Delft University of Technology

Watermarking Graph Neural Networks based on Backdoor Attacks

Xu, Jing; Koffas, Stefanos; Ersoy, Oğuzhan; Picek, Stjepan

Watermarking Graph Neural Networks based on Backdoor Attacks

Jing Xu
Delft University of Technology
Delft, Netherlands
j.xu-8@tudelft.nl

Stefanos Koffas
Delft University of Technology
Delft, Netherlands
s.koffas@tudelft.nl

Oğuzhan Ersoy
Radboud University
Nijmegen, Netherlands
oguzhan.ersoy@ru.nl

Stjepan Picek
Radboud University
Nijmegen, Netherlands
stjepan.picek@ru.nl

Abstract—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 paper 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.

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 [20], [25], [42], speech recognition [18], [21], and natural language processing [16], deep graph models such as graph neural networks (GNNs) [19], [23], [48] 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 [54]. 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 [44].

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 [35], [37], [45], 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 [26]. Multiple works discuss embedding watermarks into DNN models to protect the IP of the models [1], [12], [22], [32], [47], [61], [64]. For instance, Uchida et al. [47] presented a framework to embed watermarks into the parameters of DNNs via the parameter regularizer during training leading to its white-box setting. 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 [1]. The authors based their approach on backdoor attacks. Additionally, Zhang et al. [64] 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 [61], and Jia et al. presented an approach to force the model to entangle representations for legitimate task data and watermarks [22]. A robust black-box watermarking scheme for self-supervised learning pre-trained encoders was proposed by Cong et al. [12]. 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 [56], [68]. 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 [68]. 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 [68] 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 [10]. 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 [38]. In this paper, 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 [68]. 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 [68], 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 [68], 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 [68], i.e., up to 100%.

Following the idea of [1], 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 [52], [55], [66] 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 [68], 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).

2. Background

2.1. Graph Neural Networks (GNNs)

GNNs take a graph G as an input (including its structure information and node features) and learn a representation vector (embedding) for each node $v \in G$, z_v, or the entire graph, z_G. Modern GNNs follow a neighborhood aggregation strategy, where one iteratively updates the representation of a node by aggregating 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 (e.g., GCN [23], GraphSAGE [19], and GAT [48]): $Z^{(k)} = AGGREGATE(A, Z^{(k-1)}; \theta^{(k)})$. Here, $Z^{(k)}$ represents the node embeddings in the matrix form computed after the k-th iteration, and the $AGGREGATE$ function depends on the adjacency matrix A, the trainable parameters $\theta^{(k)}$, and the previous node embeddings $Z^{(k-1)}$. Finally, $Z^{(0)}$ is initialized as G's node features.

For the node classification task, the node representation $Z^{(k)}$ of the final iteration is used for prediction, while for the graph classification task, the $READOUT$ function pools the node embeddings from the final iteration K: $z_G = READOUT(Z^{(K)})$. $READOUT$ can be a simple permutation invariant function or a more sophisticated graph-level pooling function [62], [65].

The goal of the node classification task is that, given a single graph with partial nodes being labeled and others remaining unlabeled, GNNs can learn a robust model that effectively identifies the class labels for the unlabeled nodes [23]. In a node classification task, there are two types of training settings - inductive and transductive. In an inductive setting, the unlabeled nodes are not seen during training, while in a transductive setting, the test nodes (but not their labels) are also observed during the training process. The node classification task is used in many security applications, e.g., anomaly detection in the Bitcoin transaction network or terrorist detection in a social network [6]. The graph classification task aims to predict the class label(s) for an entire graph [65]. One practical application of the graph classification task is to detect the mutagenic effect of chemical compounds [24]. Therefore, it is relevant to consider both classification tasks when considering real-world applications.

2.2. Backdoor Attacks in GNNs

Deep Neural Networks (DNNs) are vulnerable to backdoor attacks [27], [31]. A backdoored neural network produces attacker-desired behaviors when a trigger is injected into a test sample. Several studies showed that GNNs are also vulnerable to backdoor attacks. Similar to the idea of backdoor attack in DNNs, the backdoor attack in GNNs is implemented by poisoning the training data with a trigger, which can be a subgraph with/without features [55], [66] or a subset of node features [56]. After training the GNN model with the trigger-embedded data, the backdoored GNN would predict the test example injected with a trigger as the pre-defined target label.

2.3. Digital Watermarking in Neural Networks

Digital watermarking is a technique that embeds certain watermarks in carrier multimedia data such as audio, video, or images to protect their copyright [26]. 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. The typical digital watermarking life cycle is presented in Appendix A.

At first, the goal of digital watermarking was to protect the copyright of multimedia data by embedding water-
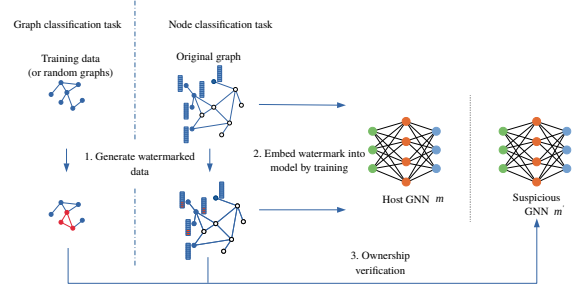

Figure 1: GNN watermarking framework.

marks into the multimedia data. 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 [1], [8], [47]. 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 [5], [28]: robustness, fidelity, capacity, integrity, generality, efficiency, and secrecy (see Section 4.2).

3. GNN Watermarking

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 1 illustrates the workflow of our GNN watermarking framework.

3.1. 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 similar 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., [3], [45]. Although the exact mechanism of how a suspect obtains the model m is out of the scope of this paper, we still evaluate our watermarking method against a model extraction attack in Section 5. 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 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.

3.2. 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 [55], [66] 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 [15] in which each possible edge occurs independently with a probability $p \in (0, 1)$. This paper 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 [51] and Preferential Attachment (PA) model [2]. 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 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 4. 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 [56] 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.

3.3. 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 1. The inputs are the pre-trained clean model m_c, 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 1) 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

---

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

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.

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 1). 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 1). 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 1: 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.

3.4. 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 3.1, 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 behavior 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 $\mathcal{H}_0$ as follows:

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

---

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.

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 $\mathcal{H}_0$ 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 $\mathcal{H}_1$ 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 $\mathcal{H}_0$ 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:

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

$$\beta_k = \frac{\sum_{i=1}^q \mathbb{I}(y_i^{c_k} = y_{wm})}{q}, k \in [1, n] \tag{2}$$

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 [53][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:

$$t = \frac{\bar{\alpha} - \bar{\beta}}{\sqrt{\frac{s_\alpha^2}{n} + \frac{s_\beta^2}{n}}}, \tag{3}$$

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 [39], [53]:

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

According to the theoretical analysis above, we can formally state under what conditions the model owner can reject the null hypothesis $\mathcal{H}_0$ 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 1. According to a Shapiro-Wilk Test [41], the p-values [50] 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

---

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 [41], and they may have different variances.

t-test is applicable. With significance level $1 - \tau = 0.05$, and the degree of freedom calculated with Eq. (4), i.e., $\nu \approx 16$, the t critical value t_τ is 2.120. Based on Eq. (3), we calculate t statistic $t = 45.82$, which is significantly larger than $t_\tau = 2.120$. Thus, we can reject the null hypothesis $\mathcal{H}_0$ 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 Appendix B. For each setting, we can reject the null hypothesis $\mathcal{H}_0$ 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 4.

If the watermark accuracy of clean and watermarked models are not normally distributed, we can apply the Mann-Whitney U test [33] to test the hypothesis. The U statistic is calculated as:

$$U_1 = R_1 - \frac{n_1(n_1 + 1)}{2}; \; U_2 = R_2 - \frac{n_2(n_2 + 1)}{2}, \quad (5)$$

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 1 where $n_1 = n_2 = 10$). Based on Eq. (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 $\mathcal{H}_0$ for a 0.05 significance level, which indicates that our watermarking method is also valid for the watermark accuracy of non-Gaussian distribution.

4. Evaluation

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 [34] - a subset of the dataset consisting of chemical compounds screened for activity against non-small cell lung cancer, (ii) COLLAB [60] - a scientific collaboration dataset, derived from three public collaboration datasets, and (iii) REDDIT-BINARY [60] - a dataset consisting of graphs corresponding to online discussions on Reddit. For the

node classification task, we use two real-world datasets: (i) Cora [40] and (ii) CiteSeer [40]. 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 2 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 4.1.

Models. We use three state-of-the-art GNN models for the graph classification task: DiffPool [62], GIN [57], and GraphSAGE [19]. For the node classification task, we use GCN [23], GAT [48], and GraphSAGE [19] as the host models. The hyperparameters for the neural networks (Table 22 in Appendix C)) are commonly used, see [55], [57], [62].

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 3.4, 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 3 and 4 for graph classification task and node classification task, respectively. In Table 3, D_{wm}^t is the watermarked data generated by embedding a watermark into sampled training data, while D_{wm}^r is the watermarked data generated by embedding a watermark into the generated random graphs. From Table 3, 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 3. Once the watermark accuracy threshold is defined, we send queries of generated watermarked data to the suspicious model m'. If the watermark accuracy of the suspicious model is over the corresponding threshold, we can reach

---

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

TABLE 1: 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 2: 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

TABLE 3: Watermark accuracy threshold for each dataset and model on the graph classification task.

Dataset	Watermark Acc. Threshold (%) ($D_{wm}^t \mid D_{wm}^r$)		
	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

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.

4.1. Experimental Results

Graph classification. As discussed in Section 3.2, 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 2. In Figure 2, 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.

TABLE 4: 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

From Figure 2a, 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 3, indicating that the model owner can use a very small watermarking rate, e.g., 0.01 to watermark the models. From Figure 2b, 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 2c, 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 2 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 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_{wm}^t 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_{wm}^r can mostly reach around 95% for all datasets.

Node classification. For the node classification task, the generated watermarked 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 3. From Figure 3a, for all datasets and models, the watermark accuracy has a dramatic rise

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

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

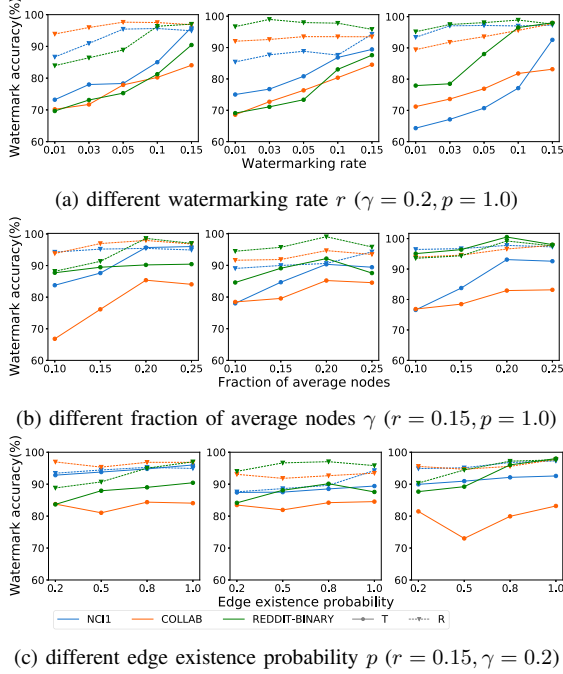(c) different edge existence probability p ($r = 0.15, \gamma = 0.2$)

Figure 2: Watermark accuracy on graph classification task (DiffPool (left), GIN (center), GraphSAGE (right), T: first watermarked data generation strategy, R: second watermarked data generation strategy).

TABLE 5: Watermark accuracy for graph classification task ($r = 0.15, \gamma = 0.2, p = 1.0$).

Dataset	Watermark Accuracy (%) ($D_{wm}^t \mid D_{wm}^r$)		
	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

(a) different watermarking rate r

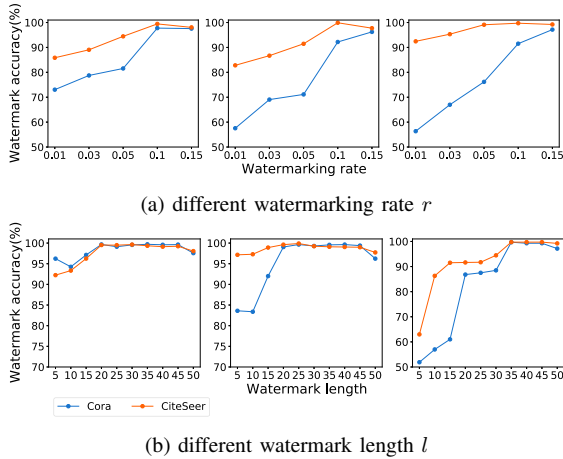(b) different watermark length l

Figure 3: Watermark accuracy on node classification task (GCN (left), GAT (center), GraphSAGE (right)).

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 3b, for

TABLE 6: 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

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 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 3b. 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 [4], 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 3 and 6 (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 6 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 [68], as shown in Figure 4. 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 [68]. Thus, our watermarking method can achieve similar or higher watermark accuracy than the state-of-the-art.
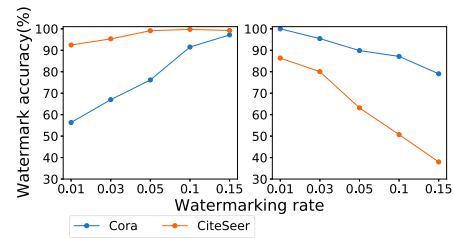

Figure 4: Comparison between our method (left) and [68] (right) on GraphSAGE model (inductive learning setting).

Impact on the original task. To measure the im-

pact 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 and 6 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, 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 6), 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 3b, 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 7 and 8 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.

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 techniques. 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 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

(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 ($r = 0.15, \gamma = 0.2$)
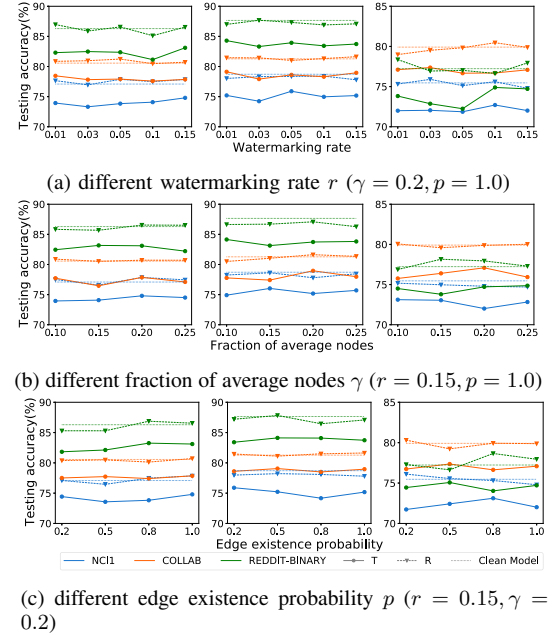
Figure 5: Testing accuracy on graph classification task (DiffPool (left), GIN (center), GraphSAGE (right), T: the first watermarked data generation strategy, R: the second watermarked data generation strategy).

(a) different watermarking rate r

(b) different watermark length l
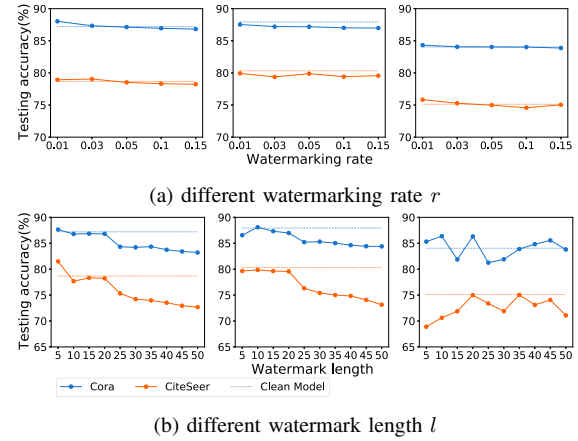
Figure 6: Testing accuracy on node classification task (GCN (left), GAT (center), GraphSAGE (right)).

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 7 and 8, respectively. For graph classification, gradually, with the increase of the stolen watermarked data rate, the watermark accuracy decreases significantly, i.e., reducing to

TABLE 7: 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

TABLE 8: Testing accuracy for the node classification.

Dataset	Testing Accuracy (%) $(CleanModel \vert D_{wm}^t)$		
	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

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 [59]. 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 [9]. In contrast, for node classification (Figure 8), 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.

4.2. On the Watermarking Requirements

Next, we explain the well-known watermarking requirements for neural networks [5] 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, our watermarking model achieves robustness against the modifications in the model.

(a) watermark accuracy

(b) testing accuracy
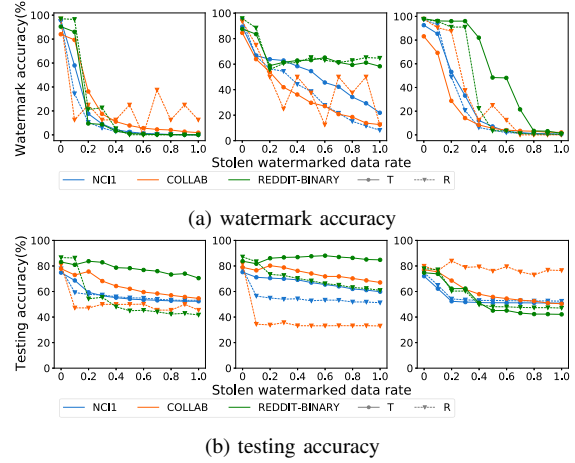
Figure 7: Watermarking performance for different stolen watermarked data rates on graph classification task (Diff-Pool (left), GIN (center), GraphSAGE (right)).

(a) watermark accuracy   (b) testing accuracy
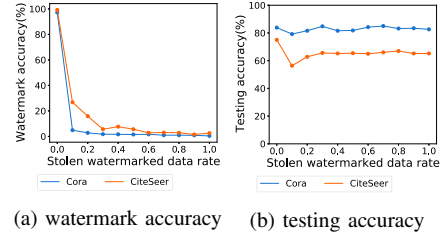
Figure 8: Watermarking performance for different stolen watermarked data rates on node classification (Graph-SAGE).

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 7 and 8 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 4.1.

Capacity: the watermark's capability to carry information, and two watermarking schemes, zero-bit, and multi-bit, are distinguished by it [5]. 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 3.4 and 4, 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 1 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 4.1, in our watermarking method, the model owners can use low watermarking rates, e.g., 0.01, to watermark their models.

In [5], 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. 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 [36]. NeuralCleanse (NC) [49] 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 samples to achieve good performance [30] while in our work, the plagiarizer has no access to the training data, and (2) NC cannot reverse engineer large triggers [14], [30], [58], 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) [7] 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) [11] (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 [13]. 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 [46]. 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 [17] since the host model is pretrained. 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 9 and 10 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 11. 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 [43], 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 robust against knowledge distillation.

TABLE 9: 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

Robustness against fine-tuning. As discussed before, training a well-performed GNN model from scratch

---

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

TABLE 10: 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 11: 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

TABLE 12: 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 13: 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 14: Watermarking performance of GraphSAGE model after fine-tuning on the node classification task.

Dataset	Watermark Accuracy (%)		Testing accuracy (%)	
	Ours	[68]	Ours	[68]
Cora	98.51	97.84	84.39	84.51
CiteSeer	99.45	86.36	75.10	75.41

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 [63]. 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 12 and 13 show the watermark accuracy and testing accuracy of the model after fine-tuning for the graph classification task. Comparing the results from Tables 5 and 12, 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 [67]. As for the testing accuracy, from Tables 7 and 13, 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 14, and the observations are similar to that for the graph classification task. To compare our watermarking mechanism with the state-of-the-art [68], we also present the watermark performance for the method in [68] after fine-tuning in Table 14. 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 [68] is less than 0.5% for both datasets.

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 [64]. We apply the pruning algorithm used in [47], 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 15 and 16 present the watermarking performance after model pruning on the graph and node classification tasks, respectively. We take the results for NCI1 and Cora as examples (the results of other datasets are shown in Appendix D.1). 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.

TABLE 15: Watermarking performance on graph classification task after model pruning (NCI1).

Pruning rate	DiffPool		GIN		GraphSAGE	
	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$
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 16: 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%

Robustness against randomized subsampling. Randomized smoothing [11] 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 [64]. Here, we explore the robustness of our watermarking method against randomized subsampling [55]. In particular, we apply a subsampling function 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 [55], in this paper, the randomized subsampling is only used to work on graphs instead of training smoothed models (thus, we use it to make robust samples). Figures 9 and 10 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.

(a) watermark accuracy

(b) testing accuracy

Figure 9: 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).

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 [29]. 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 Cora as examples, as shown

1191

TABLE 17: Watermarking performance on graph classification task after fine-pruning (NCI1).

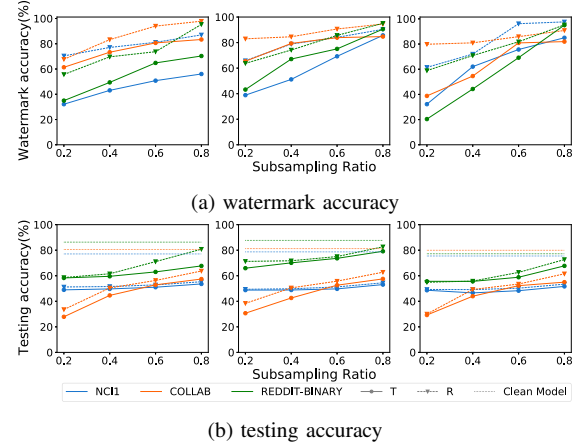| Pruning rate	DiffPool		GIN		GraphSAGE	
	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$
10%	81.32%|77.20%	95.69%|95.44%	82.32%|79.18%	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%

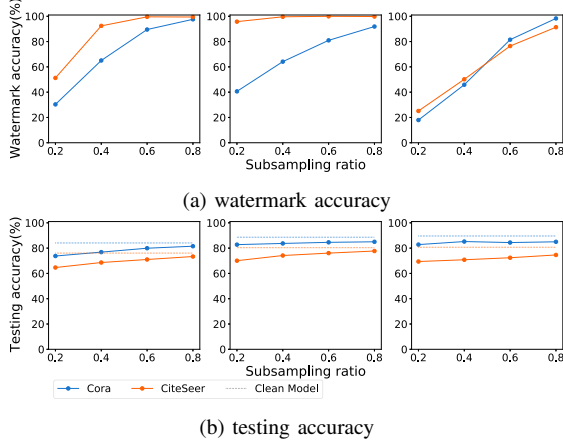(a) watermark accuracy

(b) testing accuracy

Figure 10: Watermarking performance for different subsampling ratios on node classification task (GCN (left), GCN (center), GraphSAGE (right)).

TABLE 18: 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%

in Tables 17 and 18, respectively. The results for other datasets are presented in Appendix D.2. 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 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 18 that when half of the parameters 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.

6. Conclusions and Future Work

This paper 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.

References

[1] 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.

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

[3] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 515–532. USENIX Association, 2019.

[4] Monica Bianchini, Anas Belahcen, and Franco Scarselli. A comparative study of inductive and transductive learning with feedforward neural networks. In *Conference of the Italian Association for Artificial Intelligence*, pages 283–293. Springer, 2016.

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

[6] Anshika Chaudhary, Himangi Mittal, and Anuja Arora. Anomaly detection using graph neural networks. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 346–350. IEEE, 2019.

[7] 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.

[8] 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.

[9] 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.

[10] 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*, 2021.

[11] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[12] 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.

[13] Xiang Deng and Zhongfei Zhang. Graph-free knowledge distillation for graph neural networks. In *The 30th International Joint Conference on Artificial Intelligence*, 2021.

[14] 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.

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

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

[17] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[18] 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.

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

[20] 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.

[21] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdelrahman 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.

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

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

[24] Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483*, 2012.

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

[26] 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.

[27] 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.

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

[29] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Finepruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018.

[30] 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.

[31] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Alberto Rossi, Matteo Tiezzi, Giovanna Maria Dimitri, Monica Bianchini, Marco Maggini, and Franco Scarselli. Inductive–transductive learning with graph neural networks. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 201–212. Springer, 2018.

[39] Franklin E Satterthwaite. An approximate distribution of estimates of variance components. *Biometrics bulletin*, 2(6):110–114, 1946.

[40] 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.

[41] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] 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.

[44] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[45] 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.

[46] 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.

[47] 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.

[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. accepted as poster.

[49] 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.

[50] Ronald L Wasserstein and Nicole A Lazar. The asa statement on p-values: context, process, and purpose, 2016.

[51] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393(6684):440–442, 1998.

[52] 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.

[53] Bernard L Welch. The generalization of 'student's' problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[54] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021.

[55] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph backdoor. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1523–1540, 2021.

[56] 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.

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

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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. *arXiv preprint arXiv:2101.06969*, 2021.

[68] 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.

A. Digital Watermarking Life Cycle

Figure 11 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.

B. Statistic Guarantee of Ownership Verification

The watermark accuracy of 10 watermarked models and clean models for other datasets and models are shown in Tables 19, 20, and 21. The corresponding t statistics and t critical values are also presented in these tables. As we can observe for each dataset and model, t statistic is larger than the t critical value, which supports the rejection of the null hypothesis $\mathcal{H}_0$ in 3.4.
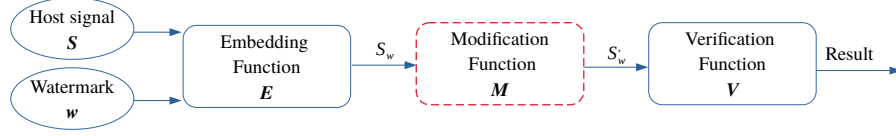
Figure 11: Digital watermarking life cycle.

TABLE 19: 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 20: 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 21: Accuracy of the watermarked models and clean models on watermarked data for node classification task ($n = 10$).

Setting	Models	Watermark Accuracy (%)										t	ν	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			

C. Hyperparameter Setting

Table 22 summarizes the default hyperparameter setting in the model architecture and training.

D. Additional Experimental Results

D.1. Model Pruning

The watermarking performance after model pruning on the COLLAB, REDDIT-BINARY, and CiteSeer

Authorized licensed use limited to: TU Delft Library. Downloaded on August 31,2023 at 12:03:25 UTC from IEEE Xplore. Restrictions apply.

TABLE 22: 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 [19]
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 [62].

TABLE 23: Watermarking performance on graph classification task after model pruning (COLLAB).

Pruning rate	DiffPool		GIN		GraphSAGE	
	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$
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 24: Watermarking performance on graph classification task after model pruning (REDDIT-BINARY).

Pruning rate	DiffPool		GIN		GraphSAGE	
	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$	Test Acc. $(D_{wm}^t\|D_{wm}^r)$	Watermark Acc. $(D_{wm}^t\|D_{wm}^r)$
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%

TABLE 25: 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%

datasets is shown in Tables 23, 24, and 25, 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

TABLE 26: Watermarking performance on graph classification task after fine-pruning (COLLAB).

Pruning rate	DiffPool		GIN		GraphSAGE	
	Test Acc. $(D^t_{wm}|D^r_{wm})$	Watermark Acc. $(D^t_{wm}|D^r_{wm})$	Test Acc. $(D^t_{wm}|D^r_{wm})$	Watermark Acc. $(D^t_{wm}|D^r_{wm})$	Test Acc. $(D^t_{wm}|D^r_{wm})$	Watermark Acc. $(D^t_{wm}|D^r_{wm})$
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 27: Watermarking performance on graph classification task after fine-pruning (REDDIT-BINARY).

Pruning rate	DiffPool		GIN		GraphSAGE	
	Test Acc. $(D^t_{wm}|D^r_{wm})$	Watermark Acc. $(D^t_{wm}|D^r_{wm})$	Test Acc. $(D^t_{wm}|D^r_{wm})$	Watermark Acc. $(D^t_{wm}|D^r_{wm})$	Test Acc. $(D^t_{wm}|D^r_{wm})$	Watermark Acc. $(D^t_{wm}|D^r_{wm})$
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%

is robust to model pruning, but the plagiarizer can still eliminate our watermarks with the cost of high accuracy drop in the main task.

D.2. Fine-pruning

The watermarking performance after fine-pruning on the COLLAB and REDDIT-BINARY datasets is shown in Tables 26 and 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 3. The results for REDDIT-BINARY follow the same behavior. Therefore, these results further verify that our watermarking mechanism is robust to fine-pruning.