. Thus, we can reject the null hypothesis $\mathscr{H}_0$ at the significance level 0.05 for the NCI1 dataset on the DiffPool model in our work. The watermark accuracy of the watermarked models and clean models of other datasets and models are presented in Table 5.1, 5.2, and 5.3. The corresponding t statistics and t critical values are also presented in these tables. As we can observe for each setting, we can reject the null hypothesis $\mathscr{H}_0$ at the significance level 0.05, which provides a statistical guarantee for our watermarking method. Based on the statistical analysis above, we can also calculate a threshold for each dataset and model to ensure a low false positive rate (FPR) and false negative rate (FNR), i.e., less than 0.0001, of our watermarking method. More details are presented in Section 5.3.

If the watermark accuracy of clean and watermarked models are not normally distributed, we can apply the Mann-Whitney U test [90] to test the hypothesis. The U statistic is calculated as:

$$U_1 = R_1 - \frac{n_1(n_1+1)}{2};\ U_2 = R_2 - \frac{n_2(n_2+1)}{2}, \tag{5.5}$$

where R_1, R_2 are the sums of ranks in watermark accuracy of clean and watermarked models respectively, and n_1, n_2 are the numbers of two populations. Then, the smaller value of U_1 and U_2 is used to check the significance of the difference between the distributions. Here, we still take the watermark accuracy results of NCI1 and DiffPool as an example (Table 5.4 where $n_1 = n_2 = 10$). Based on Eq. (5.5), we calculate U statistic and obtain $U_1 = 0$ and $U_2 = 100$. Thus, the value $U_1 = 0$ is used for the significance check. According to the reference table[6], the critical value is 23 with significance level $1-\tau = 0.05$. For our example, $U_1 = 0$ is lower than the critical value. Moreover, $U = 0$ is the lowest possible value and implies the complete separation of the groups (watermark accuracy of clean and watermarked models). For other datasets and models, the calculated U statistics are all less than the critical value. Thus, given the populations of the watermark accuracy of clean and watermarked models are not normally distributed, we can still reject the null hypothesis $\mathscr{H}_0$ for a 0.05 significance level, which indicates that our watermarking method is also valid for the watermark accuracy of non-Gaussian distribution.

[6] https://math.usask.ca/~laverty/S245/Tables/wmw.pdf

Table 5.1: Accuracy of the watermarked models and clean models on D_w^t for graph classification task ($n = 10$).

Setting	Models	Watermark Accuracy (%)										t	ν	t_τ
NCI1_GIN	m_c	0.21	3.17	8.00	13.78	14.47	8.27	16.00	9.02	8.44	12.69	26.04	14	2.145
	m_w	94.42	89.40	85.22	79.26	83.98	99.40	99.73	99.07	85.91	77.75			
NCI1_.SAGE	m_c	0.10	0.40	1.64	0.49	3.65	0.28	0.87	9.58	0.39	0.79	59.10	16	2.120
	m_w	94.21	93.16	93.43	87.50	89.99	99.98	95.40	87.47	94.54	90.02			
COLLAB_Diff.	m_c	13.63	10.84	7.39	7.62	10.46	12.47	20.72	11.49	10.44	7.06	39.17	17	2.110
	m_w	76.19	83.51	85.16	92.91	82.83	87.42	84.53	80.84	84.51	82.66			
COLLAB_GIN	m_c	14.17	9.88	9.01	16.89	21.23	17.65	19.71	14.37	7.67	16.82	25.41	15	2.131
	m_w	84.80	91.09	77.28	73.57	88.93	91.01	91.73	78.57	76.35	92.06			
COLLAB_.SAGE	m_c	14.67	17.66	16.65	17.59	11.96	22.51	7.85	17.56	8.23	13.41	37.06	17	2.110
	m_w	85.60	80.34	83.54	79.68	82.83	77.78	81.52	83.11	88.24	89.05			
REDDIT._Diff.	m_c	11.99	11.08	8.65	7.33	9.61	0.87	5.70	3.96	9.57	12.35	49.70	18	2.101
	m_w	83.48	94.49	94.76	88.66	89.85	93.03	88.23	92.65	86.49	92.75			
REDDIT._GIN	m_c	13.64	15.99	10.24	13.19	11.27	11.55	4.79	14.16	11.98	2.04	32.86	16	2.120
	m_w	92.55	86.07	97.01	91.46	77.52	83.99	80.15	86.00	91.73	89.23			
REDDIT._.SAGE	m_c	0.05	0.04	0.21	2.52	1.64	1.18	2.74	0.79	0.40	0.52	164.02	15	2.131
	m_w	96.21	98.81	99.09	99.61	99.53	97.46	97.57	94.68	98.14	99.09			

Table 5.2: Accuracy of the watermarked models and clean models on D_w^r for graph classification task ($n = 10$).

Setting	Models	Watermark Accuracy (%)										t	ν	t_τ
NCI1_Diff.	m_c	51.03	47.51	43.06	51.30	43.81	48.01	39.36	52.08	41.38	42.92	26.37	17	2.110
	m_w	95.62	90.33	93.50	94.34	96.71	99.28	97.42	99.06	95.44	87.54			
NCI1_GIN	m_c	35.00	47.02	55.15	52.48	52.65	34.15	50.74	40.17	51.01	45.20	17.71	13	2.160
	m_w	89.76	96.34	89.76	95.44	99.48	88.53	95.14	91.87	99.27	97.11			
NCI1_.SAGE	m_c	48.61	50.85	41.02	55.53	47.87	51.62	46.47	44.07	44.18	50.62	30.15	15	2.131
	m_w	95.47	93.67	97.20	99.33	99.60	99.95	97.82	98.72	99.49	91.52			
COLLAB_Diff.	m_c	26.90	30.49	30.46	26.37	28.61	34.08	25.72	24.33	27.63	30.37	54.22	17	2.110
	m_w	99.02	98.69	93.04	99.16	94.89	93.74	99.45	99.92	93.71	96.67			
COLLAB_GIN	m_c	34.52	30.76	20.49	26.68	33.93	33.44	30.72	31.67	30.99	36.68	30.81	17	2.110
	m_w	91.82	88.00	99.25	88.16	89.55	99.52	92.86	93.51	99.26	92.53			
COLLAB_.SAGE	m_c	23.07	29.23	24.54	24.79	28.36	32.10	34.99	28.20	31.53	25.01	49.00	14	2.145
	m_w	99.59	95.28	93.24	99.25	98.10	96.07	99.98	99.37	99.67	97.15			
REDDIT._Diff.	m_c	42.87	41.59	46.01	42.85	46.20	40.40	39.23	41.22	40.50	47.81	44.89	17	2.110
	m_w	95.58	95.79	99.27	94.36	97.29	92.61	99.16	99.74	99.95	96.39			
REDDIT._GIN	m_c	49.90	43.29	46.39	47.79	45.60	44.44	48.16	40.27	46.86	45.63	31.24	15	2.131
	m_w	86.38	92.03	97.16	99.92	98.44	99.27	99.39	95.35	92.99	97.14			
REDDIT._.SAGE	m_c	50.89	43.93	47.71	47.44	47.93	45.83	53.12	53.68	44.45	44.65	39.52	13	2.160
	m_w	93.85	99.33	97.34	96.49	97.49	96.15	97.42	99.39	99.17	99.47			

Table 5.3: Accuracy of the watermarked models and clean models on watermarked data for node classification task ($n = 10$).

Setting	Models	Watermark Accuracy (%)										t	v	t_τ
Cora_GCN	m_c	0.33	0.17	6.25	3.23	6.17	3.83	4.03	0.20	0.93	2.96	88.78	17	2.110
	m_w	99.66	93.14	99.42	94.37	96.97	99.91	98.11	95.73	98.47	99.86			
Cora_GAT	m_c	10.39	3.73	10.14	0.87	5.81	10.66	4.66	2.63	9.74	7.61	53.51	17	2.110
	m_w	92.30	99.90	99.73	87.90	97.98	95.01	99.24	95.55	99.57	95.16			
Cora_.SAGE	m_c	1.15	4.94	1.61	4.57	3.12	4.81	0.67	0.22	0.52	1.25	89.19	15	2.131
	m_w	97.52	93.24	92.44	99.97	95.22	99.94	99.17	96.25	97.99	99.80			
Cite._GCN	m_c	0.92	2.06	3.22	10.08	0.72	1.26	4.11	6.76	7.70	0.23	78.00	13	2.160
	m_w	99.96	99.86	95.94	95.17	99.07	97.59	96.30	99.71	97.81	99.09			
Cite._GAT	m_c	2.50	0.55	0.86	3.32	0.61	2.03	1.41	2.56	0.37	0.37	129.83	13	2.160
	m_w	99.30	99.13	99.01	97.78	98.20	94.13	95.87	94.69	99.99	99.20			
Cite._.SAGE	m_c	0.99	1.05	0.73	1.14	0.81	0.35	0.34	0.28	1.33	0.22	381.65	14	2.145
	m_w	99.75	97.83	99.95	99.41	99.74	99.85	98.22	99.13	99.36	99.01			

Table 5.4: Watermark accuracy of the watermarked and clean models on NCI1 with DiffPool ($n = 10$).

Models	Watermark Accuracy (%)									
Clean	0.70	6.49	4.27	9.77	14.85	7.43	1.59	5.71	14.09	10.94
Watermarked	94.98	94.88	99.32	92.67	92.11	99.73	99.58	89.52	99.78	97.50

Table 5.5: Datasets statistics.

Datasets	# Graphs	Avg. # nodes	Avg. # edges	Classes
NCI1	4,110	29.87	32.30	2
COLLAB	5,000	74.49	2,457.78	3
REDDIT-BINARY	2,000	429.63	497.75	2
Cora	1	2,708	5,429	7
CiteSeer	1	3,327	4,608	6

5.3. EVALUATION

In this section, we evaluate the performance of our watermarking mechanism in GNNs for graph classification and node classification tasks. We run the experiments on a remote server with one NVIDIA 1080 Ti GPU with 32GB RAM. We use PyTorch, and each experiment is repeated ten times.

Dataset. For the graph classification task, we use three publicly available real-world graph datasets (one chemical dataset and two discussion datasets): (i) NCI1 [92] - a subset of the dataset consisting of chemical compounds screened for activity against non-small cell lung cancer, (ii) COLLAB [150] - a scientific collaboration dataset, derived from three public collaboration datasets, and (iii) REDDIT-BINARY [150] - a dataset consisting of graphs corresponding to online discussions on Reddit. For the node classification task, we use two real-world datasets: (i) Cora [110] and (ii) CiteSeer [110]. These two datasets are citation networks in which each publication is described by a binary-valued word vector indicating the absence/presence of the corresponding word in the collection of 1,433 and 3,703 unique words, respectively. Table 5.5 shows the statistics of all considered datasets.

Dataset splits and parameter setting. For each graph classification dataset, we sample 2/3 as the training data and the rest as the test data. We set the watermark graph size as γ fraction of the graph dataset's average number of nodes. We then sample or generate an r fraction of the training data (with an un-target label) to embed the generated watermark. For each node classification dataset, we use 20% of total nodes as the training data. We set the size of the feature watermark to l and then sample r fraction of the training data to embed the generated feature watermark. The comparison of watermarking performance under different variants is shown in Section 5.3.1.

Models. We use three state-of-the-art GNN models for the graph classification task: DiffPool [156], GIN [146], and GraphSAGE [53]. For the node classification task, we use

GCN [67], GAT [127], and GraphSAGE [53] as the host models. The hyperparameters for the neural networks (Table 5.6) are commonly used, see [143, 156, 146].

Table 5.6: Default hyperparameter setting. G: graph classification, N: node classification.

Type	Hyperparameter	Setting
DiffPool[†]	Architecture	2 DIFFPOOL layers
GIN	Architecture	2 aggregation layers
GraphSAGE	Architecture	2 aggregation layers
	Aggregator	Mean [53]
GCN	Architecture	5 aggregation layers
GAT	#Heads	3
Training	Learning rate	0.01
	Optimizer	Adam
	Weight decay	5e-4
	Dropout	0.5
	Epochs	350 (G), 100 (N)
	Batch size	32

[†] Here, the GNN model used for DiffPool is built on top of the GraphSAGE architecture [156].

Metrics. The main purpose of our watermarking framework is to verify the ownership of the suspicious GNN model successfully. According to the statistical analysis in Section 5.2.4, we can guarantee that our watermarking method can successfully verify the ownership of the suspicious models. Specifically, based on the watermark accuracy distribution of our watermarked models and clean models, we can calculate a threshold of watermark accuracy for each dataset and model to ensure a low FPR and FNR, i.e., less than 0.0001, as shown in Tables 5.7 and 5.8 for graph classification task and node classification task, respectively. In Table 5.7, D^t_{wm} is the watermarked data generated by embedding a watermark into sampled training data, while D^r_{wm} is the watermarked data generated by embedding a watermark into the generated random graphs. From Table 5.7, the watermark accuracy threshold of the second strategy is obviously higher than the first strategy. It can be explained that, in the first strategy, the clean model will likely classify the watermarked data into the original label since it is generated based on the training data, and the clean model does not learn the watermark pattern. In the second strategy, the watermarked data is generated based on random graphs, and the clean model is likely to classify it uniformly at random. Thus, the watermark accuracy of the clean models in the first strategy is nearly 0%, and that in the second strategy is around $1/C$ (C is the number of classes). As a result, the watermark accuracy of the watermarked models in the second strategy should be higher than the first strategy to ensure the distinguishable difference between the watermarked models and the clean models. The dataset with more classes (COLLAB) has a lower threshold than the other datasets (i.e., two classes), as shown in Table 5.7. Once the watermark accuracy threshold is defined, we send queries of generated watermarked data to the suspicious model

Table 5.7: Watermark accuracy threshold for each dataset and model on the graph classification task.

Dataset	Watermark Acc. Threshold (%) ($D^t_{wm} \mid D^r_{wm}$)		
	DiffPool	GIN	GraphSAGE
NCI1	53.5|72.0	44.0|76.5	45.50|75.5
COLLAB	47.0|63.0	44.5|62.5	50.5|65.5
REDDIT-BINARY	49.5|71.0	46.5|68.0	48.5|76.0

Table 5.8: Watermark accuracy threshold for each dataset and model on the node classification task.

Dataset	Watermark Accuracy Threshold (%)		
	GCN	GAT	GraphSAGE
Cora	50.0	51.0	48.0
CiteSeer	53.0	48.0	49.5

m'. If the watermark accuracy of the suspicious model is over the corresponding threshold, we can reach the conclusion that the suspicious model is stolen or developed from the host model.

Besides a good performance on the ownership verification task, a well-designed watermarking method should have only slight side effects on the host model's original task. Thus, we check whether our watermarking framework reduces the performance of the watermarked GNN model on its original task. We compare watermarked and clean models' accuracy on the normal test data.

5.3.1. EXPERIMENTAL RESULTS

Graph classification. As discussed in Section 5.2.2, there are two strategies for generating watermarked data D_{wm}. For each strategy, three parameters (watermarking rate r, watermark graph size n, and watermark graph density p) will affect the generated watermarked data and the final watermarking performance. The watermark accuracy of different datasets and models with different variants (r, n, p) is shown in Figure 5.3. In Figure 5.3, the watermark accuracy of the second watermarked data generation strategy is generally higher than the first strategy, which means the watermark pattern in the random graphs is more likely to be successfully learned by the model. In the first strategy, the original feature pattern in the graph may influence the learning of the watermark pattern, whereas, in the second strategy, embedding the watermark in random graphs can reduce this influence as it is the only important feature in the graph.

From Figure 5.3a, for both datasets, with the increase of the watermarking rate, the watermark accuracy of all three models based on the first strategy is generally increasing, as well as for the second strategy. Indeed, with a higher watermarking rate, more training data will be embedded with the watermark so that the host model can learn the watermark pattern better. However, even with the lowest watermarking rate, the watermark accuracy on all models and datasets is higher than the threshold in Table 5.7, indicating

Table 5.9: Watermark accuracy for graph classification task ($r = 0.15, \gamma = 0.2, p = 1.0$).

Dataset	Watermark Accuracy (%) ($D^t_{wm} \mid D^r_{wm}$)		
	DiffPool	GIN	GraphSAGE
NCI1	96.01|94.92	89.41|94.27	92.57|97.28
COLLAB	84.05|96.83	84.54|93.45	83.17|97.77
REDDIT-BINARY	90.43|97.01	87.57|95.81	98.02|97.61

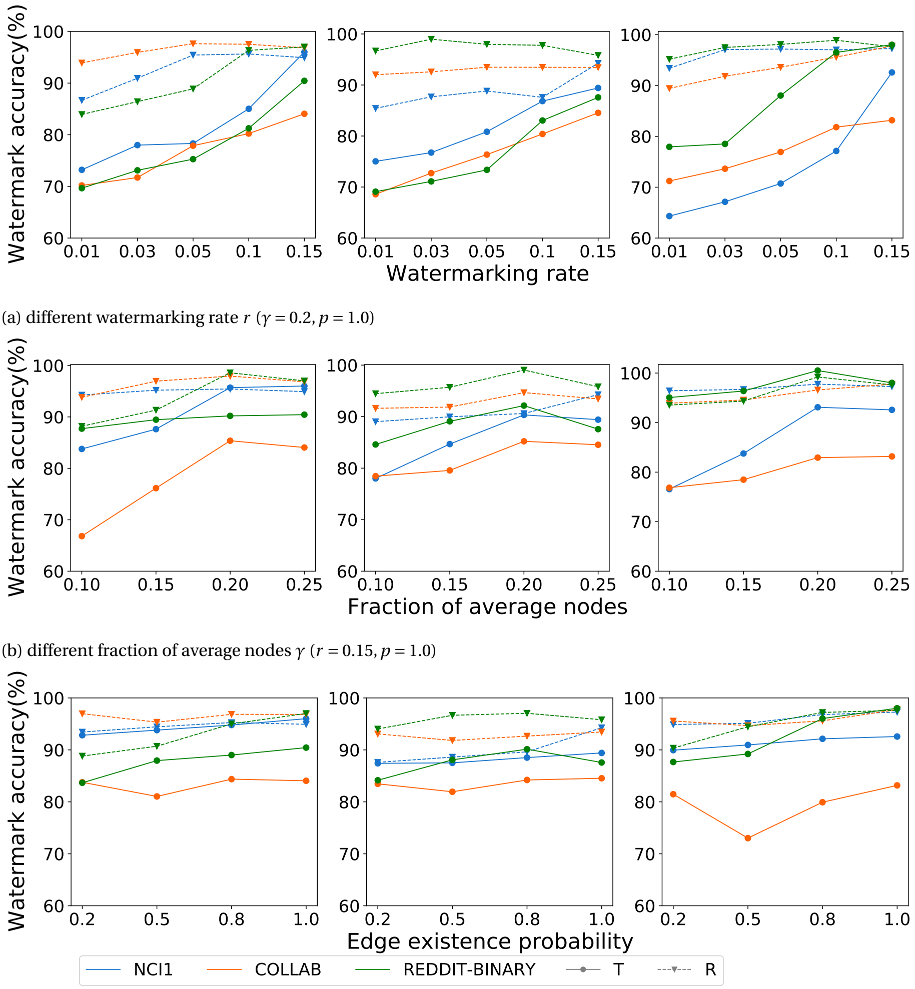
that the model owner can use a very small watermarking rate, e.g., 0.01 to watermark the models. From Figure 5.3b, in terms of watermark graph size from $\gamma = 0.1$ to $\gamma = 0.20$, the watermark accuracy of all three models and datasets gradually increases, and then there is no significant increase (even slight decrease in some cases, e.g., REDDIT-BINARY) from $\gamma = 0.20$ to $\gamma = 0.25$. When the watermark graph gets larger, it is intuitive that the model can learn the watermark pattern easier and better. With continuous growth in the size of the watermark graph, there may not be enough model capacity to learn the watermark pattern. From Figure 5.3c, for the NCI1 and REDDIT-BINARY datasets, the watermark accuracy grows slightly with the increase in the edge existence probability of the watermark graph. For the COLLAB dataset, the watermark accuracy first decreases for the range $p = 0.2$ to $p = 0.5$ and then increases. The reason may be that when the watermark graph density is farther away from the graph density of the dataset, the trained model is more likely to recognize the watermark graph successfully. The watermark accuracy is the lowest when $p = 0.5$ for the COLLAB dataset, which has a density of 0.5089. Moreover, there is no apparent increase for the other two datasets, which have a density of 0.0889 and 0.0218, respectively.

Based on the analysis of the results in Figure 5.3 and the later experimental results about the impact of watermarking GNNs on the original task, we set the parameters for the graph classification task as follows: $r = 0.15, \gamma = 0.2, p = 1.0$. Specifically, Table 5.9 shows the watermark accuracy of model m_w for the graph classification task with the selected parameters. For the binary-class datasets (NCI1 and REDDIT-BINARY), the accuracy on D^t_{wm} is around 90% while that of COLLAB is around 80%. This can be explained since COLLAB is a multi-class dataset, and it requires more model capacity to learn the features of each class so that the model has fewer redundant neurons to learn the watermark pattern compared to the other datasets. The watermark accuracy on D^r_{wm} can mostly reach around 95% for all datasets.

Table 5.10: Watermark accuracy for the node classification.

Dataset	Watermark Accuracy (%)		
	GCN	GAT	GraphSAGE
Cora	97.56	96.23	97.15
CiteSeer	98.05	97.73	99.22

Node classification. For the node classification task, the generated watermarked



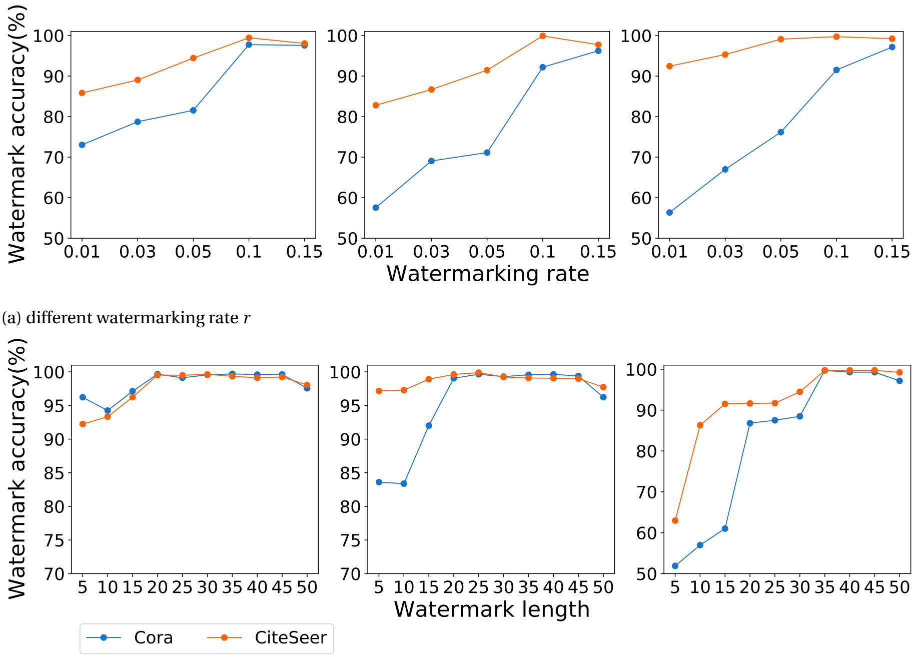
(a) different watermarking rate r ($\gamma = 0.2, p = 1.0$)

(b) different fraction of average nodes γ ($r = 0.15, p = 1.0$)

(c) different edge existence probability p ($r = 0.15, \gamma = 0.2$)

Figure 5.3: Watermark accuracy on graph classification task (DiffPool (left), GIN (center), GraphSAGE (right), T: first watermarked data generation strategy, R: second watermarked data generation strategy).

data is decided by two parameters (watermarking rate r and feature watermark length l). The watermark accuracy of three GNN models, i.e., GCN, GAT, and GraphSAGE, for the node classification task is shown in Figure 5.4. From Figure 5.4a, for all datasets and models, the watermark accuracy has a dramatic rise with r ranging from 0.01 to 0.1 and a slight increase when r is in the range of $[0.1, 0.15]$. From Figure 5.4b, for the GCN and GAT models, there is a significant increase between $l = 5$ and $l = 20$ and when l continues rising to $l = 50$, there is no obvious effect for both datasets. For the GraphSAGE model, the watermark accuracy gradually increases from $l = 5$ to $l = 35$ and stays steady. This is expected since, with more nodes embedded in the feature watermark, the GNN



(a) different watermarking rate r

(b) different watermark length l

Figure 5.4: Watermark accuracy on node classification task (GCN (left), GAT (center), GraphSAGE (right)).

model can learn the watermark pattern better. Additionally, the GNN model can better memorize the watermark pattern with a larger watermark. However, there is a decrease in the watermark accuracy for all datasets, as shown in Figure 5.4b. We believe this is because when the watermark size gets too large, e.g., $l = 50$, the GNN model does not have adequate capacity left to learn the watermark pattern well. Moreover, for watermarks with a watermark length of less than 35, the watermark accuracy for GCN and GAT models is higher than the GraphSAGE model, indicating that GCN and GAT models learn small watermarks better than the GraphSAGE model. Since transductive learning has the advantage of being able to directly use training patterns while deciding on a test pattern [11], it is easier for the GCN and GAT models (under transductive learning setting) to learn the watermark pattern of specific size than for the GraphSAGE model (under inductive learning setting). Considering the results in Figures 5.4 and 5.7 (which will be further analyzed later), we set the parameters for the node classification task as follows: $r = 0.15, l = 20$ for the GCN and GAT models, and $r = 0.15, l = 35$ for the GraphSAGE model. Table 5.10 shows the watermark accuracy of three models for the node classification task with the selected parameters.

We also compare our watermarking mechanism on node classification with the state-of-the-art [167], as shown in Figure 5.5. With the increasing watermarking rate, the watermark accuracy in that work declines to 79% and 38% for Cora and CiteSeer, respectively. Contrarily, in our watermarking method, the watermark accuracy keeps increasing to 100% for both datasets. The decline of watermark accuracy is consistent and explained in [167]. Thus, our watermarking method can achieve similar or higher watermark accuracy than the state-of-the-art.

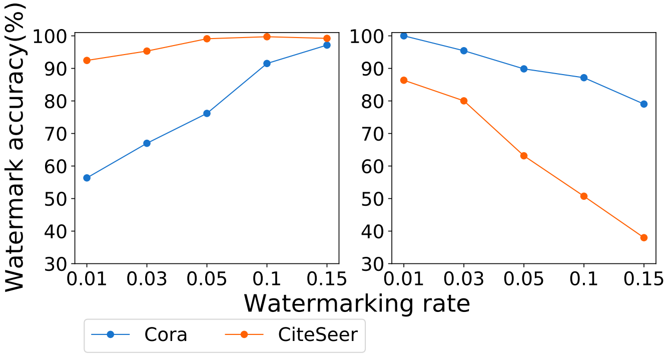
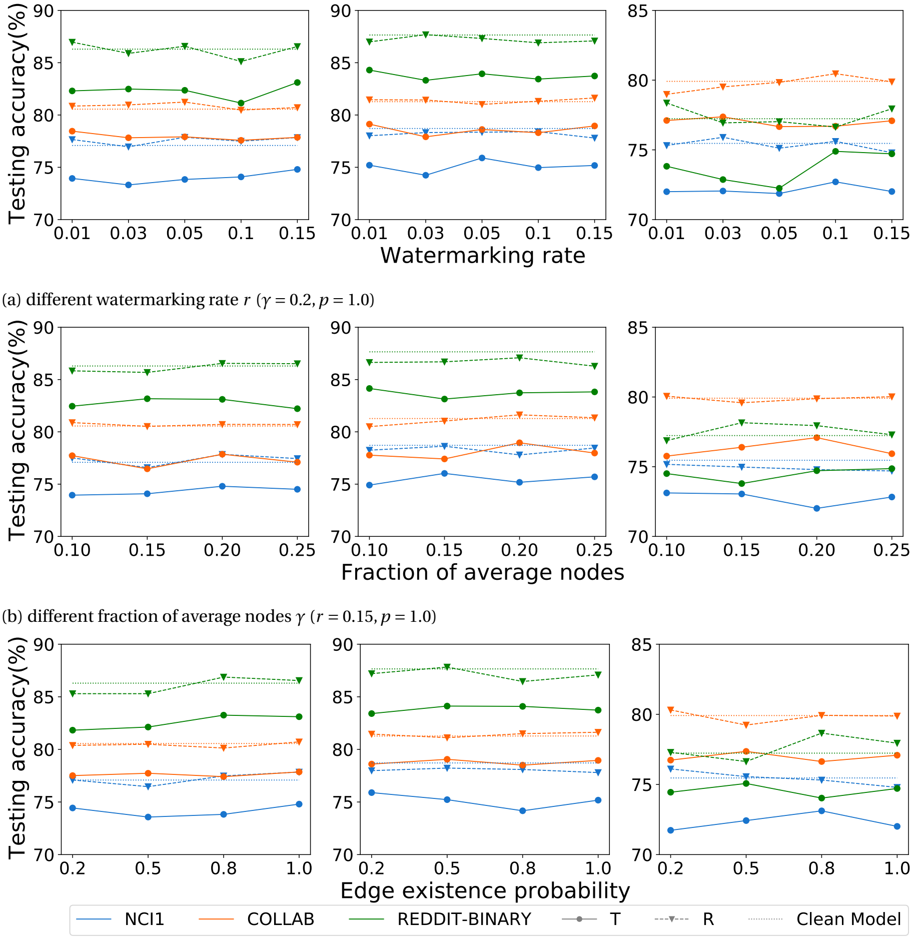


Figure 5.5: Comparison between our method (left) and [167] (right) on GraphSAGE model (inductive learning setting).

Impact on the original task. To measure the impact of our watermarking mechanism on the watermarked model's original task, we measure the accuracy of models on the normal test data. The test data is not used to train the host model. Figures 5.6 and 5.7 illustrate the testing accuracy of models with and without embedding watermarks under different variants for graph and node classification tasks, respectively. We use the testing accuracy of the clean model as the baseline, i.e., if the testing accuracy of the watermarked model is close to the baseline, we can confirm that our watermarking mechanism will not affect the watermarked model's original task. From Figure 5.6, for all three models and datasets, the testing accuracy of the second watermarked data generation strategy is always much closer to the baseline than the first strategy, which means the second strategy has less impact on the model's original task. In the first strategy, the watermarked data is generated by embedding a watermark into sampled training data, so it is possible to embed a watermark into a graph structure, which has a critical effect on the final prediction. As a result, the watermarking process of the first strategy will probably affect the parameters in the networks used for the original task. In the second strategy, random graphs are used as the watermark carrier data, so the model will try to explore redundancy in the network capacity to learn the watermark pattern while not affecting the original task. On the other hand, in the second approach, the watermark is embedded via random graphs without affecting the distribution of the original training dataset. Therefore, it has fewer side effects than the first approach.

For node classification (Figure 5.7), the watermarking rate has a negligible impact on the testing accuracy, while when the watermark length is larger than 20, there is a significant reduction in the testing accuracy for GCN and GAT models. For GraphSAGE, the testing accuracy fluctuates with the increase in watermark length. Referring to the watermark accuracy in Figure 5.4b, one possible reason is that if the watermark size continuously increases, the model's redundant capacity will be fully occupied, and the model will use some neurons originally for the main task to keep high watermark accuracy.

Thus, the testing accuracy for GCN and GAT decreases when the watermark length is larger than 20. For GraphSAGE, the testing accuracy does not reduce significantly because the GraphSAGE model has more redundant neurons than the other two models. Tables 5.11 and 5.12 show the testing accuracy with the selected parameters for graph and node classification tasks, respectively. For the graph classification task, the testing accuracy of the first strategy is about 3% less than that of the second strategy. There is less than 1% clean accuracy drop for the node classification task.



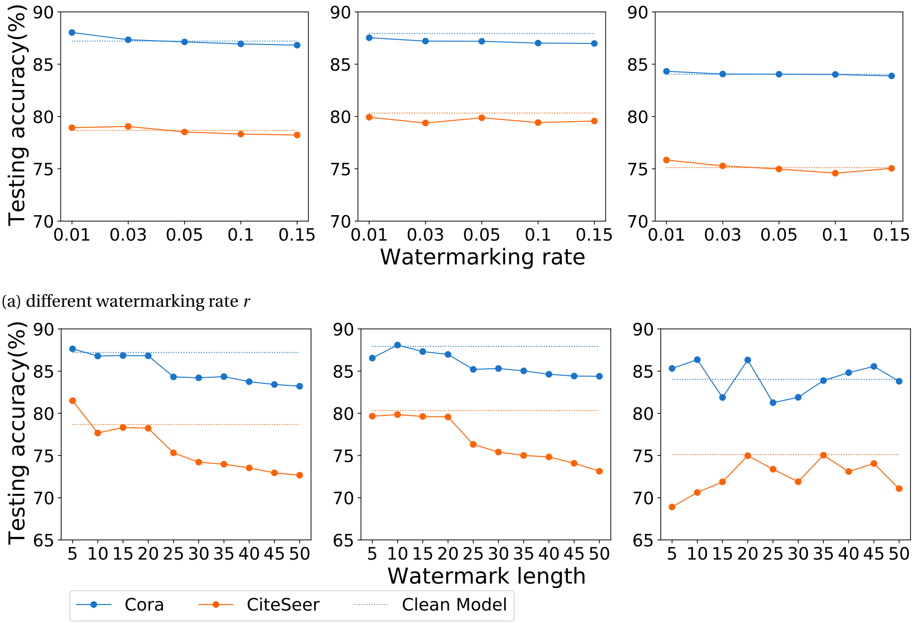
(a) different watermarking rate r ($\gamma = 0.2, p = 1.0$)

(b) different fraction of average nodes γ ($r = 0.15, p = 1.0$)

(c) different edge existence probability p ($r = 0.15, \gamma = 0.2$)

Figure 5.6: Testing accuracy on graph classification task (DiffPool (left), GIN (center), GraphSAGE (right), T: the first watermarked data generation strategy, R: the second watermarked data generation strategy).

Impact of adaptive attacker. In our threat model, we assume that the watermarking-relevant data (the watermarked data and the hyperparameters used to generate it) is securely stored after the training, which can be achieved through cryptographic tech-



(a) different watermarking rate r

(b) different watermark length l

Figure 5.7: Testing accuracy on node classification task (GCN (left), GAT (center), GraphSAGE (right)).

Table 5.11: Testing accuracy for graph classification task ($r = 0.15, \gamma = 0.2, p = 1.0$).

(a) NCI1

Testing Accuracy (%)	DiffPool	GIN	GraphSAGE
Clean Model	77.09	78.71	75.46
D_{wm}^{t}	74.80	75.18	72.01
D_{wm}^{r}	77.85	77.80	74.79

(b) COLLAB

Testing Accuracy (%)	DiffPool	GIN	GraphSAGE
Clean Model	80.56	81.27	79.91
D_{wm}^{t}	77.86	78.96	77.09
D_{wm}^{r}	80.71	81.63	79.88

(c) REDDIT-BINARY

Testing Accuracy (%)	DiffPool	GIN	GraphSAGE
Clean Model	86.30	87.65	77.23
D_{wm}^{t}	83.11	83.73	74.71
D_{wm}^{r}	86.55	87.08	77.95

niques. Additionally, to the best of our knowledge, there is no work on reversing backdoor triggers in GNNs, meaning that the attackers cannot retrieve our watermarked data

Table 5.12: Testing accuracy for the node classification.

Dataset	Testing Accuracy (%) ($CleanModel\|D_{wm}^t$)		
	GCN	GAT	GraphSAGE
Cora	87.21|86.82	87.94|86.98	84.04|83.88
CiteSeer	78.67|78.23	80.33|79.57	75.12|75.04

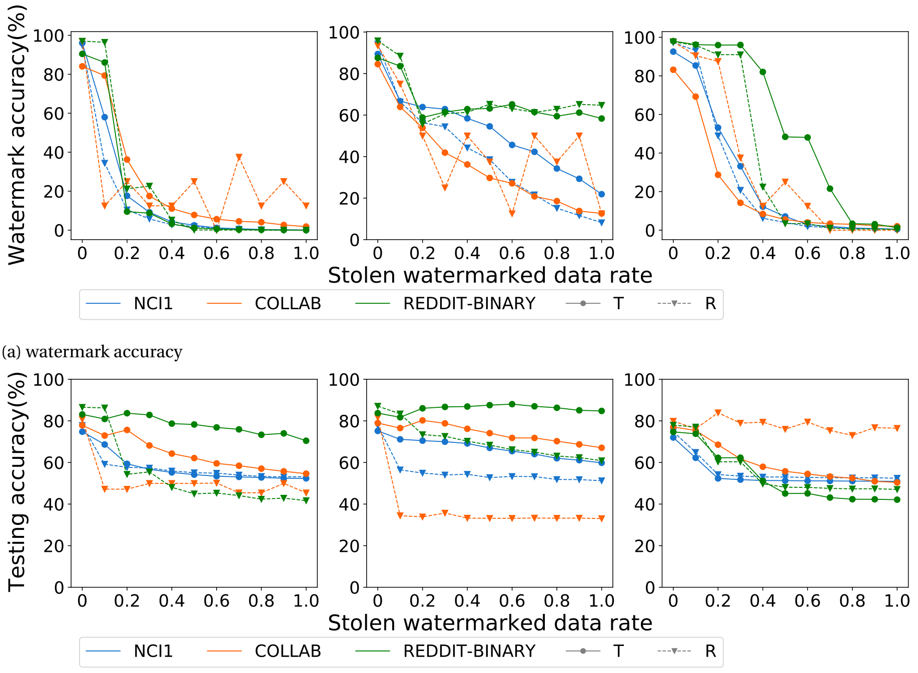
from the provided model. Thus, we consider it safe to assume our watermarked data is a secret that the adversaries have no access to, like the private key in encryption schemes. Nonetheless, we explore the performance of our watermarking mechanism against an adaptive attacker that has access to different percentages of our watermarked data and fine-tunes the watermarked model with the stolen watermarked data trying to remove the watermark functionality. Assuming the adaptive attacker steals from 0% to 100% of our watermarked data, we fine-tune the previously watermarked GNN model with the stolen watermarked data using the original labels. The results of watermarking performance with different stolen watermarked data rates for the graph and node classification tasks are shown in Figure 5.8 and 5.9, respectively. For graph classification, gradually, with the increase of the stolen watermarked data rate, the watermark accuracy decreases significantly, i.e., reducing to 0%. There is also an obvious accuracy drop on the main task, e.g., more than 30% decrease for DiffPool on the NCI1 dataset. This phenomenon is known as *catastrophic forgetting* in fine-tuning, i.e., after fine-tuning, the trained model tends to perform poorly in the source domain [149]. Thus, it is difficult for the watermarked GNN model to unlearn the watermark functionality without influencing the main task, which can also be observed in [23]. In contrast, for node classification (Figure 5.9), the watermark accuracy drops dramatically with more than 10% stolen watermarked data while the testing accuracy stays steady. This can be explained since, in node classification, we insert the watermark only to the feature vector, while in graph classification, the watermark also alters the graph structure. Thus, the watermark pattern in node classification contains less information, and it is easier to remove it from the trained model without affecting the performance on the main task. We believe the performance difference in watermark unlearning through fine-tuning between graph and node classification tasks is an interesting finding, and we plan to investigate it in the future.

5.3.2. ON THE WATERMARKING REQUIREMENTS

Next, we explain the well-known watermarking requirements for neural networks [13] and how our watermarking framework achieves them.

Robustness: resistance of the watermarking against the modifications that can be caused by malicious perturbations[7] or benign processing. A robust GNN watermarking should be recoverable even after the model has been modified by fine-tuning and network pruning. As shown in Section 5.4, our watermarking model achieves robustness against the modifications in the model.

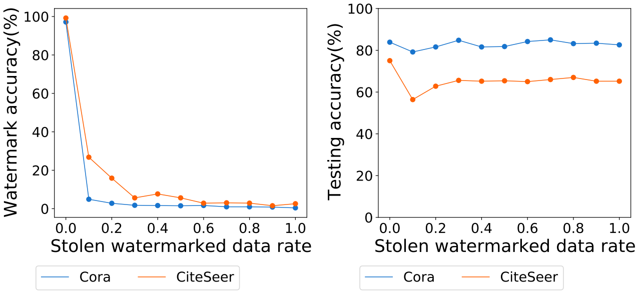
[7] In [75], this is separately defined as the *security* requirement.



(a) watermark accuracy

(b) testing accuracy

Figure 5.8: Watermarking performance for different stolen watermarked data rates on graph classification task (DiffPool (left), GIN (center), GraphSAGE (right)).



(a) watermark accuracy

(b) testing accuracy

Figure 5.9: Watermarking performance for different stolen watermarked data rates on node classification (GraphSAGE).

Fidelity: maintaining the *quality* of the watermarked object while watermarking. A GNN watermarking model satisfies the fidelity requirement if it does not significantly degrade the performance of the GNN model. Our experimental results presented in Tables 5.11 and 5.12 show that our watermarking (with random graphs) on graph classification and node classification tasks leads to a smaller than 1% decrease on the accuracy of the original task. Detailed discussion is given in Section 5.3.1.

Capacity: the watermark's capability to carry information, and two watermarking schemes, zero-bit, and multi-bit, are distinguished by it [13]. Our work applies zero-bit watermarks because they do not carry additional information, such that they solely serve to indicate the presence or the absence of the watermark in a model.

Integrity: the correct classification of the watermarking. A GNN watermarking mechanism satisfying integrity should have low FPR and FNR.[8] The false positive requirement states that a benign model not copied from the watermarked GNN model should not be seen as a malicious copy. The false negative implies that a malicious copy of the watermarked GNN model should be classified as a copy. As mentioned in Sections 5.2.4 and 5.3, we select a watermark accuracy threshold for each dataset and model to make sure the FPR and FNR are less than 0.0001.

Generality: the generalization of the watermarking method. A GNN watermarking would have generality if it is not tailored to a specific model but can be applied to other architectures or models. Our watermarking method has been evaluated in several state-of-the-art GNN models. In addition, our watermarking method does not depend on the architectures of the GNN models and can be applied to any other GNN models and graph datasets.

Efficiency: the overhead required for embedding and verifying a watermark into an object. An efficient GNN watermarking model should not add too much computational cost by adding the watermark. In our watermarking method, according to the watermark embedding algorithm 8 and the watermarking rate we set, the overhead in embedding a watermark into a GNN model is retraining a clean GNN model with 30% or 15% of the original training data. The overhead in verifying the watermark is the number of the watermarked data related to the watermarking rate, i.e., between 0.01 and 0.15. As explained in Section 5.3.1, in our watermarking method, the model owners can use low watermarking rates, e.g., 0.01, to watermark their models.

In [13], the authors also defined *secrecy* property concerning the detectability of the presence of the watermark. This property is out of scope for our watermarking framework because we expect our watermarking mechanism still works even if the adversaries know the existence of the watermark. Nevertheless, the watermark itself should be secret to the adversaries.

5.4. Robustness Against Backdoor Defenses

Our watermarking method is based on backdoor attacks. If attackers suspect the model is protected by a watermark based on backdoor attacks, they would like to use backdoor removal techniques (defenses) to remove the backdoor, i.e., our watermark. Consequently, it is intuitive here to explore whether our watermarking method is resistant to state-of-the-art defenses against backdoor attacks. The state-of-the-art defenses against backdoor attacks can be summarized in four categories: input reformation, input filtering, model sanitization, and model inspection [100]. NeuralCleanse (NC) [128] is the most representative defense in the model inspection defense category. However, it is not feasible to be applied in this work because (1) NC requires a large number of input

[8] In [75], the false negative case is the *reliability* requirement.

samples to achieve good performance [85] while in our work, the plagiarizer has no access to the training data, and (2) NC cannot reverse engineer large triggers [36, 85, 148], while in our work, ideally, there is no restriction on the watermark size. Similarly, as one of the most representative defenses in the input filtering defense category, Activation-Clustering (AC) [20] is not applicable because it requires access to poisoned training data to remove the backdoor, but in our setting, the plagiarizer has no knowledge of the watermarked data. On the other hand, Randomized-Smoothing (RS) [26] (input reformation defense) can be applied because it can only reform the input samples without the requirement of knowledge of watermarked data. Three model modifications (i.e., fine-tuning, model pruning, and fine-pruning), which can be categorized into the model sanitization defenses, are also applied in our work to explore the robustness of our watermarked model.

Next, we investigate the robustness of our watermarked model against a state-of-the-art model extraction technique: knowledge distillation and four state-of-the-art defenses against backdoor attacks: randomized subsampling, fine-tuning, pruning, and fine-pruning.

Robustness against knowledge distillation. Knowledge distillation aims at transferring knowledge from a teacher model to a student [34]. It has been used in the model extraction attacks where the teacher model is the victim model, and the student model is the stolen extracted model [123]. Here, we suppose the plagiarizer applies knowledge distillation to extract the knowledge from the host model to train the plagiarized model, and we explore the robustness of our watermarking method against this attack. Specifically, we follow the offline distillation strategy in [48] since the host model is pre-trained. We assume the student model has the same model structure as the teacher model, and we use half of the test data to be the training data for the knowledge distillation.[9] We evaluate the distilled model, i.e., the student model, with the second half of the test data.

Tables 5.13 and 5.14 show the watermark accuracy and testing accuracy of the model after knowledge distillation on the graph classification task. We can observe that the watermark accuracy decrease is less than 3.25% for two watermarked data generation strategies, for all datasets and models, except 4.99% decline for the NCI1 dataset with the GIN model. As for the testing accuracy, we see that knowledge distillation has little impact on the model's original task. For the node classification task, the watermarking performance after the knowledge distillation is shown in Table 5.15. As we can see, the watermark accuracy after the knowledge distillation even increases a little on the node classification task. Since the knowledge distillation can improve the generalization of the student model [119], it may reduce the overfitting in the original model. Moreover, the testing accuracy decreases negligibly except for CiteSeer with the GAT and GraphSAGE models. One possible explanation is that in our work, the GAT and GraphSAGE models are more complex than GCN, so it is more difficult for these two models to transfer the knowledge completely from the teacher model to the student model. Based on the observations above, we can claim that knowledge distillation can transfer the knowledge of the host model on the original task to the student model successfully. Still, it can also transfer the watermarking function, which means our watermarking mechanism is

[9] We assume that the plagiarizer has access to the testing data aiming to explore the robustness of our method under a strong adversary.

robust against knowledge distillation.

Table 5.13: Accuracy on watermarked data after knowledge distillation for graph classification task ($r = 0.15, \gamma = 0.2, p = 1.0$).

Dataset	Watermark Accuracy (%) ($D^t_{wm}\|D^r_{wm}$)		
	DiffPool	GIN	GraphSAGE
NCI1	92.99|93.88	87.90|89.28	90.36|96.27
COLLAB	83.11|95.87	82.66|92.75	80.77|94.54
REDDIT-BINARY	87.74|96.92	89.71|97.45	97.53|97.67

Table 5.14: Testing accuracy after knowledge distillation for graph classification task ($r = 0.15, \gamma = 0.2, p = 1.0$).

Dataset	Testing Accuracy (%) ($D^t_{wm}\|D^r_{wm}$)		
	DiffPool	GIN	GraphSAGE
NCI1	75.92|77.30	78.50|78.88	76.83|76.46
COLLAB	76.18|80.79	77.67|81.09	75.08|79.60
REDDIT-BINARY	83.63|86.35	83.82|87.54	77.65|78.25

Table 5.15: Watermarking performance after knowledge distillation on the node classification task.

Dataset	Watermark Accuracy (%) | Testing Accuracy (%)		
	GCN	GAT	GraphSAGE
Cora	99.64|88.22	98.98|86.28	99.23|82.19
CiteSeer	99.51|78.26	99.59|73.65	99.30|73.54

Robustness against fine-tuning. As discussed before, training a well-performed GNN model from scratch requires a large amount of training data and powerful computing resources. Fine-tuning is a practical attack on GNNs since it can be used to apply existing state-of-the-art models to other but similar tasks with less effort than training a network from scratch when sufficient training data is not available [157]. Therefore, fine-tuning is likely to be used by a suspect to train a new model on top of the stolen model using only a small amount of training data.

In this experiment, for each dataset, we use half of the test data to fine-tune the previously trained watermarked GNN model, and the second half is used to evaluate the new model. Then, we use the watermark accuracy to determine whether the watermark embedded in the previously trained GNN model stayed effective in the new model. Additionally, the testing accuracy is used to evaluate the performance of the newly trained model on its original task. Tables 5.16 and 5.17 show the watermark accuracy and testing accuracy of the model after fine-tuning for the graph classification task. Comparing the results from Tables 5.9 and 5.16, we can see that fine-tuning does not significantly reduce (less than 4.65%) the watermark accuracy for all datasets and models. Triggers rarely appear in the fine-tuning dataset; consequently, the backdoor functionality will not be eliminated [165]. As for the testing accuracy, from Tables 5.11 and 5.17, we can observe that after fine-tuning, the embedded watermark in the model still has only a slight effect on the model's original task. For the node classification task, fine-tuning is not feasible for transductive learning-based GNN models because, once the training data change, the model should be retrained from scratch. Thus, we show here the results for the GraphSAGE model (inductive learning) for the node classification task, as shown in Table 5.18, and the observations are similar to that for the graph classification task. To compare our watermarking mechanism with the state-of-the-art [167], we also present the watermark performance for the method in [167] after fine-tuning in Table 5.18. As we can observe, our method achieves higher watermark accuracy after fine-tuning than the state-of-the-art, e.g., more than 10% higher on the CiteSeer dataset. Regarding the testing accuracy after fine-tuning, the difference between our method and [167] is less than 0.5% for both datasets.

Table 5.16: Watermark accuracy after fine-tuning for graph classification task ($r = 0.15, \gamma = 0.2, p = 1.0$).

Dataset	Watermark Accuracy (%) $(D^t_{wm}\|D^r_{wm})$		
	DiffPool	GIN	GraphSAGE
NCI1	94.00|93.69	88.71|89.64	91.92|97.17
COLLAB	83.71|96.50	82.96|91.83	80.48|94.11
REDDIT-BINARY	87.61|96.78	90.11|97.87	98.68|97.90

Table 5.17: Testing accuracy after fine-tuning for graph classification task ($r = 0.15, \gamma = 0.2, p = 1.0$).

Dataset	Testing Accuracy (%) $(D^t_{wm}\|D^r_{wm})$		
	DiffPool	GIN	GraphSAGE
NCI1	73.61|77.15	75.05|77.84	71.92|74.89
COLLAB	76.10|80.67	77.63|80.92	75.93|78.31
REDDIT-BINARY	82.10|86.20	80.63|86.89	73.82|77.85

Table 5.18: Watermarking performance of GraphSAGE model after fine-tuning on the node classification task.

Dataset	Watermark Accuracy (%)		Testing accuracy (%)	
	Ours	[167]	Ours	[167]
Cora	98.51	97.84	84.39	84.51
CiteSeer	99.45	86.36	75.10	75.41

Robustness against model pruning. Model pruning is a technique to develop a neural network model that is smaller and more efficient by setting some parameters to zero while maintaining the performance on the primary task [162]. We apply the pruning algorithm used in [125], which prunes the parameters whose absolute values are very small. Specifically, for all the watermarked models, we remove a certain number of parameters with the smallest absolute values by setting them to zero. The ratio between the number of pruned parameters and the total number of parameters is the pruning rate (here from 10% to 90%). Then, we measure the watermark and the testing accuracy of the pruned watermarked model.

Tables 5.19 and 5.20 present the watermarking performance after model pruning on the graph and node classification tasks, respectively. We firstly take the results for NCI1 and Cora as examples. For the NCI1 dataset, even when 40% of the parameters are pruned, our watermarked model still has a high watermark accuracy, i.e., less than 1% drop in all models. Especially for the GraphSAGE model, there is only a 0.68% drop even though 70% of the parameters are pruned. We can also observe that when 90% of the parameters are pruned, the watermark accuracy drops dramatically for all models, e.g., it drops to less than 10% for the DiffPool model. We also notice that in this case, there is a significant testing accuracy drop for the model (more than 20%), which means the plagiarizer has to take the expense of dramatically degrading the model's performance on the original task to eliminate our watermark. As for the Cora dataset, the watermark accuracy decreases gradually, as well as the testing accuracy for GCN and GAT models. For the GraphSAGE model, model pruning also leads to a more obvious watermark accuracy drop, i.e., more than 60%, as well as an apparent testing accuracy drop (more than 10%). Thus, our watermarking mechanism is generally robust to model pruning. The plagiarizer can only eliminate our watermarks with the cost of a considerable accuracy drop in the main task. The watermarking performance after model pruning on the COLLAB, REDDIT-BINARY, and CiteSeer datasets is shown in Tables 5.21, 5.22, and 5.23, respectively. For the COLLAB dataset, when the pruning rate is less than 50%, the watermark accuracy drops slightly. With a pruning rate of more than 50%, the watermarking accuracy decreases significantly, as well as the testing accuracy for all models. The results for REDDIT-BINARY have the same behavior. As for the CiteSeer dataset, even with 90% of the parameters pruned, the watermark accuracy on the GCN and GAT models is still high, i.e., more than 99%. On the other hand, with more than 50% of the parameters pruned, the watermark accuracy on the GraphSAGE model drops dramatically, but the testing accuracy decreases significantly as well. Therefore, these results further verify that our watermarking mechanism is robust to model pruning, but the plagiarizer can still eliminate our watermarks with the cost of high accuracy drop in the main task.

Table 5.19: Watermarking performance on graph classification task after model pruning (NCI1).

Pruning	DiffPool		GIN		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
rate	$(D^t_{wm} \mid D^r_{wm})$	$(D^t_{wm} \mid D^r_{wm})$	$(D^t_{wm} \mid D^r_{wm})$	$(D^t_{wm} \mid D^r_{wm})$	$(D^t_{wm} \mid D^r_{wm})$	$(D^t_{wm} \mid D^r_{wm})$
10%	77.77%|77.86%	95.52%|95.40%	78.21%|77.81%	89.96%|90.38%	74.66%|74.83%	93.06%|97.65%
20%	77.72%|77.84%	95.47%|95.26%	77.79%|77.62%	89.82%|90.51%	74.77%|74.46%	92.83%|97.68%
30%	77.61%|77.69%	94.70%|95.48%	77.24%|76.95%	89.41%|90.63%	73.56%|73.93%	92.78%|97.55%
40%	76.79%|77.25%	94.68%|95.36%	72.93%|71.76%	89.33%|90.50%	73.32%|72.36%	92.71%|97.47%
50%	70.77%|74.71%	86.89%|95.21%	63.34%|61.31%	83.96%|90.23%	70.08%|70.12%	92.58%|97.54%
60%	60.17%|66.46%	68.92%|84.61%	57.60%|56.66%	76.53%|88.43%	61.62%|62.71%	92.65%|97.45%
70%	52.46%|55.14%	34.31%|75.71%	56.39%|52.79%	73.90%|89.45%	53.58%|52.91%	92.38%|97.48%
80%	50.66%|51.35%	7.58%|60.71%	54.11%|51.55%	62.71%|88.65%	50.26%|51.10%	89.17%|97.45%
90%	50.66%|50.36%	7.58%|51.00%	52.00%|50.47%	58.77%|89.54%	49.67%|49.07%	76.90%|82.74%

Table 5.20: Watermarking performance on node classification task after model pruning (Cora).

Pruning rate	GCN		GAT		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
10%	86.51%	99.51%	85.23%	97.05%	83.40%	99.62%
20%	86.49%	98.50%	85.22%	96.80%	83.37%	99.58%
30%	86.24%	98.27%	85.26%	96.46%	83.24%	99.23%
40%	85.68%	98.20%	85.24%	95.80%	82.45%	98.12%
50%	83.63%	97.39%	85.29%	95.55%	82.35%	98.08%
60%	82.60%	97.14%	85.28%	94.80%	80.94%	98.04%
70%	82.52%	96.70%	85.25%	93.69%	80.37%	71.48%
80%	82.41%	96.40%	85.19%	93.46%	78.20%	42.48%
90%	82.20%	90.64%	84.97%	93.30%	72.00%	34.64%

Table 5.21: Watermarking performance on graph classification task after model pruning (COLLAB).

Pruning	DiffPool		GIN		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
rate	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$
10%	80.36%|80.69%	85.30%|98.01%	81.39%|81.66%	85.24%|94.08%	79.65%|79.89%	82.89%|97.19%
20%	80.12%|80.55%	85.35%|97.88%	80.71%|81.49%	84.70%|94.02%	79.59%|79.90%	82.91%|97.10%
30%	79.89%|80.15%	84.71%|97.91%	79.28%|81.44%	83.49%|93.56%	79.37%|79.69%	82.90%|96.39%
40%	78.43%|79.28%	84.78%|97.53%	77.13%|80.84%	83.34%|92.46%	78.96%|79.57%	82.72%|96.50%
50%	70.29%|76.36%	70.79%|84.06%	73.46%|79.24%	78.70%|69.19%	76.97%|79.34%	81.45%|96.48%
60%	59.58%|64.80%	64.05%|63.23%	71.12%|76.55%	78.16%|43.72%	72.91%|77.91%	76.84%|96.43%
70%	50.64%|47.63%	61.47%|35.44%	67.22%|70.30%	75.53%|43.72%	62.64%|72.64%	62.68%|96.32%
80%	47.93%|36.60%	63.13%|35.44%	60.99%|63.52%	73.98%|43.72%	47.54%|57.77%	44.52%|82.91%
90%	47.90%|32.53%	63.09%|28.50%	55.86%|42.34%	74.33%|30.99%	35.17%|37.26%	36.40%|62.39%

Table 5.22: Watermarking performance on graph classification task after model pruning (REDDIT-BINARY).

Pruning	DiffPool		GIN		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
rate	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$
10%	86.75%|86.55%	90.05%|98.40%	87.41%|87.05%	91.81%|99.09%	78.30%|77.94%	99.35%|98.87%
20%	86.78%|86.58%	90.15%|97.99%	87.20%|87.11%	91.34%|98.84%	78.33%|77.89%	99.30%|99.09%
30%	86.79%|86.24%	89.98%|98.40%	87.30%|86.84%	91.14%|98.69%	78.33%|77.77%	99.48%|99.00%
40%	86.58%|86.42%	89.79%|98.41%	87.26%|85.92%	91.24%|98.78%	77.96%|77.74%	99.00%|98.82%
50%	86.91%|86.55%	89.10%|98.63%	84.76%|84.98%	91.75%|98.77%	77.70%|77.48%	99.05%|98.08%
60%	86.43%|85.70%	89.43%|97.73%	81.45%|83.57%	88.26%|96.09%	76.85%|77.56%	99.66%|98.91%
70%	82.61%|82.89%	88.94%|93.02%	77.99%|80.72%	77.11%|76.73%	73.40%|76.34%	99.43%|99.20%
80%	72.62%|74.39%	80.58%|87.82%	73.48%|76.22%	70.71%|82.72%	64.95%|72.43%	99.48%|98.81%
90%	60.69%|63.94%	31.76%|66.76%	68.11%|70.56%	67.31%|77.22%	53.79%|59.67%	98.36%|99.08%

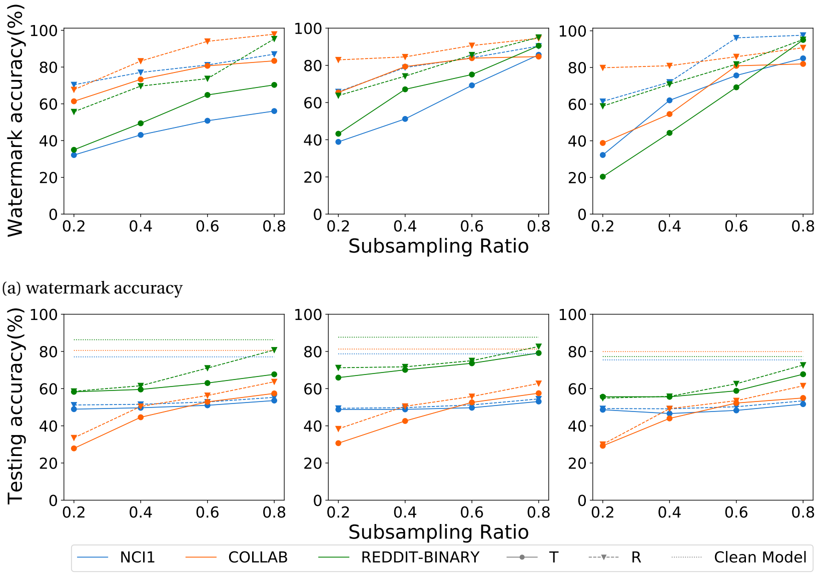
Robustness against randomized subsampling. Randomized smoothing [26] is a state-of-the-art technique for building robust machine learning. For binary data, a randomized smoothing method based on randomized subsampling can achieve promising certified robustness [162]. Here, we explore the robustness of our watermarking method against randomized subsampling [143]. In particular, we apply a subsampling function

Table 5.23: Watermarking performance on node classification task after model pruning (CiteSeer)

Pruning rate	GCN		GAT		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
10%	74.50%	99.45%	78.55%	99.59%	80.25%	93.90%
20%	74.42%	99.42%	78.42%	99.55%	80.34%	93.82%
30%	74.47%	99.40%	78.57%	99.57%	80.24%	93.86%
40%	74.45%	99.35%	78.49%	99.60%	80.28%	93.90%
50%	74.41%	99.31%	78.51%	99.52%	80.45%	93.80%
60%	74.35%	99.24%	78.35%	99.59%	75.32%	82.79%
70%	74.22%	99.01%	78.32%	99.53%	74.27%	49.52%
80%	73.93%	99.74%	78.08%	99.43%	73.20%	16.02%
90%	72.93%	99.95%	77.59%	99.87%	72.00%	8.46%

over a given graph G to create a set of subsampled graphs $G_{s_1}, G_{s_2}, \ldots, G_{s_n}$ by keeping some randomly subsampled nodes in G and removing the remaining nodes. We then feed the subsampled graphs to the watermarked model and take a majority voting of the predictions over such graphs as G's final prediction. In the randomized subsampling technique, there is an important parameter β (subsampling ratio) that specifies the randomization degree. For example, if $\beta = 0.2$, for the graph classification task, we randomly keep 20% of G's nodes and remove the rest of the nodes, and for the node classification task, we randomly keep the 20% of the nodes' features in the graph and set the remaining features to 0. Similar to [143], in this chapter, the randomized subsampling is only used to work on graphs instead of training smoothed models (thus, we use it to make robust samples). Figures 5.10 and 5.11 show the watermarking performance with different subsampling ratios in the graph classification task and the node classification task, respectively. We see that for the graph classification task, a decrease of β decreases the watermark accuracy, and in most cases, the testing accuracy is significantly lower than the clean model's. For the node classification task, the reduction of the subsampling ratio leads to a significant drop in watermark accuracy, but the testing accuracy is still close to the clean model. Therefore, randomized subsampling is not effective in attacking our watermarking mechanism for the graph classification task, as the penalty in the testing accuracy is unacceptable. However, it is effective for the node classification task.

Robustness against fine-pruning. Fine-pruning is an effective defense against backdoor attacks on deep neural networks, combining two promising defenses, pruning and fine-tuning [82]. As shown in the results for the two defenses above, i.e., fine-tuning and pruning, neither is sufficient to eliminate the watermarking function. Therefore, we assume the plagiarizer applies fine-pruning, a more effective defense, to train a new model on top of the stolen model. We follow the settings in the experiments of two defenses before, i.e., model pruning and fine-tuning. The pruning ratio is set from 10% to 90%. We also take the results for NCI1 and two node classification task datasets as examples, as shown in Tables 5.24 and 5.25, respectively. We can observe that the test accuracy of the second watermarked data generation strategy reduces slightly, which is different from the model pruning, where the test accuracy drops significantly. Thus, fine-pruning is a more powerful technique than model pruning for the plagiarizer to try to steal the model



(a) watermark accuracy

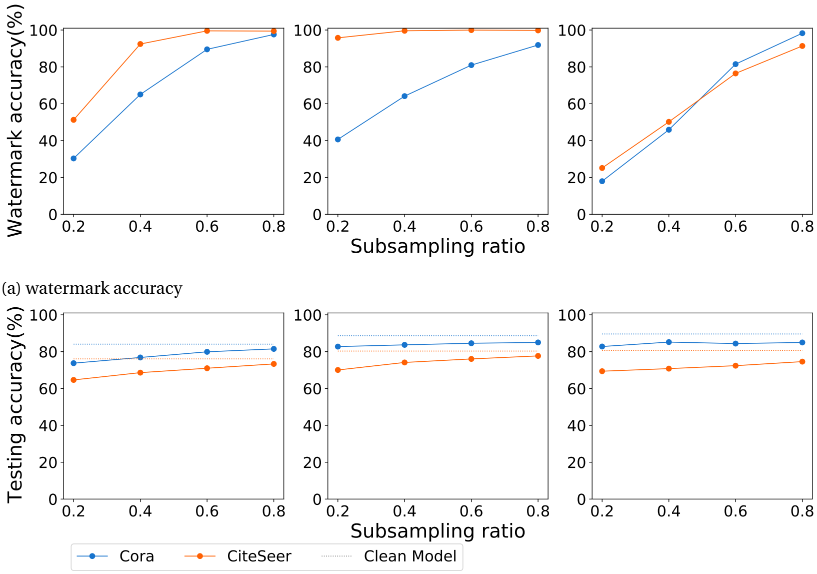
(b) testing accuracy

Figure 5.10: Watermarking performance for different subsampling ratios on graph classification task (DiffPool (left), GIN (center), GraphSAGE (right), T: first watermarked data generation strategy, R: second watermarked data generation strategy).

Table 5.24: Watermarking performance on graph classification task after fine-pruning (NCI1).

Pruning	DiffPool		GIN		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
rate	$(D_{wm}^t \mid D_{wm}^r)$	$(D_{wm}^t \mid D_{wm}^r)$	$(D_{wm}^t \mid D_{wm}^r)$	$(D_{wm}^t \mid D_{wm}^r)$	$(D_{wm}^t \mid D_{wm}^r)$	$(D_{wm}^t \mid D_{wm}^r)$
10%	81.32%|77.20%	95.69%|95.44%	82.32%|79.18%	90.24%|90.61%	74.96%|75.71%	93.46%|97.92%
20%	81.00%|77.43%	94.46%|92.10%	82.01%|79.12%	90.19%|89.45%	75.00%|75.51%	93.39%|97.29%
30%	81.16%|77.52%	93.46%|88.64%	81.57%|78.99%	90.07%|89.46%	74.58%|75.50%	92.79%|97.62%
40%	80.35%|77.43%	93.38%|88.44%	81.12%|79.06%	90.05%|89.54%	73.74%|75.43%	92.90%|97.02%
50%	79.20%|77.58%	89.78%|87.61%	80.10%|78.98%	90.14%|89.32%	72.37%|75.66%	92.71%|96.80%
60%	76.83%|77.48%	88.36%|86.80%	78.93%|78.92%	90.03%|89.17%	68.01%|75.17%	93.10%|96.75%
70%	75.14%|77.48%	88.10%|83.30%	75.74%|78.70%	89.89%|89.10%	61.86%|74.25%	92.66%|96.82%
80%	69.73%|77.20%	81.67%|82.87%	72.01%|77.78%	89.93%|89.30%	55.62%|73.95%	93.05%|96.77%
90%	66.71%|75.21%	67.42%|82.45%	66.93%|75.89%	82.64%|83.06%	50.71%|66.37%	92.91%|73.55%

while keeping the model's performance on the original task. As we can also see from the results on the NCI1 dataset, with a pruning rate of 80%, the watermark accuracy decreases less than 2% for the GIN and GraphSAGE models. For the DiffPool model, with the increasing of the pruning rate, the watermark accuracies for two watermarked data generation strategies drop to 67.42% and 82.45%, respectively. Still, they are larger than the corresponding thresholds, i.e., 53.5% and 72.0%, respectively. As for the node classification task, based on the explanation in the fine-tuning experiments, we show the results for the GraphSAGE model. We can see from Table 5.25 that when half of the parameters



(a) watermark accuracy

(b) testing accuracy

Figure 5.11: Watermarking performance for different subsampling ratios on node classification task (GCN (left), GCN (center), GraphSAGE (right)).

Table 5.25: Watermarking performance on the node classification task after fine-pruning (GraphSAGE).

Pruning rate	Cora		CiteSeer	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
10%	84.04%	99.70%	72.12%	99.27%
20%	83.45%	99.83%	71.37%	99.49%
30%	83.06%	99.53%	71.00%	99.14%
40%	83.45%	99.23%	71.56%	99.59%
50%	83.26%	99.07%	71.93%	99.65%
60%	82.48%	98.06%	71.56%	88.96%
70%	83.06%	69.18%	71.19%	51.60%
80%	71.70%	36.24%	61.93%	16.75%
90%	64.87%	26.71%	56.55%	8.86%

are pruned in the fine-pruning, the drop on the watermark accuracy is less than 1% for both datasets. With the pruning rate continuously increasing to 70%, the watermark accuracy decreases obviously, i.e., it drops to 69.18% and 51.60% for Cora and CiteSeer respectively, but it is still higher than the verification threshold. When the pruning rate is higher than 70%, the watermark accuracy reduces below the verification threshold, but simultaneously, the test accuracy drops dramatically. Thus, even if the plagiarizer performs fine-pruning to train a new model, our watermarking mechanism can verify the ownership of the model. The watermarking performance after fine-pruning on the COLLAB and REDDIT-BINARY datasets is shown in Tables 5.26 and 5.27, respectively. For the COLLAB dataset, with the increase in the pruning rate, the watermark accuracy gradually decreases. However, even with the 90% pruning rate, most of the watermark accuracy is still higher than the threshold in Table 5.7. The results for REDDIT-BINARY follow the same behavior. Therefore, these results further verify that our watermarking mechanism is robust to fine-pruning.

Table 5.26: Watermarking performance on graph classification task after fine-pruning (COLLAB).

Pruning	DiffPool		GIN		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
rate	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$
10%	71.77%|92.62%	85.37%|97.95%	69.55%|87.42%	85.21%|94.66%	69.07%|92.06%	82.93%|96.59%
20%	72.07%|92.71%	85.37%|80.59%	69.52%|87.56%	85.03%|88.29%	69.39%|91.98%	82.93%|95.19%
30%	70.50%|93.07%	85.33%|71.91%	68.04%|87.48%	83.97%|87.96%	69.20%|92.00%	82.91%|85.19%
40%	65.29%|93.08%	85.33%|71.88%	65.80%|87.07%	81.24%|87.77%	69.38%|91.93%	82.91%|87.12%
50%	60.62%|93.28%	85.31%|71.74%	62.80%|86.84%	81.08%|56.46%	69.20%|91.84%	82.90%|86.02%
60%	56.44%|93.25%	85.22%|71.66%	57.29%|86.10%	80.45%|50.09%	65.23%|91.87%	82.15%|85.77%
70%	53.53%|93.35%	84.73%|63.22%	53.28%|86.47%	79.86%|49.53%	60.01%|91.59%	82.00%|85.02%
80%	52.39%|93.16%	84.46%|63.14%	52.93%|85.95%	78.57%|43.72%	54.77%|91.00%	81.41%|83.92%
90%	52.45%|92.70%	84.20%|63.09%	52.44%|84.64%	78.04%|43.38%	51.94%|89.87%	80.23%|73.79%

Table 5.27: Watermarking performance on graph classification task after fine-pruning (REDDIT-BINARY).

Pruning	DiffPool		GIN		GraphSAGE	
	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.	Test Acc.	Watermark Acc.
rate	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$	$(D^t_{wm}\|D^r_{wm})$
10%	83.29%|95.52%	90.20%|98.59%	84.83%|93.35%	92.14%|99.04%	73.76%|88.37%	99.51%|99.21%
20%	83.02%|95.52%	88.02%|96.13%	84.47%|93.03%	90.73%|96.31%	73.67%|87.86%	99.96%|98.90%
30%	83.20%|95.52%	87.54%|96.08%	85.97%|92.82%	86.78%|98.35%	73.85%|87.77%	99.28%|99.32%
40%	84.84%|95.67%	86.45%|95.65%	85.10%|93.06%	86.67%|97.62%	73.94%|87.83%	99.80%|99.20%
50%	84.45%|95.91%	86.11%|95.60%	84.65%|93.09%	87.13%|97.19%	73.88%|88.07%	99.68%|98.54%
60%	84.72%|96.15%	74.72%|94.74%	83.51%|93.20%	66.76%|82.40%	73.49%|87.86%	99.46%|98.80%
70%	83.62%|95.94%	74.04%|93.58%	83.00%|93.65%	65.47%|82.07%	72.60%|87.56%	99.40%|98.42%
80%	82.66%|95.73%	71.52%|92.76%	80.18%|93.71%	35.34%|81.18%	72.30%|88.58%	99.50%|98.33%
90%	78.17%|95.94%	54.68%|84.63%	73.74%|93.74%	28.87%|79.52%	69.07%|90.00%	99.52%|98.39%

5.5. Conclusions and Future Work

This chapter proposed a watermarking framework for GNNs, which includes generating watermarked data with different strategies, embedding the watermark into the host model through training, and verifying the ownership of the suspicious model using previously generated watermarked data. We designed a watermarking mechanism for two GNN applications: the graph classification task and the node classification task, and provided statistical analysis for the model ownership verification results. We conducted a comprehensive evaluation of our watermarking framework on different datasets and models and demonstrated that our method could achieve powerful watermarking performance while having a negligible effect on the host model's original task. We also explored our watermarking mechanism against an adaptive attacker who has knowledge of the watermarked data. We further show that our method is robust against a model extraction attack and four state-of-the-art defenses for backdoor attacks: randomized subsampling, fine-tuning, model pruning, and fine-pruning. For future work, we are interested in exploring methods that are more robust, e.g., against randomized smoothing for the node classification task and studying embedding watermarks into various types of GNNs besides node and graph-level tasks.

6

Discussion

Graph Neural Networks are powerful in processing graph data due to their superior ability to incorporate information from neighboring nodes in the graph recursively. However, GNNs are vulnerable to backdoor attacks, which can be implemented by training GNNs with poisoned training datasets. There are studies on backdoor attacks on Convolutional Neural Networks, but the unique characteristics of graph data make it challenging to explore backdoor attacks on GNNs. The results and findings in this thesis pave the way toward more powerful and efficient backdoor attacks on GNNs, including centralized and federated settings.

In this chapter, we present our findings and discuss their limitations and possible future work. In Section 6.1, 6.2 and 6.3, we will address the three subquestions introduced in Section 1.4. Limitations and future work are discussed in Section 6.4 and 6.5, respectively.

6.1. Backdoor attacks on centralized GNNs

In this section, we explain our contributions to the first subquestion:

RQ: *Can we design backdoor attacks on centralized GNNs?*

To answer this question, in Chapter 3, we explore and design backdoor attacks on centralized GNNs. Specifically, we give solutions to the following research questions, i.e., *What is the impact of trigger injection position on graph backdoor? How can we explain it?* and *Can we design the clean-label backdoor attack on GNNs?* The motivation for this question comes from the necessity of understanding the influence of trigger injecting position on the backdoor attack performance on GNNs. Compared to the Euclidean data, e.g., images, there is no location information in a graph, which means the attacker cannot inject a trigger into a specific spatial location. Although there is no location information in a graph, we can get insight into which part of the graph contributes more to the final prediction of the GNN model with the help of explanation techniques. Following this idea, in this chapter, we apply two powerful GNN explainability approaches to select the most or least important area of the sample as the trigger-injecting position for both graph and node classification tasks. The experimental results show that the attacker can select the least important parts of the graph to inject the trigger, thus reducing the chances of easy detection by the defender.

Through experiments, we find that there are indeed some differences between the two trigger-injecting strategies, i.e., LIAS (Least Important Area Strategy) always achieves higher attack performance than MIAS (Most Important Area Strategy). To further explain the different attack performances of these two trigger-injecting strategies, we also design a novel explanation framework with quantitative analysis. We compute the similarity of the predicted mask of the representative features from the backdoored model and the target mask of the representative features from the clean model.

From the experimental results, first, we can observe that most poisoned testing samples have a recall score of more than 0.5 in both MIAS and LIAS, which results in a high attack success rate for both strategies. Second, we notice that LIAS has fewer nodes with low recall scores than MIAS, which we believe is the reason for the higher ASR (Attack Success Rate) of LIAS than MIAS.

Dirty-label backdoor attacks are commonly used on GNNs, i.e., the adversary is assumed to have the capability to introduce arbitrary, often clearly mislabeled, inputs to the training set. However, these poisoned inputs are very likely to be detected as outliers. To make the resulting poisoned inputs appear consistent with their original labels so that it is more difficult to detect the poisoned inputs, we design a clean-label backdoor attack on GNNs. We sample subsets of the original training dataset, which is from the target class, and we inject a trigger (a graph) into the selected samples by sampling a specific number of nodes in the graph uniformly at random and replacing their connection with that in the trigger graph.

We observe that overall, a clean-label backdoor attack on GNNs can achieve high attack effectiveness, i.e., with an attack success rate over 84%. It can also be observed that generally, a clean-label backdoor attack on GNNs has a low clean accuracy drop, i.e., around 1%, which indicates that a clean-label backdoor attack on GNNs has a negligible impact on the original task of the model.

6.2. BACKDOOR ATTACKS ON FEDERATED GNNS

In this section, we discuss the second subquestion of this thesis:

RQ: *Can we design backdoor attacks on federated GNNs?*

To answer this question, in Chapter 4, we aim to answer the following research questions, i.e., *Can we design label-only membership inference attack on GNNs?* and *Is it possible to design backdoor attacks on Federated GNNs?* GNNs can face MIAs (Membership Inference Attacks) when centrally trained, which aims to classify whether the samples are used to train the target model. If GNNs only provide the prediction label for the input instead of the model's probability output, previous MIAs are not feasible anymore. A natural solution is to train a shadow model with a shadow dataset to mimic the behavior of the target model, considering that the shadow model and membership situation of nodes in the shadow dataset are transparent to the adversary. Following this idea, in this chapter, we propose a label-only MIA against GNNs for node classification with the help of GNNs' flexible prediction mechanism. The average attack accuracy, precision, and AUC values of our label-only MIA are competitive or even better than previous probability-based MIAs under the same settings.

In addition to the aforementioned MIAs, there are also other concerns on the centralized GNNs, e.g., privacy concerns, regulatory restrictions, and commercial competition. Considering these concerns, Federated GNNs are proposed to train GNNs over isolated graph data. Given that federated learning is designed to work with thousands or even millions of users without restrictions on eligibility, there are two ways to implement backdoor attacks on Federated GNNs: 1) We assume there are multiple (i.e., at least two) malicious clients and each of them has its local trigger. Each malicious client injects its local trigger into its training dataset. All malicious clients have the same backdoor task; 2) We assume there is only one malicious client in FL, and a global trigger consisting of local triggers is injected into this malicious client's local training dataset. Following this idea, in this chapter, we also design two backdoor attacks on Federated GNNs, i.e., Centralized Backdoor Attack and Distributed Backdoor Attack.

Through extensive experiments, we showed that, generally, DBA achieves a higher attack success rate than CBA. We showed that in CBA, the ASR of local triggers could be as high as the global trigger even if, during training, only the global trigger is embedded in the model. The impact of the percentage of malicious clients on DBA's ASR is analyzed with correlation, where we confirm the intuition that more malicious clients lead to more successful attacks. We also observe that DBA and CBA are robust against two defenses for the backdoor attacks in FL.

6.3. PROTECTING OWNERSHIP OF GNNS

In this section, we discuss the third subquestion:

RQ: *Is it possible to design backdoor attacks on GNNs*
for defensive purposes, e.g., watermarking GNNs?

To answer this question, in Chapter 5, we propose a watermarking method to protect the ownership of GNNs. Indeed, GNNs are very powerful in processing graph data, but building a powerful GNN model is not a trivial task. Also, with the development of model stealing attacks, there is a significant risk of trained GNN models being stolen by adversaries, potentially resulting in substantial commercial losses. Thus, from the perspective of a GNN model owner, it's necessary to verify the ownership of his/her model. Embedding watermarks into DNN models to protect the IP of the models has been conducted in multiple works, one of which is based on backdoor attacks. Backdoor attacks possess the secrecy characteristic, which makes it intuitive to consider watermarking GNNs by training the model with the backdoor functionality and verifying the ownership of suspicious GNN models with pre-determined backdoored data. Following this idea, in this chapter, we present a watermarking framework for GNNs for both graph and node classification tasks. We 1) design two strategies to generate watermarked data for the graph classification task and one for the node classification task, 2) embed the watermark into the host model through training to obtain the watermarked GNN model, and 3) verify the ownership of the suspicious model in a black-box setting along with statistical analysis.

The experiments show that our framework can verify the ownership of GNN models with a very high probability (up to 99%) for both tasks while having a negligible effect on the host model's original task. We also explored our watermarking mechanism against an adaptive attacker who has knowledge of the watermarked data. We further show that our method is robust against a model extraction attack and four state-of-the-art defenses for backdoor attacks: randomized subsampling, fine-tuning, model pruning, and fine-pruning.

6.4. LIMITATIONS

Although this thesis explores and designs effective and powerful backdoor attacks on centralized and federated GNNs, several limitations still need to be addressed.

Limited subset of practical datasets. The experiments conducted in this thesis primarily utilize publicly available datasets that are widely used in the literature for comparing GNNs. It is crucial to recognize that they do not represent "standard" benchmarks and encompass only a limited subset of practical datasets. Consequently, the proposed approaches may not be comprehensively evaluated based on these datasets. For instance, regarding the node classification task in this thesis, the utilization of small graph datasets such as Cora and CiteSeer is common. However, to further assess the effectiveness of the proposed approaches, experiments on datasets with different distributions or sizes (e.g., larger graphs) are important. Additionally, the dataset splitting methodology employed in this thesis involves a single dataset split for model evaluation and selection. Nonetheless, research indicates that the performance of GNN models can be significantly influenced by the specific choice of dataset splitting [113, 40]. Therefore, it is recommended in future work to select and evaluate GNNs on multiple data splits to ensure a fair comparison and robust assessment of their performance.

Attack scenarios from a research perspective. The attack scenarios of this thesis are mostly from a research but not practical perspective. For example, the experimental setting in the work of exploring backdoor attacks on federated GNNs is in a cross-silo federated learning setting which assumes a maximum of 100 clients. However, in the real-world applications of FL, cross-device FL, which assumes more than 100 clients, is considered.

No investigation on other advanced backdoor attacks. In this thesis, we focus on the classical dirty-label and clean-label backdoor attacks on GNNs. There are also many other advanced backdoor attacks that were proposed in regular data, e.g., SSBA [76] and WaNet [94]. More exploration and evaluation of advanced backdoor attacks on GNNs are expected.

Inadequate explanation for experimental results. In all experimental parts of this thesis, we try to analyze and explain the experimental results from different perspectives, such as dataset characteristics, model capability, and threat models. Nonetheless, some of our experimental results are only provided with hypotheses that are not finally verified. For instance, in the analysis of Chapter 4 we make a hypothesis for the ineffectiveness of the RLR defense against our attacks. While this hypothesis can explain the poor performance of RLR against our backdoor attacks on Federated GNNs, it would be advisable to investigate this hypothesis with extended experiments.

6.5. Future Work

In this thesis, we offer solutions to develop effective backdoor attacks on centralized and federated GNNs. However, considering the complexity of graph data and the difference between graph and other regular data (i.e., image), the backdoor attack scenarios in this thesis are limited. There are still many interesting new directions. We here discuss some open problems in the backdoor attacks on GNNs that could help future research on this topic.

Multi-target backdoor attacks on GNNs This thesis focuses on exploring backdoor attacks that have a single backdoor target and are triggered by a single backdoor. However, on the centralized GNNs, the backdoor attacks against multiple target classes, and backdoor attacks triggered by multiple backdoors have not been studied yet. If we refer the backdoor attacks on centralized GNNs in our thesis to *One-to-One attack*, how to design the *One-to-N attack* where the attacker can trigger multiple backdoor targets by controlling the different properties of the same backdoor, and *N-to-One attack* where there are multiple triggers and such attack is triggered only when all the triggers are satisfied?

Backdoor attacks on Federated GNNs for other tasks Federated GNNs are also popular in node-level tasks. For example, in a social media app where each user has a local social network G^k, and $\{G^k\}$ constitutes the latent entire human social network G, the developers can train a fraud detection GNN model through FL. In such a case, an attacker may conduct a backdoor attack to force the trained global model to misclassify a fraud node into benign. In Chapter 4, we evaluated the backdoor attack on Federated GNNs only for graph classification tasks. Is it also possible to design backdoor attacks on Federated GNNs for other tasks, e.g., node classification task?

Defenses against backdoor attack on Federated GNNs In Section 4.3.4, we evaluated the robustness of our backdoor attacks on Federated GNNs against two defenses. Based on experimental results, we find that the defense based on cosine similarity between updates is not effective in the graph domain. One reason may be that this defense applies cosine distance to try to identify malicious models, i.e., the distance between malicious updates is smaller between honest updates. Still, in our attacks, the malicious clients' updates could already be very dissimilar to each other, so the malicious updates are likely to be clustered into honest updates. Is it possible to design a novel defense against backdoor attacks on Federated GNNs combining the specific characteristics of graph data?

Ownership verification for GNNs based on dataset inference In our thesis, we used the watermarking technique to protect the ownership of the GNN models. The aim of watermarking GNNs is to detect theft by allowing the victim to claim ownership by verifying that a suspicious model responds with the expected outputs on watermarked inputs. This strategy requires re-training, and it can be vulnerable to adaptive attackers that try to remove the watermarks. Dataset inference is a process of identifying whether a suspicious model has private knowledge from the original model's dataset as a defense against model stealing. It is based on the observation that knowledge contained in the stolen model's training set is what is common to all stolen copies. We believe it would be a promising direction to use dataset inference to conduct ownership resolution on GNNs.

Bibliography

[1] Ahmed Abusnaina, Aminollah Khormali, Hisham Alasmary, Jeman Park, Afsah Anwar, and Aziz Mohaisen. Adversarial learning attacks on graph-based iot malware detection systems. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*, pages 1296–1305. IEEE, 2019.

[2] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, 2018.

[3] Marco Ancona, Cengiz Oztireli, and Markus Gross. Explaining deep neural networks with a polynomial time algorithm for shapley value approximation. In *International Conference on Machine Learning*, pages 272–281. PMLR, 2019.

[4] Valerio Arnaboldi, Marco Conti, Massimiliano La Gala, Andrea Passarella, and Fabio Pezzoni. Ego network structure in online social networks and its impact on information diffusion. *Computer Communications*, 76:26–41, 2016.

[5] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[6] Pierre Baldi and Gianluca Pollastri. The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. *The Journal of Machine Learning Research*, 4:575–602, 2003.

[7] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[8] Luciano Baresi and Reiko Heckel. Tutorial introduction to graph transformation: A software engineering perspective. In *International Conference on Graph Transformation*, pages 402–429. Springer, 2002.

[9] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 515–532, 2019.

[10] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019.

[11] Monica Bianchini, Anas Belahcen, and Franco Scarselli. A comparative study of inductive and transductive learning with feedforward neural networks. In *AI* IA 2016 Advances in Artificial Intelligence: XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29–December 1, 2016, Proceedings XV*, pages 283–293. Springer, 2016.

[12] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30, 2017.

[13] Franziska Boenisch. A systematic review on model watermarking for neural networks. *Frontiers in big Data*, 4:729663, 2021.

[14] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*.

[15] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388, 2019.

[16] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[17] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[18] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2020.

[19] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914. IEEE, 2022.

[20] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[21] Chaochao Chen, Jun Zhou, Longfei Zheng, Huiwen Wu, Lingjuan Lyu, Jia Wu, Bingzhe Wu, Ziqi Liu, Li Wang, and Xiaolin Zheng. Vertically federated graph neural network for privacy-preserving node classification. *arXiv preprint arXiv:2005.11903*, 2020.

[22] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pages 105–113, 2019.

[23] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph unlearning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 499–513, 2022.

[24] Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks. In *International conference on machine learning*, pages 1964–1974. PMLR, 2021.

[25] Giorgio Ciano, Alberto Rossi, Monica Bianchini, and Franco Scarselli. On inductive–transductive learning with graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(2):758–769, 2021.

[26] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR, 2019.

[27] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 77–ff, 2003.

[28] Tianshuo Cong, Xinlei He, and Yang Zhang. Sslguard: A watermarking scheme for self-supervised learning pre-trained encoders. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 579–593, 2022.

[29] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

[30] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. *arXiv preprint arXiv:2204.08570*, 2022.

[31] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[32] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. McGraw-Hill Higher Education New York, 2008.

[33] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic

and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2), 1991.

[34] Xiang Deng and Zhongfei Zhang. Graph-free knowledge distillation for graph neural networks. *arXiv preprint arXiv:2105.07519*, 2021.

[35] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *ACM SIGKDD Explorations Newsletter*, 24(2):61–77, 2022.

[36] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. Februus: Input purification defense against trojan attacks on deep neural network systems. In *Annual computer security applications conference*, pages 897–912, 2020.

[37] Ye Dong, Xiaojun Chen, Kaiyun Li, Dakui Wang, and Shuai Zeng. Flod: Oblivious defender for private byzantine-robust federated learning with dishonest-majority. In *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I*, pages 497–518. Springer, 2021.

[38] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. Quantifying privacy leakage in graph embedding. In *MobiQuitous 2020-17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 76–85, 2020.

[39] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[40] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[41] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

[42] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security*, 2020.

[43] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[44] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[45] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[46] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[47] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[48] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[49] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[50] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[51] Sensen Guo, Xiaoyu Li, and Zhiying Mu. Adversarial machine learning on social network: A survey. *Frontiers in Physics*, 9:766540, 2021.

[52] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

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

[54] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[55] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Carl Yang, Han Xie, Lichao Sun, Lifang He, Liangwei Yang, Philip S Yu, Yu Rong, et al. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145*, 2021.

[56] Chaoyang He, Emir Ceyani, Keshav Balasubramanian, Murali Annavaram, and Salman Avestimehr. Spreadgnn: Serverless multi-task federated learning for graph neural networks. *arXiv preprint arXiv:2106.02743*, 2021.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[58] Xinlei He, Rui Wen, Yixin Wu, Michael Backes, Yun Shen, and Yang Zhang. Node-level membership inference attacks against graph neural networks. *arXiv preprint arXiv:2102.05429*, 2021.

[59] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[60] Yifan Hu and Lei Shi. Visualizing large graphs. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(2):115–136, 2015.

[61] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[62] Bo Hui, Yuchen Yang, Haolin Yuan, Philippe Burlina, Neil Zhenqiang Gong, and Yinzhi Cao. Practical blind membership inference attack via differential comparisons. *arXiv preprint arXiv:2101.01341*, 2021.

[63] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1937–1954, 2021.

[64] Meng Jiang, Taeho Jung, Ryan Karl, and Tong Zhao. Federated dynamic gnn with secure aggregation. *arXiv preprint arXiv:2009.07351*, 2020.

[65] Zhihua Jin, Yong Wang, Qianwen Wang, Yao Ming, Tengfei Ma, and Huamin Qu. Gnnlens: A visual analytics approach for prediction error diagnosis of graph neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[66] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[67] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[68] Boris Knyazev, Graham W Taylor, and Mohamed Amer. Understanding attention and generalization in graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[70] Aditya Kuppa and Nhien-An Le-Khac. Adversarial xai methods in cybersecurity. *IEEE transactions on information forensics and security*, 16:4924–4938, 2021.

[71] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[72] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. Peer-to-peer federated learning on graphs. *arXiv preprint arXiv:1901.11173*, 2019.

[73] Gerhard C Langelaar, Iwan Setyawan, and Reginald L Lagendijk. Watermarking digital image and video data. a state-of-the-art overview. *IEEE Signal processing magazine*, 17(5):20–46, 2000.

[74] Shaofeng Li, Minhui Xue, Benjamin Zi Hao Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2088–2105, 2020.

[75] Yue Li, Hongxia Wang, and Mauro Barni. A survey of deep neural network watermarking techniques. *Neurocomputing*, 461:171–193, 2021.

[76] Yuezun Li, Yiming Li, Baoyuan Wu, Longkang Li, Ran He, and Siwei Lyu. Invisible backdoor attack with sample-specific triggers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16463–16472, 2021.

[77] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR, 2019.

[78] Zheng Li and Yang Zhang. Membership leakage in label-only exposures. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 880–895, 2021.

[79] Lingyu Liang, Lianwen Jin, and Yong Xu. Adaptive gnn for image analysis and editing. *Advances in Neural Information Processing Systems*, 32, 2019.

[80] Xiang Ling, Lingfei Wu, Wei Deng, Zhenqing Qu, Jiangyu Zhang, Sheng Zhang, Tengfei Ma, Bin Wang, Chunming Wu, and Shouling Ji. Malgraph: Hierarchical graph neural networks for robust windows malware detection. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1998–2007. IEEE, 2022.

[81] Fenglin Liu, Xian Wu, Shen Ge, Wei Fan, and Yuexian Zou. Federated learning for vision-and-language grounding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11572–11579, 2020.

[82] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International symposium on research in attacks, intrusions, and defenses*, pages 273–294. Springer, 2018.

[83] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13172–13179, 2020.

[84] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning. *arXiv preprint arXiv:2211.12814*, 2022.

[85] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1265–1282, 2019.

[86] Yunhui Long, Lei Wang, Diyue Bu, Vincent Bindschaedler, Xiaofeng Wang, Haixu Tang, Carl A Gunter, and Kai Chen. A pragmatic approach to membership inferences on machine learning models. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 521–534. IEEE, 2020.

[87] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. Sok: How robust is image classification deep neural network watermarking? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 787–804. IEEE, 2022.

[88] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[89] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020.

[90] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[91] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[92] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[93] Azqa Nadeem, Daniël Vos, Clinton Cao, Luca Pajola, Simon Dieck, Robert Baumgartner, and Sicco Verwer. Sok: Explainable machine learning for computer security applications. *arXiv preprint arXiv:2208.10605*, 2022.

[94] Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. *arXiv preprint arXiv:2102.10369*, 2021.

[95] Thien Duc Nguyen, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Shaza Zeitouni, et al. {FLAME}: Taming backdoors in federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1415–1432, 2022.

[96] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning attacks on federated learning-based iot intrusion detection system. In *Proc. Workshop Decentralized IoT Syst. Secur.(DISS)*, 2020.

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

[98] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4954–4963, 2019.

[99] Mustafa Safa Ozdayi, Murat Kantarcioglu, and Yulia R Gel. Defending against backdoors in federated learning with robust learning rate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9268–9276, 2021.

[100] Ren Pang, Zheng Zhang, Xiangshan Gao, Zhaohan Xi, Shouling Ji, Peng Cheng, and Ting Wang. Trojanzoo: Everything you ever wanted to know about neural backdoors (but were afraid to ask). *arXiv preprint arXiv:2012.09302*, 2020.

[101] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[102] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 70:1142–1154, 2022.

[103] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4422–4431, 2018.

[104] Pavel Pudlák, Vojtěch Rödl, and Petr Savickỳ. Graph complexity. *Acta Informatica*, 25:515–535, 1988.

[105] Shadi Rahimian, Tribhuvanesh Orekondy, and Mario Fritz. Differential privacy defenses and sampling attacks for membership inference. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, pages 193–202, 2021.

[106] Alberto Rossi, Matteo Tiezzi, Giovanna Maria Dimitri, Monica Bianchini, Marco Maggini, and Franco Scarselli. Inductive–transductive learning with graph neural networks. In *Artificial Neural Networks in Pattern Recognition: 8th IAPR TC3 Workshop, ANNPR 2018, Siena, Italy, September 19–21, 2018, Proceedings 8*, pages 201–212. Springer, 2018.

[107] Ahmed Salem, Yang Zhang, Mathias Humbert, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Network and Distributed Systems Security Symposium 2019*. Internet Society, 2019.

[108] Franklin E Satterthwaite. An approximate distribution of estimates of variance components. *Biometrics bulletin*, 2(6):110–114, 1946.

[109] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[110] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[111] Giorgio Severi, Jim Meyer, Scott E Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *USENIX Security Symposium*, pages 1487–1504, 2021.

[112] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

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

[114] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1354–1371. IEEE, 2022.

[115] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[116] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[117] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[118] Liwei Song and Prateek Mittal. Systematic evaluation of privacy risks of machine learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2615–2632, 2021.

[119] Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A Alemi, and Andrew G Wilson. Does knowledge distillation really work? *Advances in Neural Information Processing Systems*, 34:6906–6919, 2021.

[120] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[121] Toyotaro Suzumura, Yi Zhou, Natahalie Baracaldo, Guangnan Ye, Keith Houck, Ryo Kawahara, Ali Anwar, Lucia Larise Stavarache, Yuji Watanabe, Pablo Loyola, et al. Towards federated graph learning for collaborative financial crimes detection. *arXiv preprint arXiv:1909.12946*, 2019.

[122] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[123] Jean-Baptiste Truong, Pratyush Maini, Robert J Walls, and Nicolas Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4771–4780, 2021.

[124] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks. 2018.

[125] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*, pages 269–277, 2017.

[126] Maria V Valueva, NN Nagornov, Pavel A Lyakhov, Georgii V Valuev, and Nikolay I Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and computers in simulation*, 177:232–243, 2020.

[127] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[128] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[129] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 968–977, 2019.

[130] Jianian Wang, Sheng Zhang, Yanghua Xiao, and Rui Song. A review on graph neural network methods in financial applications. *arXiv preprint arXiv:2111.15367*, 2021.

[131] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[132] Xiao Wang, Hongrui Liu, Chuan Shi, and Cheng Yang. Be confident! towards trustworthy graph neural networks via confidence calibration. *Advances in Neural Information Processing Systems*, 34:23768–23779, 2021.

[133] Ronald L Wasserstein and Nicole A Lazar. The asa statement on p-values: context, process, and purpose, 2016.

[134] Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’networks. *nature*, 393(6684):440–442, 1998.

[135] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

[136] Bernard L Welch. The generalization of ‘student’s’problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[137] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. Adapting membership inference attacks to gnn for graph classification: approaches and implications. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1421–1426. IEEE, 2021.

[138] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925*, 2021.

[139] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610*, 2019.

[140] Lingfei Wu, Yu Chen, Heng Ji, and Bang Liu. Deep learning on graphs for natural language processing. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2651–2653, 2021.

[141] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. Graph neural networks. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 27–37. Springer, 2022.

[142] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[143] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph backdoor. In *USENIX Security Symposium*, pages 1523–1540, 2021.

[144] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International conference on learning representations*, 2020.

[145] Jing Xu, Minhui Xue, and Stjepan Picek. Explainability-based backdoor attacks against graph neural networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*, pages 31–36, 2021.

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

[147] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018.

[148] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 103–120. IEEE, 2021.

[149] Ying Xu, Xu Zhong, Antonio Jose Jimeno Yepes, and Jey Han Lau. Forget me not: Reducing catastrophic forgetting for domain adaptation in reading comprehension. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[150] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[151] Peng Yang, Yingjie Lao, and Ping Li. Robust watermarking for deep neural networks via bi-level optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14841–14850, 2021.

[152] Wenkai Yang, Yankai Lin, Peng Li, Jie Zhou, and Xu Sun. Rethinking stealthiness of backdoor attack against nlp models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5543–5557, 2021.

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

[154] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR, 2018.

[155] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[156] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[157] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[158] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 430–438, 2020.

[159] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[160] He Zhang, Bang Wu, Xingliang Yuan, Shirui Pan, Hanghang Tong, and Jian Pei. Trustworthy graph neural networks: Aspects, methods and trends. *arXiv preprint arXiv:2205.07424*, 2022.

[161] Huanding Zhang, Tao Shen, Fei Wu, Mingyang Yin, Hongxia Yang, and Chao Wu. Federated graph learning–a position paper. *arXiv preprint arXiv:2105.11099*, 2021.

[162] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pages 159–172, 2018.

[163] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[164] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 15–26, 2021.

[165] Zhengyan Zhang, Guangxuan Xiao, Yongwei Li, Tian Lv, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Xin Jiang, and Maosong Sun. Red alarm for pre-trained models: Universal vulnerability to neuron-level backdoor attacks. *Machine Intelligence Research*, 20(2):180–193, 2023.

[166] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14443–14452, 2020.

[167] Xiangyu Zhao, Hanzhou Wu, and Xinpeng Zhang. Watermarking graph neural networks by random graphs. In *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–6. IEEE, 2021.

[168] Xinghua Zhu, Jianzong Wang, Zhenhou Hong, and Jing Xiao. Empirical studies of institutional federated learning for natural language processing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 625–634, 2020.

[169] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.

Acknowledgements

I am not sure whether doing a PhD is one of the best decisions in my life but I am sure it undeniably becomes one of the most unforgettable experiences in my life. Over the past four years, I have encountered many moments — happiness, sadness, struggle, loneliness, and excitement. At the end of my PhD journey, I would like to express my gratitude to the people who have enriched these years.

First and foremost, I would like to thank my daily supervisor Dr. Stjepan Picek, my promotors Prof.dr.ir Inald Lagendijk, and Dr. Frans Oliehoek. Stjepan, I could not have undertaken this journey without your help, guidance and support. I was not confident in my research and ability at the beginning. Thank you for your unwavering encouragement and assistance which have strongly supported my pursuit of PhD studies. Many thanks for your weekly online meetings with me during the pandemic of COVID-19, which gave me great support to get through that tough time. In addition to supporting my research study, you also behaved like my mentor who offered me encouragement so that I could have the courage to face the difficulties I encountered. Inald, thank you for always giving me professional and detailed advice. Your passion for research is truly admirable, and the discussions during our meetings were quite helpful for my exploration of machine learning and security. Thank you for your control of the overall direction of this PhD research, which allowed me to finish this meaningful study. I am also extremely grateful to Frans. This wonderful journey would not be possible without your crucial role in initiating my PhD study at TU Delft. Thank you for the constructive and insightful comments you have always given me. In addition to the research insight and ability, I admire and hope to learn from your extensive knowledge and attitude of always keep learning.

I also extend my gratitude to the committee members Prof.dr. Georgios Smaragdakis, Prof.dr. Ahmad-Reza Sadeghi, Prof.dr. Lorenzo Cavallaro, and Dr. Elvin Isufi for their commitment to reviewing my thesis, providing valuable feedback, and being part of the defense ceremony.

Throughout my PhD journey, I have been lucky to have met many wonderful colleagues. I would like to extend my sincere thanks to the lovely people currently or formerly in the Cybersecurity group. Geroge, I will miss the jokes you made. I will always cherish the memories of your warm hugs and smiles. Thank you for being our "Big Geroge". Zeki, I know you love playing jokes, and I enjoy our conversations by the coffee machine. Mauro, thank you for supporting my research visit to Padova. Collaborating with you has been rewarding and enriching. Sicco, thanks for your every time's 早安 (good morning in Chinese). Kaitai, thanks for your advice on whether to join the industry or academia after my PhD graduation. Sandra, your assistance and support throughout my stay in the Cybersecurity group have been invaluable. I am grateful for the vibrant atmosphere you created and the various group events you organized, which fostered a sense of belonging.

Stefanos, working alongside you in the same office has been an absolute delight. You are so friendly, funny, and always ready to help everyone. It is enjoyable to work with you, especially when we complain together about the review comments, repeated experiments and your every "last minute" story. Marina, I can still remember your kindness and support during a particularly challenging moment in the first year of my PhD. Thanks so much for your warm comfort. I hope you continue to find joy in both your work and life. I would also like to express my gratitude to Tianyu Li, Huanhuan Chen, Rui Wang, Yanqi Qiao, Dazhuang Liu, and Zeshun Shi. Thanks for all the memorable moments we shared playing board games, enjoying BBQs, and celebrating at parties. Ozzy, collaborating with you was truly enjoyable. I admire your knowledge of mathematics. Daniël, thanks for your assistance with the Dutch summary of this thesis. I can (hopefully) recognize you and Jelle now. Clinton, I have witnessed your remarkable progress in speaking Chinese. I look forward to our next conversation, in Chinese. Florine, you are one of the bravest people I met. May you continue living as you like. Stefano, thanks for all the joyful talks we had during your stay at TU Delft. I miss your infectious laughter and hope you can smile like a child always. I also want to thank Ruud and Bart, who offered me technical support throughout my entire PhD journey. Although, unfortunately, I experienced computer breakdowns several times, I was able to continue working with timely help from Ruud. Thank you very much.

Also, I want to thank my colleagues in AISyLab at Radboud University. Azade, thanks for inviting me to the relaxing citywalk in Delft. Gorka, Behrad, the time we tried to catch a deadline together in Belgium brought me a "joyful" memory. I would also like to express my appreciation to my PhD fellows, including Leo Weissbart, Luca Mariot, Guilherme Perin, Xiaoyun Xu, and Lichao Wu.

Outside of work, I am fortunate to have amazing friends who provided invaluable support during my PhD journey. Fenghua, there are too many unforgettable memories we have created together throughout this journey. Thank you for doing silly things with me, making tasty birthday cakes for me, and comforting me when I was down. Chen, thank you for being my "elder brother", although my birthday is actually earlier than yours. Biyue, I remember the gathering at your home every Friday evening during the second and third years of my PhD, which brought me great joy. Yawei, thanks for introducing your friends to me so that this long journey was much more enjoyable. It was quite enjoyable staying with you all, no matter what we did. So many times, we shared fun stuff, had Chinese dishes together, and played switch games, so on and so forth, like a family. I cannot imagine finishing my PhD without you.

Huimin, you are not only my colleague in the same office but also one of my closest friends at TU Delft. Your never-giving-up attitude gives me valuable mental support, especially during challenging times like the lockdown period. Shenglan, our conversations over drinks about random topics like dating shows have brought me great enjoyment. Sitong, thank you for the warm hospitality and delicious dishes you prepared during the many invitations to your home. They were truly the best meals I've had in the Netherlands. Fengqiao, thank you for inviting me to the delicious hotpots at your home. Your and Lichao's hospitality at the beginning of my PhD study was heartwarming and unforgettable. Yun Zeng, Zhangyue Wei, I really like playing board games with you, even if I was often the last one. Dingding, thanks for your valuable suggestions in the first year of my PhD. I also would like to extend my warm thanks to Li Zou, Bo Sun, Shilun Zhang, Huiyuan Lai, Jiaxin Li, Junhan Wen, Wei Yuan, Qisong Yang, Li Wang, Langzi Chang, Longjian Piao, Yunlong Guo and Lili Ma.

My most special thanks go to my boyfriend, Jinke He. During the last four years, you have been my unlimited source of support. Every time I was happy, sad, stressed, or confused, you were always the first one I wanted to share my feelings with, and you were always ready to give me unconditional support. Your wisdom and broad perspective have often guided me when I felt stuck. Thank you for tolerating my impatience, supporting every decision I made, and telling me that it's perfect to be imperfect. Our paths crossed in 2019 when we both just started our PhD study, and since then, you have become an important part of my PhD journey.

Words cannot express my gratitude to my family. I am grateful to have a loving family. Especially, I would like to express my sincere appreciation to my parents. Thank you for offering me unconditional love and support, which have empowered me to pursue my dreams fearlessly. I also want to express my heartfelt appreciation to my grandparents. The cherished memories of my childhood spent with you remain etched in my heart, reminding me of the warmth and love that surrounded me all the time.

Over the past four years, I continued asking myself what is the meaning of pursuing a PhD. Now my answer is *to find out how many possibilities you have*. These four years will certainly be one of the most valuable assets in my life. Thank you all for being with me on this wonderful journey!

CURRICULUM VITÆ

Jing XU

Jing Xu was born in Zhejiang, China on April 8, 1994. She obtained her bachelor's degree in Engineering with distinction from Shanghai University, Shanghai, China in 2016. After that, she received her master's degree in Optical Engineering from Beihang University, Beijing, China in 2019.

In 2019, Jing Xu started her PhD research at the Department of Intelligent Systems, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, with a grant from the China Scholarship Council. In the first year of her PhD, Jing focused on research about adversary attacks on reinforcement learning under the supervision of dr. F.A. Oliehoek. During that time, she found her special interest in the security of machine learning, and then she joined the Cybersecurity group and was additionally supervised by prof. dr. ir. R.L. Lagendijk and dr. S. Picek. In the next three years of her PhD, Jing focused on the security of machine learning, especially backdoor attacks on graph neural networks. She designed various backdoor attacks on centralized and federated GNNs. During her PhD, she has published and presented her works at multiple international conferences.

List of Publications

Published

10. **Xu, J.**, Koffas, S., & Picek, S., *Unveiling the Threat: Investigating Distributed and Centralized Backdoor Attacks in Federated Graph Neural Networks*, Digital Threats: Research and Practice (DTRAP 2023).
9. **Xu, J.**, Picek, S., *Poster: Multi-target & Multi-trigger Backdoor Attacks on Graph Neural Networks*, ACM Conference on Computer and Communications Security (CCS 2023).
8. **Xu, J.**, Koffas, S., Ersoy, O., & Picek, S., *Watermarking Graph Neural Networks based on Backdoor Attacks*, IEEE European Symposium on Security and Privacy (Euro S&P 2023).
7. **Xu, J.**, Abad, G., & Picek, S., *Rethinking the Trigger-injecting Position in Graph Backdoor Attack*, International Joint Conference on Neural Networks (IJCNN 2023).
6. **Xu, J.**, Koffas, S., & Picek, S., *Exploring Backdoor Attacks in Federated Graph Neural Networks*, The Learning from Authoritative Security Experiment Results (LASER workshop 2022).
5. **Xu, J.**, Wang, R., Koffas, S., Liang, K., & Picek, S, *More is Better (Mostly): On the Backdoor Attacks in Federated Graph Neural Networks*, Annual Computer Security Applications Conference (ACSAC 2022).
4. **Xu, J.**, Picek, S., *Poster: Clean-label Backdoor Attack on Graph Neural Networks*, ACM Conference on Computer and Communications Security (CCS 2022).
3. Conti, M., Li, J., Picek, S., **Xu, J.**, *Label-Only Membership Inference Attack against Node-Level Graph Neural Networks*, 15th ACM Workshop on Artificial Intelligence and Security (AISec 2022).
2. Koffas, S., **Xu, J.**, Conti, M., & Picek, S., *Can you hear it? backdoor attacks via ultrasonic triggers*, ACM Workshop on Wireless Security and Machine Learning (WiseML 2022).
1. **Xu, J.**, Xue, M., & Picek, S., *Explainability-based Backdoor Attacks Against Graph Neural Networks*, ACM Workshop on Wireless Security and Machine Learning (WiseML 2021).

Under review

2. Abad, G., **Xu, J.**, Koffas, S., Tajalli, B., & Picek, S., *A Systematic Evaluation of Backdoor Trigger Characteristics in Image Classification*, arXiv preprint (2023).
1. Arazzi, M., Conti, M., Koffas, S., Krcek, M., Nocera, A., Picek, S., **Xu, J.**, *Label Inference Attacks against Node-level Vertical Federated GNNs*, arXiv preprint (2023).