NI-GDBA: Non-Intrusive Distributed Backdoor Attack Based on Adaptive Perturbation on Federated Graph Learning

Anonymous Author(s)

Abstract

Federated Graph Learning (FedGL) is an emerging Federated Learning (FL) framework that learns the graph data from various clients to train better Graph Neural Networks(GNNs) model. Owing to concerns regarding the security of such framework, numerous studies have attempted to execute backdoor attacks on FedGL, with a particular focus on distributed backdoor attacks. However, all existing methods posting distributed backdoor attack on FedGL only focus on injecting distributed backdoor triggers into the training data of each malicious client, which will cause model performance degradation on original task and is not always effective when confronted with robust federated learning defense algorithms, leading to low success rate of attack. What's more, the backdoor signals introduced by the malicious clients may be smoothed out by other clean signals from the honest clients, which potentially undermining the performance of the attack.

To address the above significant shortcomings, we propose a non-intrusive graph distributed backdoor attack(NI-GDBA) that does not require backdoor triggers to be injected in the training data. Our attack trains an adaptive perturbation trigger generator model for each malicious client to learn the natural backdoor from the GNN model downloading from the server with the malicious client's local data. In contrast to traditional distributed backdoor attacks on FedGL via trigger injection in training data, our attack on different datasets such as Molecules and Bioinformatics have higher attack success rate, stronger persistence and stealth, and has no negative impact on the performance of the global GNN model. We also explore the robustness of NI-GDBA under different defense strategies, and based on our extensive experimental studies, we show that our attack method is robust to current federated learning defense methods, thus it is necessary to consider non-intrusive distributed backdoor attacks on FedGL as a novel threat that requires custom defenses. Code is available at an anonymous github repository: https://anonymous.4open.science/r/NI-GDBA-64E5/

CCS Concepts

• **Security and privacy → Distributed systems security**.

Keywords

Federated Graph Learning, Backdoor Attacks

ACM Reference Format:
Anonymous Author(s). 2018. NI-GDBA: Non-Intrusive Distributed Backdoor Attack Based on Adaptive Perturbation on Federated Graph Learning. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Graph data is a very pervasive type of data in the Web, which can be found in recommender systems[4, 8], social networks[6, 41], financial systems[2, 27]. Graph Neural Networks (GNNs) generalize traditional Deep Neural Networks (DNNs) to learn graph data, thereby opening up a promising avenue for effectively learning from complex graph data. In certain scenarios of Web, like the development of recommender systems, having comprehensive graph data from platforms such as Taobao, Temu, and Amazon simultaneously would enable us to construct a recommender system that effectively caters to diverse geographic demographics. However, stringent privacy policies and fierce business competition[17] prevent the sharing of graph data between platforms. In practice, the performance of GNN models trained individually by platforms with their own graph data is often unsatisfactory, therefore, a distributed training approach where organizations collaborate to train a GNN model without exchanging graph data is critically necessary[12].

Federated Graph Learning(FedGL) is a promising solution to address such data isolation/privacy issues[13, 18, 20–24, 26, 28, 30, 46]. Specifically, FedGL allows a central server to coordinate multiple clients in training a GNN model collaboratively, without sharing their graph data. In FedGL, each client has its own set of graphs. The server and clients collaboratively train a shared GNN model without directly accessing the clients' graphs. The learnt shared GNN model is then used by all clients for testing.

Although FedGL can successfully train high-performing GNN models with graph data scattered across various platforms, it may be susceptible to potential security vulnerabilities when applied in real Web applications. Recent researches have attempted to examine the security issues of FedGL in real Web application scenarios through backdoor attacks[3, 31, 34, 36, 39, 43, 45, 47]. The so-called backdoor attacks involve embedding a malicious and concealed backdoor within deep learning models. The backdoor model would behave normally on benign inputs, but the hidden backdoor will be activated to mislead the model when the attack-defined trigger is presented[9, 15, 38]. Within FedGL, there emerges a novel and highly threatening form of attack known as distributed backdoor attack[16, 33, 35, 37, 40]. In such distributed backdoor attack, an attacker controls a fraction of clients as malicious clients, who aim to embed a backdoor within the GNN model. These clients inject different local backdoor triggers, such as various subgraphs, into part of their training data and label it with a target label chosen by the attacker, which is different from the true label. When the model is deployed in real-world scenarios, to activate hidden backdoors, it merely requires injecting those local triggers into the graph data.

However, such distributed backdoor attack based on injecting triggers into training data has several obvious drawbacks. Xu et al. [37] firstly proposed distributed backdoor attack in FedGL based on data poisoning, however, due to the fact that the triggers are randomly generated subgraphs, backdoor signals are easily smoothed out by clean signals, leading to decreasing attack success rates with increasing training rounds in the case of a small number of malicious clients. Liu et al. [16] improves the distributed backdoor attacks in FedGL through more appropriate trigger position selection as well as trigger generation algorithms. However, since it still has to inject triggers into the training data, which can cause the local model parameters of the malicious clients to be significantly different from those of the honest clients, thus its attack success rate is not satisfactory under the application of various federated learning defense algorithms. Xu et al. [35] further suggests that, in FedGL, the negative impact of distributed backdoor attack on the original task of the global GNN model is severe. Based on these drawbacks, we propose the following questions.

RQ1: Can distributed backdoor attack achieve high attack success rate without affecting the performance of the GNN model on the original task?

RQ2: Can distributed backdoor attack continue to ensure that backdoor signals brought in by malicious clients will not be smoothed out by clean signals from honest clients when the total number of malicious clients is limited, thus ensuring the effectiveness of the backdoor attack?

RQ3: Can distributed backdoor attack remain effective with existing federated learning defense algorithms?

To answer the questions mentioned above, we introduce a non-intrusive graph distributed backdoor attack(NI-GDBA) based on adaptive perturbation .

Our non-intrusive distributed backdoor attack on FedGL: The main reason we find that past distributed backdoor attacks on FedGL have those shortcomings mentioned above is the use of trigger injection in training data. Previous researches all attempted to have the model construct mappings between data containing subgraph triggers and the target label during the training process. To compensate for the shortcomings of such attack, we propose a non-intrusive distributed backdoor attack on FedGL. In our NI-GDBA, we propose a perturbation trigger generator model that adaptively optimizes trigger without interfering with the FedGL training process. And it learns the most suitable local trigger for each malicious client. The perturbation trigger generator model occurs in two steps: 1) The malicious clients provide local clean training data to participate in FedGL training process, and then downloads the global GNN model from the server. This process solves the drawbacks of previous backdoor attacks causing the model's performance degradation on the original task as well as the problem of backdoor signals being smoothed out by the clean signals of the honest client, which also provides answers to RQ1 and RQ2. 2) In each malicious client, the global GNN model is applied to train a local perturbation trigger generator model, which continuously optimizes the local perturbations trigger to learn the global GNN model's natural backdoor[42]) based on its loacl graph data. This process makes our perturbation trigger able to achieve effective backdoor attacks without being detected by various federated learning defense algorithms, which are designed for FedGL training process. And it also offers a brilliant answer to RQ3.

Empirical and theoretical evaluations: We extensively evaluate the NI-GDBA attack on six benchmark graph datasets. Our attack results excellently answer RQ1,RQ2 and RQ3 presented above.

RA1: Distributed backdoor attack can achieve high attack success rate without affecting the performance of the GNN model on the original task. Our NI-GDBA is able to maintain an average attack success rate of over 97.5% while having no negative impact at all on the performance of the GNN model on the original task.

RA2: Distributed backdoor attack can ensure the effectiveness of backdoor attack when the number of malicious clients is limited. Our NI-GDBA attack performance is largely independent of the number of malicious clients, and according to our experimental results, its average attack success rate drops by less than 0.5% when the number of malicious cliens decreases.

RA3: Distributed backdoor attack can be still effective with existing federated learning defense algorithms. With an average attack success rate over 97% , the average backdoor performance of NI-GDBA improves by 60% to 64% compared to existing work under two federated learning defense algorithms, which validates our NI-GDBA's groundbreaking success in answering RQ3.

Contributions:

- We propose a non-intrusive graph distributed backdoor attack(NI-GDBA) method that enables effective, stealthy and persistent backdoor attack on FedGL.
- Our NI-GDBA effectively addresses the significant problems associated with the impact of distributed backdoor attacks within FedGL. It does so by clean FedGL training process and by developing adaptive perturbation triggers with the global GNN model.
- We have developed a novel non-intrusive framework for distributed backdoor attack that leverages the natural backdoors in global GNN model. To our knowledge, this is the first study to successfully implement distributed backdoor attack in FedGL without disrupting the training process.

2 BACKGROUND AND PROBLEM DEFINITION

2.1 Federated Graph Learning (FedGL)

We define the graph as $G = (V, E, X)$, where V is node set, E is the edge set, and $X \in \mathbb{R}^{|V| \times d}$ is node feature matrix, with d the dimensionality of the features and $|V|$ the total number of the nodes. The adjacency matrix $A \in \{0, 1\}^{|V| \times |V|}$ is introduced, where $A_{u,v} = 1$ if the edge (u, v) is an element of E, and 0 otherwise. In the context of this study, graph classification is identified as the focal task, with each graph G assigned a label y from the label space $\mathcal{Y}$. The process of graph learning (GL) involves the ingestion of a graph G to train a graph classifier f, which predicts the graph's label, formalized as $f : G \to \mathcal{Y}$.

Federated Graph Learning(FedGL) extends GL in the federated learning setting. It enables C clients to train a global GNN model θ collaboratively without revealing local datasets. In contrast to centralized graph learning, which collects diverse graph data at a central server before training, FedGL operates by having clients upload the weights of their local models (denoted as $\{\theta^i | i \in C\}$) to a parametric server. Specifically, the objective of FedGL is to refine the loss function for enhanced performance

$$\min_{\theta} \ell(\theta) = \sum_{i=1}^{C} \frac{k_i}{C} L_i(\theta), L_i(\theta) = \frac{1}{k_i} \sum_{j \in P_i} \ell_j(\theta, x_j), \tag{1}$$

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

In the t-th iteration, FedGL adheres to the following three-step procedure:

- *Global model download*. The clients download the latest global GNN model θ_t from the server
- *Local training*. The client refines the model parameters θ_t with its local training graph data and updates its parameters according to the rule: $\theta_t^i \leftarrow \theta_t^i - \eta \frac{\partial L(\theta_t, b)}{\partial \theta_t^i}$, where θ_t^i represents the parameters of the model at iteration t for the i-th client, and η is the learning rate.
- *Aggregation.* After the clients upload local models, the server updates the global model by aggregating the local models with specific aggregation algorithms.

2.2 Distributed Backdoor Attacks on FedGL

In the context of a backdoor attack against Federated Graph Learning (FedGL), previous scholarly endeavors have focused on the injection of effective triggers into the training data. This manipulation is designed to induce the global GNN model to establish a conditional mapping, wherein the presence of the graphs injected with trigger is associated with specific target label. We categorize this approach as an intrusive distributed backdoor attack. Building upon this concept, we introduce a novel non-intrusive distributed backdoor attack.

- **Intrusive Distributed Backdoor Attack:** Backdoor attack by injecting various triggers into each malicious client's training data, which can interfere with the normal machine learning model training process.
- **Non-Intrusive Distributed Backdoor Attack:** Backdoor attack realized by multiple malicious clients cooperating without interfering with the machine learning model training process.

For intrusive distribued backdoor attack on FedGL, a recent work[37] inspired by [44] proposes a kind of such backdoor attack by applying random subgraphs as triggers, which is named Rand-GDBA in [40], where the prefix "Rand" means malicious clients randomly choose nodes from their clean graphs as the location to inject the trigger. Meanwhile, a recent study[16] extends the methodology for trigger generating algorithm in Rand-GDBA, which are tested by us in our experiments.

Rand-GDBA: Each malicious client i has its local trigger k_i, and inject it into a fraction of its training graph data, where the inject position is nodes randomly selected from the graph. After the backdoored graphs G_b^i is formed by each malicious client i, the local backdoored model θ_B^i of malicious client i can be learned as below:

$$\theta_B^i = \arg\min_{\theta_B^i} L(G_B^i \cup G_C^i; \theta) \tag{2}$$

where θ is the global model, G_C^i contains the remaining clean graphs in G^i. The server will aggregate the local models with specific aggregation algorithms. The final backdoored GNN model is shared with all clients. When the model is deployed in real-world scenarios, injecting all malicious clients' local triggers k_i into the graph data with randomly chosed nodes as trigger position can activate the hidden backdoor in the model.

2.3 Threat Model

Our goal is to answer questions that distributed backdoors based on trigger injection in training data leaves. As an attacker, we hope to achieve a stealthy and effective distributed backdoor attack without interfering with the training process.

Attacker: We assume the attacker manipulates a number c^m of the total C clients, namely malicious clients.

- **Attacker's knowledge:** All malicious clients only know their own training graphs and the global GNN model during FedGL training.
- **Attacker's capability:** Malicious clients can use local graph data to learn natural backdoor of the global GNN model.
- **Attacker's objective:** Malicious clients aim to learn a backdoored FedGL model such that: it predicts the backdoored testing graphs as the target label, while correctly predicting the clean testing graphs. This implies the model will achieve a high attack success rate as well as a high original task accuracy.

3 Non-intrusive DBA on FedGL

Our NI-GDBA consists of two main steps. First, the malicious clients supply local unmodified graph data to acquire a clean global GNN model. Second, each malicious client learns the GNN model's natural backdoor by training a local adaptive perturbation trigger generator model with their own graph data.

3.1 FedGL Training with Clean Graph Data

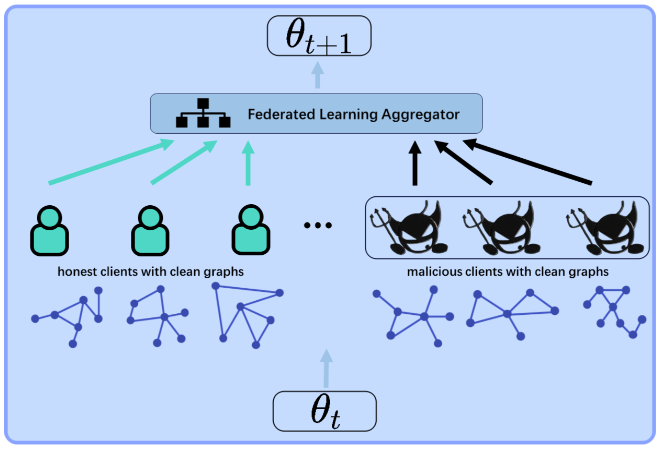
θ_{t+1}
Federated Learning Aggregator
honest clients with clean graphs
malicious clients with clean graphs
θ_t

Figure 1: Global GNN model training process of our NI-GDBA.

As show in figure 1, the malicious clients provide local clean graph data for FedGL just like honest clients to train a global GNN model, which will be used for the adaptive perturbation trigger generator model's training process.

Algorithm 1 FedGL training with clean graph data

```
Input: Total clients $C$ with clean graphs $\{G^i\}_{i\in C}$, training iterations $iter$,
initial global model $\theta_1$
Output: Global GNN model $\theta_{iter}$.
1: for each iteration $t$ in [1,iter] do
2:     for each client $i \in C$ do
3:         $\theta_t^i = \arg\min_{\theta^i} L(G^i;\theta_t)$
4:     end for
5:     Server randomly selects $C_t$ clients for aggregation
6:     $\theta_{t+1} = \frac{1}{|C_t|}\sum_{i\in C_t}\theta_t^i$
7: end for
```

Local Model Training: As described in Algorithm 1, for each client i, it updates the local model via minimizing the loss on all its clean graphs $\{G^i\}_{i\in C}$ as:

$$\theta_t^i = \arg\min_{\theta^i} L(G^i;\theta_t) \tag{3}$$

Updating the Global GNN Model: The server averages the local models θ^i of the selected clients to update the global GNN model θ.

3.2 Learning Natural Backdoor of GNN Model

We learn the natural backdoor of global GNN model through an adaptive trigger generator model, which has two main parts: trigger location learning and perturbation trigger learning. The former is used to find the appropriate trigger location for trigger injection, and the latter is applied to generate the adaptive perturbation trigger that can effectively realize the backdoor attack.

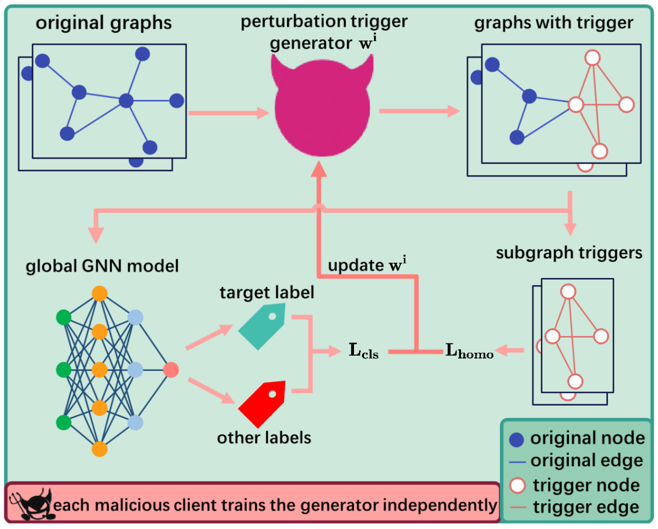

Figure 2: Perturbation trigger generator model training process.

1)Trigger Location Learning: To reduce the time complexity of natural backdoor learning of GNN model, we optimize the trigger location learning by a specific algorithm rather than an iterative optimization algorithm based on the results of backdoor attacks on local training graphs of malicoius clients. Specifically, we introduce an efficient and low time-complexity clustering algorithm to select the important nodes in a graph to be injected with perturbation triggers, which uses a customized clustering coefficient c_u defined as follows to measure the influence of a node in the graph[16].

Unweighted Graphs: For unweighted graphs, the clustering coefficient c_u for a node u is defined as the fraction of possible triangles through that node that actually exist. This is calculated using the formula:

$$c_u = \frac{2T(u)}{deg(u)(deg(u)-1)} \tag{4}$$

where $T(u)$ is the number of triangles through node u, and $deg(u)$ is the degree of u. If $deg(u) < 2$, the clustering coefficient is set to 0.

Weighted Graphs: For weighted graphs, the clustering coefficient is generalized to account for the weights of the edges forming the triangles. The function computes the geometric average of the subgraph edge weights:

$$c_u = \frac{1}{deg(u)(deg(u)-1)} \sum_{vw} (\hat{w}_{uv}\hat{w}_{uw}\hat{w}_{vw})^{1/3} \tag{5}$$

where $\hat{w}_{uv}$ represents the normalized edge weight w_{uv} with respect to the maximum weight in the network.

Directed Graphs: For directed graphs, the clustering coefficient considers the directionality of the edges. The formula for the directed clustering coefficient c_u is:

$$c_u = \frac{T(u)}{2(deg^{tot}(u)(deg^{tot}(u)-1) - 2deg^{\leftrightarrow}(u))} \tag{6}$$

where $T(u)$ is the number of directed triangles through node u, $deg^{tot}(u)$ is the sum of the in-degree and out-degree, and $deg^{\leftrightarrow}(u)$ is the reciprocal degree.

2)Perturbation Trigger Learning: As shown in figure 2 and algorithm 2, the adaptive perturbation trigger generator model learns the natural backdoor of the GNN model. In our backdoor attack, graph G is fed to the adaptive perturbation trigger generator model, then a suitable perturbation trigger is added after the trigger location is found. We then feed the graph G to the GNN model for prediction and continuously optimize our adaptive perturbation trigger generator model based on the prediction results and the covert nature of the perturbation triggers so that it can generate more effective perturbation triggers under various federated learning defense algorithms. Specifically, our preliminary optimization goal is to minimize the the cross entropy loss for graph classification:

$$L_{cls} = -\frac{1}{N}\sum_{i=1}^{N} -(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)) \tag{7}$$

where y_i is the target label of backdoor attack, and $\hat{y}_i$ is the prediction label of the GNN model.

For the architecture of the adaptive perturbation trigger generator model ω^i, it proposes a perturbation trigger g_t tailored to a given subgraph g within G. At a high level, ω^i contains two key operations: (i) it first encodes both subgraph g's node features and

Algorithm 2 Learning Natural Backdoor of GNN Model

Input: Malicious clients c^m with clean graph $\{G^i\}_{i\in c^m}$, GNN model θ, malicious clients' initial adaptive perturbation trigger generator model $\{\omega_1^i\}_{i\in c^m}$, training iterations $iter^i{}_{i\in c^m}$, trigger node size n_{tri}
Output: Malicious clients' perturbation trigger $\{\omega^i\}_{i\in c^m}$

```
1: for each client $i \in c^m$ do
2:     for each iteration $t$ in $[1, iter^i]$ do
3:         Client $i$ divides $G^i$ into $G^i_{cm}$ and to-be-backdoored $G^i_o$
4:         $\mathbf{s}^i$ = Node_score($G^i_o$) // Node importance score
5:         $\mathcal{V}^i_{def}$ = rank($\mathbf{s}^i$, $n_{tri}$) // Trigger location
6:         $\tilde{G}^i_B$ = Perturbation_Trigger_Learning($G^i_o$, $\mathcal{V}^i_{def}$)
7:         $\omega^i_t = \arg\min_{\omega^i} L(\tilde{G}^i_B; \theta)$
8:     end for
9: end for
```

topological structures, which maps each node i in g to its encoding z_i ; (ii) it applies two generator functions structured by neural networks. One is used for mapping g's node encodings to g_t's topological structures, and the other is designed for mapping g's node encodings to g_t's node features. These two neural networks work together to accomplish the formation of the perturbation trigger.

i) Encoder for g's feature and structure: We employ recent advances on graph attention mechanisms[25] to encode the node features and topological structure of graph g. When considering a specific node pair i and j, we determine an attention coefficient α_{ij} that quantifies the significance of node j in relation to node i, taking into account their respective node features and topological connection. Subsequently, we construct the representation for node i by aggregating its neighboring encodings, with each encoding weighted according to the pertinent attention coefficient, following an application of a nonlinear transformation. Here, we use $z_i \in \mathbb{R}^d$ to represent the encoding of node i, where d is the encoding dimensionality.

ii) Mapping g's encoding to g_t: Given two nodes $i, j \in g$ with their encodings z_i and z_j , we define their corresponding connectivity $\tilde{A}_{ij}$ in g_t using their parameterized cosine similarity:

$$\tilde{A}_{ij} = \mathbb{1}_{z_i^\top W_c^\top W_c z_j \geq \|W_c z_i\| \|W_c z_j\|/2} \tag{8}$$

where $W_c \in \mathbb{R}^{d\times d}$ is learnable and $\mathbb{1}_p$ is an indicator function returning 1 if p is true and 0 otherwise. Intuitively, i and j are connected in g_t if their similarity score exceeds 0.5.

Meanwhile, for node $i \in g$, we define its feature $\tilde{X}_i$ in g_t as

$$\tilde{X}_i = \sigma(W_f z_i + b_f) \tag{9}$$

where $W_f \in \mathbb{R}^{d\times d}$ and $b_f \in \mathbb{R}^d$ are both learnable, and $\sigma(\cdot)$ is a non-linear activation function.

In order to improve the covert of perturbation trigger injection, we restrict the generation of perturbation triggers based on the homogeneity assumption, specifically, we introduce the following homogeneity loss:

$$L_{homo} = \frac{1}{|E|} \sum_{(i,j)\in E} \max\left(0, \theta - \mathrm{sim}(x_i, x_j)\right) \tag{10}$$

where $|E|$ represents the number of edges in the graph, (i, j) denotes an edge in the graph, θ is a predefined similarity threshold, and $\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j)$ represents the similarity between nodes i and j.

Then the final optimization objective of our adaptive perturbation trigger generator model is to minimize the following losses:

$$L = L_{cls} + \beta L_{homo} \tag{11}$$

4 Attack Results

Next, we conduct an empirical study of NI-GDBA to answer the following key question:
Q1 - How much will NI-GDBA negatively affect the performance of the GNN model on the original task?
Q2 - How effective is the NI-GDBA with a limited number of malicious clients?
Q3 - How effective is the NI-GDBA when confronted with existing federated learning defense algorithms?

4.1 Experimental Setup

We implemented our NI-GDBA on FedGL using the PyTorch framework. All experiments were run on a server with 8 NVIDIA 4090 GPU. Each experiment was repeated five times with different random seed to obtain the average attack results.
Datasets and training/testing sets: We evaluate our attack on six benchmark real-world graph datasets for graph classification, which are presented in detail in Table 1. For each dataset, we randomly sample 80% of the data instances as the training dataset and the rest as the test dataset.

Table 1: Statistics of datasets

Datasets	Graphs	Classes
AIDS	2,000	2
NCI1	4,110	2
PROTEINS-full	1,113	2
DD	1,178	2
ENZYMES	600	2
COLORS-3	10,500	11

Attack baseline: We compare our NI-GDBA with Rand-GDBA with four different trigger generation algorithms. We obtained the name of the method in our experiments by directly splicing Rand-GDBA with the corresponding trigger generation method. For example, Rand-GDBA with trigger generation model Watts-Strogats (WS) model [29] is called Rand-GDBA-WS

- **Rand-GDBA-ER:** Each malicious client, Following [44], generates the trigger with the Erdős-Rényi (ER) random graph model[10], where the number of edges e_{tri} with a trigger node size n_{tri} can be controlled. These triggers are subsequently attached to random nodes within the graphs that are to be backdoored. To amplify the effectiveness of the attack, we also ensure a diverse set of local triggers is used across different malicious clients. This is achieved by storing a collection of local triggers generated with the ER model and distributing them uniquely to each malicious client. The only difference between Rand-GDBA-ER and the following methods is the trigger generation algorithm, which remains consistent in other experimental settings.

- **Rand-GDBA-WS:** It generates the local triggers by Watts-Strogats model.
- **Rand-GDBA-BA:** It uses the Barabási-Albert (BA) model[1] to generate triggers.
- **Rand-GDBA-RR:** Its triggers are generated by random regular graph (RR) model[5].
- **Our NI-GDBA:** Each malicious client provides local clean graph data to participate in the FedGL training process. Then the malicious clients learn the natural backdoor of the global GNN model by the adaptive perturbation trigger generator model on their local training graphs. Note that the trigger node size n_{tri} and trigger edge size e_{tri} are predefined (same as methods above).

During testing, the local triggers are combined into a global trigger. For fair comparison, we let all attacks use a complete subgraph as the global trigger. In our NI-GDBA, for each testing graph, we learn the vital nodes that determine the trigger location, and then add perturbation trigger with each malicious client's local trigger generator model.

Parameter setting: Due to the mutual influence of different parameters, we adopt different parameter setting schemes for the solutions of Q1, Q2, and Q3 to make the comparison more fair.

- **For Q1:** In order to examine the impact of all the above distributed backdoor attacks on the performance of the GNN model on the original task, we make all the methods have high attack success rates without defense algorithms by suitable parameter settings, but since the attack success rates of the methods other than NI-GDBA on AIDS and COLOR-3 are too low, we only select the remaining four datasets for the experiments. We also introduce a method named None that trains the GNN model without any backdoor attack to compare with the attacks above. During FedGL training, we use a total of $C = 5$ clients and evenly distribute the training graphs in each dataset to the clients. The total number of iterations is 40 in all datasets. Graph Convolutional Networks(GCN)[14], Graph Attention Networks(GAT)[25] and GraphSAGE[11] are introduced respectively as the graph classifier. There are several hyperparameters that can affect all attacks' performance on FedGL: number of malicious clients c^m, fraction of trigger nodes n_{tri}, fraction of trigger edges e_{tri} and fraction of each malicious client's training data injected with trigger p. In our NI-GDBA, we define that the fraction of each malicious client's training data used for perturbation trigger learning is p. We set $c^m = 2$, $n_{tri} = 0.5$, $e_{tri} = 0.3$ and $p = 0.5$ by default.
- **For Q2:** In order to explore whether the backdoor signals introduced by different distributed backdoor attacks are easily smoothed out when the number of malicious clients is extremely limited. we extend the total number of clients C to 10, set $n_{tri} = 0.1$, $e_{tri} = 0.3$ and $p = 0.2$ as a mean of examining the extent to which the effectiveness of the backdoor attack is weakened in the presence of a decreasing number of malicious clients. The total number of iterations is 40 in all datasets and the clients use GCN as the graph classifier.
- **For Q3:** To examine the performance of various distributed backdoor attacks under different defense strategies, we use three strategies, specifically: no defense, foolsgold[7] and federated averaging(Fedavg)[19]. The total number of clients $C = 10$, $n_{tri} = 0.1$, $e_{tri} = 0.3$ and $p = 0.2$. The number of FedGL training iterations is 40 in all datasets, and the graph classifier is GCN.

Evaluation metrics: We use the attack success rate (ASR) to evaluate the attack effectiveness and original task accuracy(OA) to evaluate the GNN model's performance.

$$Attack\ Success\ Rate(ASR) = \frac{\#successful\ trials}{\#total\ trials} \tag{12}$$

4.2 Experimental Results

4.2.1 Main results of the compared attacks. Table 2 shows the comparison results of the attacks' impact on GNN models' performance on original task in the default setting. Table 3 shows the comparison results of the attacks with decreasing number of malicious clients. Table 4, 5 and 6 shows the comparison results of the attacks with various defense strategy. We have the below key observations:

(1) For Q1: NI-GDBA has no negative impact on the performance of the GNN model on the original task: All attack methods, except for NI-GDBA, cause OA decrease of the original GNN model by 5% to 8% and achieve a close OA(i.e., the differences between them in all cases are $\leq$ 3%). The OA of the GNN model when subjected to the NI-GDBA attack remains consistent with the OA of the GNN model in the absence of any attack. This demonstrates that we have brilliantly answered to Q1.

(2) For Q2: NI-GDBA's attack performance is largely independent of the number of malicious clients: For all attacks other than NI-GDBA, the ASR decreases by an average of 7% to 11% as the number of malicious clients decreases. However, our NI-GDBA exhibits an average ASR decrease of less than 1% and with the reduction of clients, while keeping an average ASR of over 98.3%. This demonstrates that our NI-GDBA are still effective in scenarios with limited number of malicious clients in distributed backdoor attacks, thereby answering Q2.

(3) For Q3: NI-GDBA performs well with existing federated learning defense algorithms: Our experimental findings indicate that, when faced with federated learning defense algorithms, attack methods other than NI-GDBA experience an average decrease in ASR ranging from 1% to 4.5%. In contrast, our NI-GDBA exhibits a decrease in ASR of less than 0.3% under the same conditions, while maintaining an average ASR exceeding 96.5%. This confirms that NI-GDBA remains effective in executing attacks despite the presence of current federated learning defense algorithms, which offers a satisfying answer for Q3.

4.2.2 Impact of hyperparameters on our NI-GDBA. In this set of experiments, we will study in-depth the impact of other important hyperparameters on NI-GDBA.

Impact of the fraction of training data p applied in NI-GDBA. Tabel 4, 5, 6 show the attack results when $p = 0.1, 0.2, 0.3$. For instance, on NCI1, when p is from 0.1 to 0.3 with defense as foolsgold, the ASR of NI-GDBA is always 100%, but the ASR of Rand-GDBA-ER, Rand-GDBA-WS, Rand-GDBA-BA, Rand-GDBA-RR can be increased from 50% to 60%, from 57% to 74%, from 57% to %70 and from 60% to 79%. This demonstrates that, unlike other attack methodologies, the ASR achieved by our NI-GDBA remains largely

Table 2: GNN classifier's performance on original task of all the compared attacks in the Q1 default setting.

Datasets	NCI1			PROTEINS-full			DD			ENZYMES		
GNN model	GCN	GAT	GraphSAGE	GCN	GAT	GraphSAGE	GCN	GAT	GraphSAGE	GCN	GAT	GraphSAGE
None	**0.77**	**0.76**	**0.76**	**0.66**	**0.71**	**0.70**	**0.61**	**0.57**	**0.66**	**0.42**	**0.38**	**0.33**
NI-GDBA	**0.77**	**0.76**	**0.76**	**0.66**	**0.71**	**0.70**	**0.61**	**0.57**	**0.66**	**0.42**	**0.38**	**0.33**
Rand-GDBA-ER	0.67	0.68	0.68	0.65	0.66	0.66	0.45	0.47	0.65	0.33	0.25	0.25
Rand-GDBA-WS	0.73	0.69	0.70	0.66	0.65	0.66	0.55	0.47	0.61	0.33	0.33	0.29
Rand-GDBA-BA	0.66	0.70	0.61	0.65	0.66	0.68	0.53	0.49	0.60	0.33	0.29	0.29
Rand-GDBA-RR	0.65	0.71	0.62	0.66	0.61	0.66	0.59	0.48	0.63	0.33	0.33	0.29

Table 3: Attack results of all the compared attacks in the Q2 default setting

Methods	**NI-GDBA**			**Rand-GDBA-ER**			**Rand-GDBA-WS**			**Rand-GDBA-BA**			**Rand-GDBA-RR**		
c^m	2	3	4	2	3	4	2	3	4	2	3	4	2	3	4
AIDS	**1.00**	**1.00**	**1.00**	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00
NCI1	**1.00**	**1.00**	**1.00**	0.42	0.56	0.64	0.54	0.70	0.74	0.55	0.68	0.72	0.54	0.72	0.73
PROTEINS-full	**0.97**	**1.00**	**1.00**	0.44	0.53	0.67	0.28	0.53	0.67	0.44	0.53	0.67	0.43	0.53	0.68
DD	**1.00**	**1.00**	**1.00**	0.70	0.84	0.93	0.56	0.53	0.60	0.71	0.84	0.94	0.57	0.50	0.57
ENZYMES	**0.94**	**0.93**	**0.86**	0.11	0.41	0.66	0.19	0.45	0.69	0.16	0.42	0.58	0.15	0.48	0.63
COLOR-3	**0.99**	**1.00**	**1.00**	0.05	0.07	0.12	0.06	0.06	0.12	0.05	0.06	0.11	0.05	0.06	0.12

Table 4: Attack results of all the compared attacks in the Q3 default setting without defense

Methods	**NI-GDBA**			**Rand-GDBA-ER**			**Rand-GDBA-WS**			**Rand-GDBA-BA**			**Rand-GDBA-RR**		
p	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
AIDS	**1.00**	**1.00**	**1.00**	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
NCI1	**1.00**	**1.00**	**1.00**	0.57	0.56	0.66	0.52	0.70	0.81	0.58	0.68	0.74	0.67	0.72	0.76
PROTEINS-full	**1.00**	**1.00**	**1.00**	0.52	0.53	0.58	0.54	0.53	0.60	0.53	0.53	0.60	0.52	0.53	0.58
DD	**1.00**	**1.00**	**1.00**	0.76	0.84	0.80	0.60	0.53	0.59	0.73	0.84	0.76	0.54	0.50	0.57
ENZYMES	**0.81**	**0.93**	**0.84**	0.19	0.41	0.61	0.14	0.45	0.61	0.16	0.42	0.64	0.17	0.48	0.61
COLOR-3	**1.00**	**1.00**	**1.00**	0.06	0.07	0.07	0.06	0.06	0.07	0.05	0.06	0.07	0.05	0.06	0.07

Table 5: Attack results of all the compared attacks in the Q3 default setting with defense as foolsgold

Methods	**NI-GDBA**			**Rand-GDBA-ER**			**Rand-GDBA-WS**			**Rand-GDBA-BA**			**Rand-GDBA-RR**		
p	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
AIDS	**1.00**	**1.00**	**1.00**	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.02	0.01
NCI1	**1.00**	**1.00**	**1.00**	0.50	0.64	0.60	0.57	0.68	0.74	0.57	0.65	0.70	0.60	0.74	0.79
PROTEINS-full	**1.00**	**1.00**	**1.00**	0.48	0.59	0.65	0.48	0.59	0.58	0.48	0.59	0.58	0.47	0.58	0.57
DD	**1.00**	**1.00**	**1.00**	0.77	0.79	0.60	0.56	0.58	0.60	0.79	0.82	0.55	0.53	0.52	0.57
ENZYMES	**0.85**	**0.92**	**0.87**	0.22	0.38	0.56	0.19	0.45	0.51	0.21	0.36	0.61	0.21	0.39	0.59
COLOR-3	**1.00**	**1.00**	**1.00**	0.06	0.07	0.07	0.05	0.07	0.06	0.06	0.07	0.07	0.06	0.07	0.06

Table 6: Attack results of all the compared attacks in the Q3 default setting with defense as Fedavg

Methods	**NI-GDBA**			**Rand-GDBA-ER**			**Rand-GDBA-WS**			**Rand-GDBA-BA**			**Rand-GDBA-RR**		
p	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
AIDS	**1.00**	**1.00**	**1.00**	0.01	0.01	0.03	0.02	0.05	0.03	0.01	0.03	0.01	0.02	0.01	0.19
NCI1	**1.00**	**1.00**	**1.00**	0.23	0.33	0.46	0.30	0.36	0.56	0.53	0.26	0.30	0.46	0.41	0.23
PROTEINS-full	**1.00**	**1.00**	**1.00**	0.61	0.53	0.72	0.49	0.81	0.81	0.58	0.72	0.75	0.56	0.54	0.70
DD	**1.00**	**1.00**	**1.00**	0.94	0.88	0.87	0.45	0.69	0.40	0.35	0.53	0.92	0.46	0.48	0.39
ENZYMES	**0.63**	**0.96**	**0.83**	0.28	0.19	0.14	0.14	0.42	0.77	0.21	0.23	0.53	0.19	0.12	0.24
COLOR-3	**1.00**	**1.00**	**1.00**	0.04	0.10	0.09	0.02	0.06	0.19	0.04	0.10	0.18	0.10	0.14	0.06

Table 7: Comparing two trigger location learning schemes in our NI-GDBA

Dataset	**AIDS**			**NCI1**			**PROTEINS-full**			**DD**			**ENZYMES**			**COLOR-3**		
p	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3			
Cluster	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**0.81**	**0.93**	0.84	**1.00**	1.00	**1.00**
Random	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.73	0.85	**0.88**	0.99	1.0	0.99

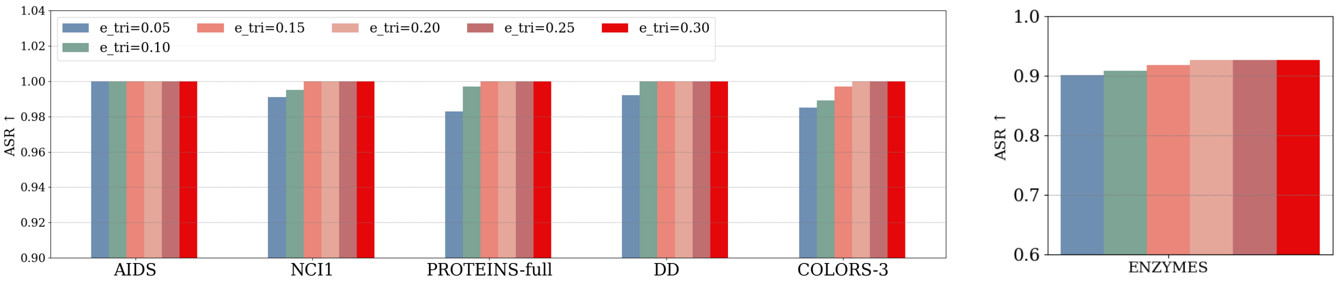

Figure 3: Attack results of NI-GDBA with the fraction of trigger edges e_{tri} ranging from 0.05 to 0.30

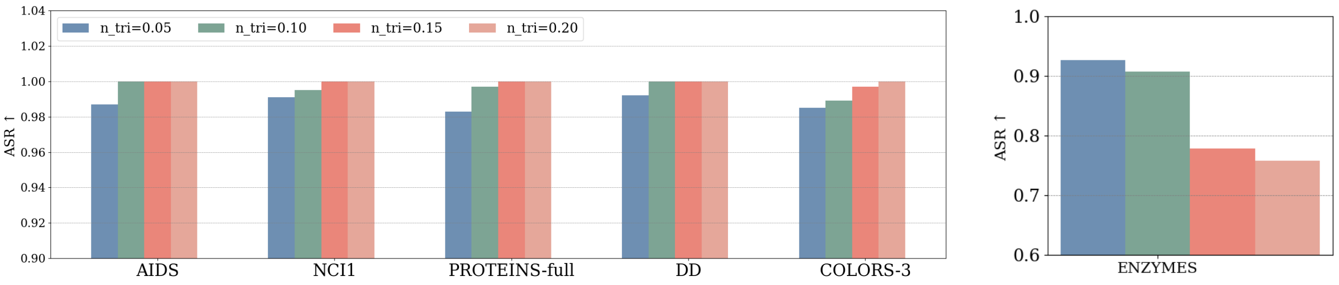

Figure 4: Attack results of NI-GDBA with the fraction of trigger nodes n_{tri} ranging from 0.05 to 0.20

unaffected by the parameter p, and we can conduct an effective attack with very little training data.

Impact of the trigger size e_{tri} and n_{tri}. Figure 3 and 4 respectively show the attack results when e_{tri} ranges from 0.05 to 0.3 and n_{tri} ranges from 0.05 to 0.2. For instance, on PROTEINS-full, when e_{tri} is from 0.05 to 0.30 with no defense, the ASR of NI-GDBA increases from 98.5% to 100%, and when n_{tri} is from 0.05 to 0.20, the ASR of NI-GDBA increases from 98.7% to 100%. This indicates that the parameters e_{tri} and n_{tri} have a limited impact on the ASR of our attack, which for NI-GDBA does increase with higher values of these parameters.

4.2.3 Ablation study. In this experiment, we examine the contribution of trigger position selecting module in our NI-GDBA. The results are in Table 7 with $C = 10$, $c^m = 3$, $n_{tri} = 0.1$, $e_{tri} = 0.2$ and the GCN model as the graph classifier. In Table 7, we compare our clustering-based trigger location selection algorithm, named Cluster, with a random number-based algorithm, named Random. According to our experimental results, the Cluster algorithm achieves a higher ASR when the ASR is less than 100%. Furthermore, when the ASR reaches 100%, our perturbation trigger generator model with the Cluster to select the trigger position, learns the natural backdoor of the GNN model more rapidly.

5 RELATED WORK

Backdoor attacks on centralized graph learning: Unlike non-graph data, which can be represented using Cartesian coordinates and have a fixed input size, graphs do not lend themselves to such representation and often vary in size, making it difficult to define a trigger. To address this challenge, two recent studies[32, 44] suggest using subgraphs as triggers. Zhang et al. [44] employ a random subgraph as the trigger, generated by random graph generation models such as Erdős-Rényi[10], Small World[29], and Preferential Attachment[1], and select nodes at random as the trigger's location. In contrast to using a random trigger shape, Xi et al. [32] developed a trigger generator that learns to create trigger shapes based on the edge and node feature information of each graph. However, similar to the previous approach, the trigger still randomly selects nodes as its location.

Backdoor attacks on federated graph learning: Xu et al. [37] is the first study to investigate backdoor attacks on FedGL. It draws inspiration from [33, 44] to realize DBA and CBA in FedGL, which uses a random subgraph as the attack trigger. Yang et al. [40] developed a trigger generator for each malicious client that learns to generate adaptive trigger for each graph, and then it injects the trigger into malicious clients' local training graph data before FedGL training process to embed the backdoor into the global GNN model.

6 CONCLUSION

We study the robustness of FedGL from the attacker's perspective. We design an effective, stealthy, persistent and non-intrusive distributed backdoor attack on FedGL. Instead of injecting trigger into malicious clients' local training data, we let all malicious clients provide unchanged graph data to gain a clean global GNN model, and then each malicious client try to learn the natural backdoor of the GNN model by an adaptive perturbation trigger generator model with their local graph data. Our attack results show that existing federated learning defense algorithms based on backdoor detection or removal are ineffective, and our attack can achieve effective attack with limited malicious clients, while the performance of the global GNN model will not deteriorate.

References

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

[2] Dawei Cheng, Fangzhou Yang, Sheng Xiang, and Jin Liu. 2022. Financial time series forecasting with multi-modality graph neural network. *Pattern Recognition* 121 (2022), 108218.

[3] Enyan Dai, Minhua Lin, Xiang Zhang, and Suhang Wang. 2023. Unnoticeable backdoor attacks on graph neural networks. In *Proceedings of the ACM Web Conference 2023*. 2263–2273.

[4] Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, and Phuong T Nguyen. 2021. A GNN-based recommender system to assist the specification of metamodels and models. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 70–81.

[5] P ERDdS and A R&wi. 1959. On random graphs I. *Publ. math. debrecen* 6, 290-297 (1959), 18.

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

[7] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2020. The limitations of federated learning in sybil settings. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*. 301–316.

[8] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. 2022. Graph neural networks for recommender system. In *Proceedings of the fifteenth ACM international conference on web search and data mining*. 1623–1625.

[9] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. 2020. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv preprint arXiv:2007.10760* (2020).

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

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

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

[13] Chaoyang He, Emir Ceyani, Keshav Balasubramanian, Murali Annavaram, and Salman Avestimehr. 2022. Spreadgnn: Decentralized multi-task federated learning for graph neural networks on molecular data. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 36. 6865–6873.

[14] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[15] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. 2022. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems* 35, 1 (2022), 5–22.

[16] Fan Liu, Siqi Lai, Yansong Ning, and Hao Liu. 2023. Bkd-FedGNN: A Benchmark for Classification Backdoor Attacks on Federated Graph Neural Network. *arXiv preprint arXiv:2306.10351* (2023).

[17] Rui Liu, Pengwei Xing, Zichao Deng, Anran Li, Cuntai Guan, and Han Yu. 2024. Federated graph neural networks: Overview, techniques, and challenges. *IEEE Transactions on Neural Networks and Learning Systems* (2024).

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

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

[20] Liang Peng, Nan Wang, Nicha Dvornek, Xiaofeng Zhu, and Xiaoxiao Li. 2022. Fedni: Federated graph learning with network inpainting for population-based disease prediction. *IEEE Transactions on Medical Imaging* 42, 7 (2022), 2032–2043.

[21] Bastian Pfeifer, Hryhorii Chereda, Roman Martin, Anna Saranti, Sandra Clemens, Anne-Christin Hauschild, Tim Beißbarth, Andreas Holzinger, and Dominik Heider. 2023. Ensemble-GNN: federated ensemble learning with graph neural networks for disease module discovery and classification. *Bioinformatics* 39, 11 (2023), btad703.

[22] Sina Sajadmanesh and Daniel Gatica-Perez. 2021. Locally private graph neural networks. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*. 2130–2145.

[23] Toyotaro Suzumura, Yi Zhou, Natahalie Baracaldo, Guangnan Ye, Keith Houck, Ryo Kawahara, Ali Anwar, Lucia Larise Stavarache, Yuji Watanabe, Pablo Loyola, et al. 2019. Towards federated graph learning for collaborative financial crimes detection. *arXiv preprint arXiv:1909.12946* (2019).

[24] Yue Tan, Yixin Liu, Guodong Long, Jing Jiang, Qinghua Lu, and Chengqi Zhang. 2023. Federated learning on non-iid graphs via structural knowledge sharing. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 9953–9961.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[26] Binghui Wang, Ang Li, Meng Pang, Hai Li, and Yiran Chen. 2022. Graphfl: A federated learning framework for semi-supervised node classification on graphs. In *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 498–507.

[27] Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. 2019. A semi-supervised graph attentive network for financial fraud detection. In *2019 IEEE international conference on data mining (ICDM)*. IEEE, 598–607.

[28] Zhen Wang, Weirui Kuang, Yuexiang Xie, Liuyi Yao, Yaliang Li, Bolin Ding, and Jingren Zhou. 2022. Federatedscope-gnn: Towards a unified, comprehensive and efficient package for federated graph learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4110–4120.

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

[30] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Tao Qi, Yongfeng Huang, and Xing Xie. 2022. A federated graph neural network framework for privacy-preserving personalization. *Nature Communications* 13, 1 (2022), 3091.

[31] Jiahao Wu, Ning Lu, Zeiyu Dai, Wenqi Fan, Shengcai Liu, Qing Li, and Ke Tang. 2024. Backdoor Graph Condensation. *arXiv preprint arXiv:2407.11025* (2024).

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

[33] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. 2019. Dba: Distributed backdoor attacks against federated learning. In *International conference on learning representations*.

[34] Jing Xu, Gorka Abad, and Stjepan Picek. 2023. Rethinking the trigger-injecting position in graph backdoor attack. In *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[35] Jing Xu, Stefanos Koffas, and Stjepan Picek. 2024. Unveiling the Threat: Investigating Distributed and Centralized Backdoor Attacks in Federated Graph Neural Networks. *Digital Threats: Research and Practice* 5, 2 (2024), 1–29.

[36] Jing Xu and Stjepan Picek. 2022. Poster: clean-label backdoor attack on graph neural networks. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3491–3493.

[37] Jing Xu, Rui Wang, Stefanos Koffas, Kaitai Liang, and Stjepan Picek. 2022. More is better (mostly): On the backdoor attacks in federated graph neural networks. In *Proceedings of the 38th Annual Computer Security Applications Conference*. 684–698.

[38] Shuiqiao Yang, Bao Gia Doan, Paul Montague, Olivier De Vel, Tamas Abraham, Seyit Camtepe, Damith C Ranasinghe, and Salil S Kanhere. 2022. Transferable graph backdoor attack. In *Proceedings of the 25th international symposium on research in attacks, intrusions and defenses*. 321–332.

[39] Xiao Yang, Gaolei Li, and Jianhua Li. 2024. Graph Neural Backdoor: Fundamentals, Methodologies, Applications, and Future Directions. *arXiv preprint arXiv:2406.10573* (2024).

[40] Yuxin Yang, Qiang Li, Jinyuan Jia, Yuan Hong, and Binghui Wang. 2024. Distributed Backdoor Attacks on Federated Graph Learning and Certified Defenses. *arXiv preprint arXiv:2407.08935* (2024).

[41] Xiaoyan Yin, Wanyu Lin, Kexin Sun, Chun Wei, and Yanjiao Chen. 2022. A 2 S 2-GNN: Rigging GNN-based social status by adversarial attacks in signed social networks. *IEEE Transactions on Information Forensics and Security* 18 (2022), 206–220.

[42] Hangfan Zhang, Jinghui Chen, Lu Lin, Jinyuan Jia, and Dinghao Wu. 2023. Graph contrastive backdoor attacks. In *International Conference on Machine Learning*. PMLR, 40888–40910.

[43] Jiale Zhang, Chengcheng Zhu, Bosen Rao, Hao Sui, Xiaobing Sun, Bing Chen, Chunyi Zhou, and Shouling Ji. 2024. " No Matter What You Do!": Mitigating Backdoor Attacks in Graph Neural Networks. *arXiv preprint arXiv:2410.01272* (2024).

[44] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. 2021. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*. 15–26.

[45] Zhiwei Zhang, Minhua Lin, Enyan Dai, and Suhang Wang. 2024. Rethinking graph backdoor attacks: A distribution-preserving perspective. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4386–4397.

[46] Wei Zhu, Jiebo Luo, and Andrew D White. 2022. Federated learning of molecular properties with graph neural networks in a heterogeneous setting. *Patterns* 3, 6 (2022).

[47] Yuxuan Zhu, Michael Mandulak, Kerui Wu, George Slota, Yuseok Jeon, Ka-Ho Chow, and Lei Yu. 2024. On the Robustness of Graph Reduction Against GNN Backdoor. *arXiv preprint arXiv:2407.02431* (2024).

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009