Distributed Backdoor Attacks on Federated Graph Learning and Certified Defenses

Yuxin Yang
yuxiny22@mails.jlu.edu.cn
College of Computer Science and Technology, Jilin University
Changchun, Jilin, China
Department of Computer Science, Illinois Institute of Technology
Chicago, Illinois, USA

Qiang Li
li_qiang@jlu.edu.cn
College of Computer Science and Technology, Jilin University
Changchun, Jilin, China

Jinyuan Jia
jinyuan@psu.edu
College of Information Sciences and Technology,
The Pennsylvania State University
University Park, Pennsylvania, USA

Yuan Hong
yuan.hong@uconn.edu
School of Computing,
University of Connecticut
Storrs, Connecticut, USA

Binghui Wang
bwang70@iit.edu
Department of Computer Science,
Illinois Institute of Technology
Chicago, Illinois, USA

Abstract

Federated graph learning (FedGL) is an emerging federated learning (FL) framework that extends FL to learn graph data from diverse sources without accessing the data. FL for non-graph data has shown to be vulnerable to backdoor attacks, which inject a shared backdoor trigger into the training data such that the trained backdoored FL model can predict the testing data containing the trigger as the attacker desires. However, FedGL against backdoor attacks is largely unexplored, and no effective defense exists.

In this paper, we aim to address such significant deficiency. First, we propose an effective, stealthy, and persistent backdoor attack on FedGL. Our attack uses a subgraph as the trigger and designs an adaptive trigger generator that can derive the effective trigger location and shape for each graph. Our attack shows that empirical defenses are hard to detect/remove our generated triggers. To mitigate it, we further develop a certified defense for any backdoored FedGL model against the trigger with any shape at any location. Our defense involves carefully dividing a testing graph into multiple subgraphs and designing a majority vote-based ensemble classifier on these subgraphs. We then derive the deterministic certified robustness based on the ensemble classifier and prove its tightness. We extensively evaluate our attack and defense on six graph datasets. Our attack results show our attack can obtain > 90% backdoor accuracy in almost all datasets. Our defense results show, in certain cases, the certified accuracy for clean testing graphs against an arbitrary trigger with size 20 can be close to the normal accuracy under no attack, while there is a moderate gap in other cases. Source code is available at: https://github.com/Yuxin104/Opt-GDBA. The full report is at: https://arxiv.org/abs/2407.08935.

CCS Concepts

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

Keywords

Federated Graph Learning, Backdoor Attacks, Certified Defenses

ACM Reference Format:
Yuxin Yang, Qiang Li, Jinyuan Jia, Yuan Hong, and Binghui Wang. 2024. Distributed Backdoor Attacks on Federated Graph Learning and Certified Defenses. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3690187

1 Introduction

Graph is a pervasive data type consisting of nodes and edges, where nodes represent entities and edges represent relationships among entities. Learning on graph data (or *graph learning*) has gained great attention in both academia [13, 25, 32, 54, 85] and industry [29, 61, 90, 98] in the past several years. A particular task, i.e., graph classification, predicting the label of a graph has applications in a wide variety of domains including healthcare, bioinformatics, transportation, financial services, to name a few [37, 75, 97].

Despite notable advancements in graph learning, most require the consolidation of graph data from various sources into a single machine. With the increasing importance on data privacy [19], this requirement becomes infeasible. For instance, a third-party service provider trains a graph learning model for a bunch of financial institutions to help detect anomalous customers. Each institution has its own graph dataset of customers, where each graph can be a customer's transaction records with other customers, and each customer also has personal information. Due to the business competition and rigorous privacy policies, each institution's customer data cannot be shared with other institutions or the service provider.

Binghui Wang is the corresponding author.

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690187

Federated Learning (FL) [41], a new distributed learning paradigm, aims to address the data isolation/privacy issue. Specifically, FL enables a central server coordinating multiple clients to collaboratively train a machine learning model without the need of sharing clients' data. Federated graph learning (FedGL) generalizes graph learning in the FL setting and has attracted increasing attention recently [1, 26, 27, 47, 52, 58, 65, 69, 74, 81, 82, 91] with various successful applications such as disease prediction [47], molecular classification [27], and recommendation [1, 74]. In FedGL for graph classification, each client owns a set of graphs, and the server and the participating clients collaboratively learn a shared graph classifier without accessing the clients' graphs. The learnt shared graph classifier is then used by all clients for testing.

However, recent works show that FL for *non-graph data* (e.g., images, videos) is vulnerable to backdoor attacks [2, 20, 21, 51, 67, 79, 95]. In backdoor attacks on FL, a fraction of malicious clients is controlled by an attacker. The malicious clients inject a backdoor trigger (e.g., a sticker) into part of their training data (e.g., images) and flag these backdoored training data with an attacker-chosen *target label* (different from their true label). The clients' backdoored data and clean data are used for FL training, such that the trained backdoored FL model will predict malicious clients' testing data with the trigger as the target label, while those without the trigger still as the true label. While backdoor attacks on FL for non-graph data is widely studied, those for graph data is underexplored. Note that backdoor attacks on FedGL would cause serious issues for safety/security-critical applications. For instance, Alibaba and Amazon have deployed and open-sourced their FedGL framework (FederatedScope-GNN [69] and FedML-GNN [14]). When these FedGL packages are used for disease prediction [47] but backdoored, the patients' safety could be jeopardized.

In this paper, we aim to design effective backdoor attacks on FedGL, as well as effective defense to mitigate the backdoor attack.
Challenges in designing effective backdoors on FedGL: Comparing with non-graph data, designing effective backdoors on graph data used by FedGL faces unique challenges: 1) Backdoor attacks on non-graph data (e.g., images) require *same* input size, while graph data have varying sizes (in terms of number of nodes and edges); 2) Backdoor attacks on non-graph data can leverage shared property (e.g., important pixels in images with the same label are spatially-close), while graph-data do not have such property: even graphs have the same label, their locations of crucial nodes can be significantly different (see Figure 3); 3) Graph backdoors solely based on node features (like pixels in images) is not effective enough. Edge information is equally important and should be considered.

We notice a recent work [84] proposed a *random* backdoor attack on FedGL inspired by [79, 94]. Specifically, it uses a *subgraph* as a trigger, and each malicious client *randomly* generates the trigger shape and *randomly* picks nodes from local graphs as the location to inject the trigger. However, our results show this attack attains unsatisfactory backdoor performance (see Table 1).
Our optimized distributed backdoor attacks on FedGL: We observe the ineffectiveness of the existing backdoor attack on FedGL is primarily due to the random nature of the trigger, i.e., it does not use any graph or client information unique to FedGL. An effective backdoor attack on FedGL should design the trigger by explicitly considering the individual graph and client information. To bridge the gap, we propose an optimized DBA on FedGL (termed Opt-GDBA). As a trigger consists of trigger location, size, and shape, our Opt-GDBA hence designs an adaptive trigger generator that *adaptively optimizes the location and shape of the subgraph trigger and learns a local trigger for each graph using the graph and client information*. Specifically, the trigger generator consists of three modules: 1) the first module obtains nodes' importance scores by leveraging both the edge and node feature information in a given input graph; 2) the second module learns the trigger location based on the nodes' importance scores. In particular, we design two trigger location learning schemes, i.e., Definable-Trigger and Customized-Trigger, where the first scheme predefines the trigger node size and the second one automatically identifies the important nodes in the graph as the trigger nodes; 3) given the trigger location, the third module further learns the trigger shape (i.e., determines the trigger's node features and edges) via introducing edge/node attention and local trigger differentiation mechanisms. By incorporating our adaptive trigger generator into the backdoored FedGL training, the generated backdoored graphs can be more stealthy and diverse, and make the backdoor attack much more effective and persistent.
Challenges in designing effective defenses on backdoored FedGL: Once a backdoored FedGL model is trained, we test empirical defenses, e.g., based on backdoor detection or backdoor removal, are hard to mitigate the backdoored effect induced by our Opt-GDBA (see Tables 3 and 4). Moreover, empirical defenses can be often broken by adaptive attacks [67]. Hence, we focus on certified defenses with provable guarantees. Particularly, we expect the defense can *i) provably predict the correct label for clean testing graphs injected with an arbitrary trigger (shape and location) with a bounded size;* **and** *ii) provably predict a non-target label for backdoored testing graphs, both with probability 100%.* However, it is extremely challenging to design such a certified defense due to: *1) the size of testing graphs varies; 2) should not rely on a specific model; 3) a trigger can arbitrary perturb any edges and nodes in a testing graph;* and *4) a deterministic guarantee*. Note that certified defenses for non-graph data [63, 71] require same size inputs, which inherently cannot be applied to graph data. Existing certified defenses for graph data [5, 64, 94] are also insufficient: they are either against node feature or edge perturbation, but not both; their robustness guarantee is for a fixed input size or specific model, or incorrect with a certain probability.
Our certified defense against backdoored FedGL: We design an effective majority-voting based certified defense to address all above limitations. Majority-voting is a generic ensemble method [12], and different methods develop the respective voter for their own purpose (see more details in Section 7). Our tailored majority-vote based defense includes three critical steps. First, we carefully divide a (clean or backdoored) testing graph into multiple subgraphs such that the graph division is deterministic, and for any pair of subgraphs, their nodes and edges are non-overlapped. Second, we build a majority vote-based ensemble graph classifier for predicting these subgraphs and each prediction on a subgraph is treated as a vote. This classifier ensures a bounded number of subgraphs' predictions be different after injecting the trigger and the expectations i) and ii) be satisfied (per Theorems 1 & 3). Third, we derive the deterministic robustness guarantee of the ensemble classifier against a (bounded size) trigger with arbitrary edge and node (feature) perturbations. We also prove that our certified defense is tight.

Empirical and theoretical evaluations: We extensively evaluate our Opt-GDBA attack and certified defense on six benchmark graph datasets. Our attack results show that: 1) Compared with the existing work [84], Opt-GDBA has a gain from 30% to 46% on the backdoor performance, and generates triggers with less number of nodes or/and edges; 2) The Customized-Trigger scheme is more stealthy than the Definable-Trigger scheme, indicating it uncovers more important nodes in the trigger; 3) Our generated backdoored graphs are persistent and hard to be detected or removed.

We further test our defense on the backdoored FedGL trained with our attack. Our defense results show that: 1) In some cases, the certified main accuracy against a trigger arbitrarily perturbing 20 nodes/edges in total can be close to the accuracy without attack; 2) The certified backdoor accuracy in all datasets is 0, which indicates the backdoored testing graphs generated by our Opt-GDBA are completely broken by our defense.

Contributions: We summarize our main contributions as below:

- We propose Opt-GDBA, an optimized DBA to FedGL, that is effective, stealthy, and persistent.
- We develop a certified defense applicable for any (backdoored) FedGL against any graph structure and node feature perturbation. Moreover, our robustness guarantee is deterministic and tight.
- Our extensive empirical and theoretical evaluations verify the effectiveness of our proposed attack and defense.

2 Background and Problem Definition

2.1 Federated Graph Learning (FedGL)

We denote $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ as a graph where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the node feature matrix, with d the number of features and $|\mathcal{V}|$ the total number of nodes. We let $\mathbf{A} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ be the adjacency matrix with $A_{u,v} = 1$, if $(u,v) \in \mathcal{E}$, and 0, otherwise. $\mathbf{A}$ hence contains all edge information in G. We consider graph classification as the task of interest, where each graph G has a label y from a label set $\mathcal{Y}$. Graph learning (GL) takes a graph G as input and learns a graph classifier, denoted as f, that outputs an estimated graph label, i.e., $f : G \longrightarrow \mathcal{Y}$.

FedGL extends GL in the FL setting. Assume C clients $\mathcal{C} = \{1, 2, \cdots, C\}$ and a server participating in FedGL. Each client i has a set of labeled training graphs $\mathcal{G}^i = \{(G_1^i, y_1^i), \cdots, (G_{|\mathcal{G}^i|}^i, y_{|\mathcal{G}^i|}^i)\}$. In a t-th round, the server randomly selects a subset of clients $\mathcal{C}_t \subset \mathcal{C}$ and broadcasts the current global model θ_t on the server to $\mathcal{C}_t$. A client $i \in \mathcal{C}_t$ updates its local model $\theta_t^i = \partial_{\theta_t} L(\mathcal{G}^i; \theta_t)$ using its training graphs $\mathcal{G}^i$ and the shared θ_t, and submits θ_t^i to the server. Here $L(\mathcal{G}^i; \theta^t)$ is a loss function used by the client i, e.g., cross-entropy loss. The server then aggregates the $\mathcal{C}_t$ clients' models $\{\theta_t^i\}_{i \in \mathcal{C}_t}$ to learn the global model θ_{t+1} for the next iteration using some aggregation algorithm. For instance, when using the common average aggregation [41, 65], $\theta_{t+1} = \frac{1}{|\mathcal{C}_t|} \sum_{i \in \mathcal{C}_t} \theta_t^i$. Next, the server randomly selects a new subset of clients $\mathcal{C}_{t+1} \subset \mathcal{C}$ and broadcasts θ_{t+1} to them. This process is repeated until the global model converges or reaching the maximal iterations. The final global model is shared with clients for their task, e.g., classify their testing graphs.

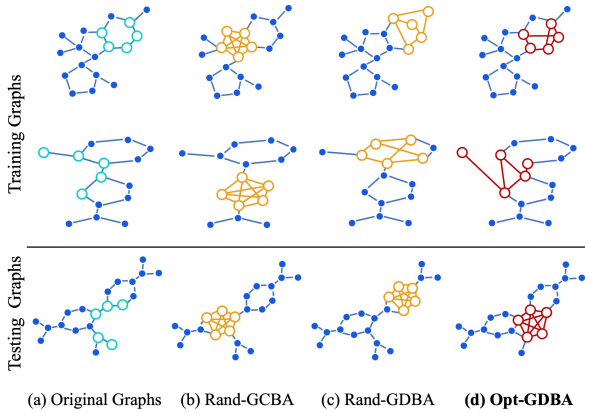

Figure 1: Comparing the triggers of the backdoor attacks on FedGL: (b) Rand-GCBA, (c) Rand-GDBA, and (d) our Opt-GDBA. Opt-GDBA strategically selects critical nodes and their connected edges in individual graphs, resulting in more effective local triggers and the combined global trigger.

2.2 Backdoor Attacks on FedGL

In backdoor attacks on FedGL, malicious clients inject a subgraph trigger (consisting of edges and nodes with features) into part of their training graphs and set backdoored graphs with a *target label*. Depending on how the trigger is designed, a recent work [84] proposed two attacks: *centralized* backdoor attack (CBA) inspired by [94], and *distributed* backdoor attack (DBA) inspired by [79]. We denote the two attacks as *Rand-GCBA* and *Rand-GDBA*, respectively, where the prefix "Rand" means malicious clients *randomly* generate the shape of the trigger and *randomly* choose nodes from their clean graphs as the location to inject the trigger.

Rand-GCBA: All malicious clients use a *shared* trigger κ. In each to-be-backdoored graph, malicious clients randomly sample a subset of nodes from the graph as the trigger location and replace the connections of these nodes with the trigger κ. Then each malicious client i iteratively learns its local backdoored model θ_B^i as below:

$$\theta_B^i = \arg\min_{\theta_B^i} L(\mathcal{G}_B^i \cup \mathcal{G}_C^i; \theta), \tag{1}$$

where $\mathcal{G}_B^i = \{R(G_j^i, \kappa), y_B\}$ is a set of backdoored graphs, $R(G_j^i, \kappa)$ is function that generates a backdoored graph of G_j^i by attaching the trigger κ, and y_B denotes the target label. θ is the global model. $\mathcal{G}_C^i$ contains the remaining clean graphs in $\mathcal{G}^i$, and $|\mathcal{G}_B^i| + |\mathcal{G}_C^i| = |\mathcal{G}^i|$. The server will aggregate the local models of chosen malicious clients and normally trained benign clients. The final backdoored graph classifier, denoted as f_B, is shared with all clients. During testing, malicious clients will use the same κ for their testing graphs, but the trigger location is randomly chosen.

Rand-GDBA: Each malicious client i has its own local trigger κ^i (often sparser/smaller than κ), and injects κ^i into a fraction of its training graphs $\mathcal{G}^i$, where the trigger location is randomly chosen. Then each malicious client generates its backdoored graphs $\mathcal{G}_B^i = \{R(G_j^i, \kappa^i), y_B\}$ for training (i.e., minimizing the loss in Equation (1)). During testing, all malicious clients' triggers $\{\kappa^i\}$ will be combined into a single one. The combined trigger, with a random location, will be injected into testing graphs.

Rand-GDBA is shown to be more effective than Rand-GCBA [84]. Figure 1 shows example shared trigger in Rand-GCBA, local triggers in Rand-GDBA across clients, and triggers in our attack.

2.3 Threat Model

We aim to understand the robustness of FedGL from both the attacker's and defender's perspective. As an attacker, we expect to design an effective and stealthy DBA to FedGL during training. As a defender, in contrast, we expect to design an effective certified defense against the worst-case DBA on a backdoored FedGL model.
Attacker: We assume the attacker manipulates a fraction (say ρ) of the total C clients, namely malicious clients.

- **Attacker's knowledge:** All malicious clients only know their own training graphs and the shared global model in the whole of (backdoored) FedGL training.
- **Attacker's capability:** Malicious clients can inject a subgraph trigger into any location/part of their training graphs during training. To ensure effectiveness and stealthiness for the attack, we follow [79, 84] to inject the trigger in every training iteration, but its size (w.r.t. number of nodes or/and edges) is small.
- **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 backdoor accuracy* as well as *a high main task accuracy*.

Defender: The defender aims to build a certifiably robust defense, under which the learnt backdoored FedGL can achieve two goals.

- **High certified main task accuracy:** provably predict correct labels as many as possible for clean testing graphs against arbitrary trigger (any shape and location) with a bounded size.
- **Low certified backdoor accuracy:** provably predict the target label as few as possible for backdoored testing graphs (that are generated by our Opt-GDBA).

3 Optimized DBAs on FedGL

Recall the existing DBA to FedGL generates triggers with *random* locations and *random* shape, and obtains unsatisfactory backdoor performance. We propose an optimized DBA on FedGL (called Opt-GDBA) to address the limitation. Our Opt-GDBA designs an adaptive trigger generator to adaptively optimize the trigger location and shape by integrating the edge and node feature information in individual graphs. See Figure 2 for the pipeline of Opt-GDBA.

3.1 Adaptive Trigger Generator

The proposed adaptive trigger generator consists of three modules: *1) node importance score learning, 2) trigger location learning, and 3) trigger shape learning*. For simplicity, we use a client i's graph $G^i = (\mathcal{V}^i, \mathcal{E}^i, \mathbf{X}^i)$ with the adjacency matrix $\mathbf{A}^i$ for illustration. We first obtain the nodes' importance scores using module 1). We next input the nodes' scores to module 2) to decide the trigger location with a predefined or customized trigger size. Finally, module 3) learns the trigger shape and generates the backdoored graph G_B^i for G^i together with module 2). ***Detailed architecture of all the described networks below are in the full report.***

Algorithm 1 k-means_gap to learn customized trigger size

Input: Nodes' scores $\mathbf{s}^i$, maximum trigger size n^*_{tri}.
Output: Important nodes $\mathcal{V}^i_{cus}$.

$\mathbf{s}^i = \mathbf{s}^i / \|\mathbf{s}^i\|_1$
for $k = 1, 2, ..., K$ **do**
  $C_i, \mu_i = k$-means $(\mathbf{s}^i)$
  $V_k = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_j\|^2$
  for $b = 1, 2, ..., B$ **do**
    $x_{j,b}$ = sample($|\mathbf{s}^i|$) in $[0, 1]$
    $C_{i,b}, \mu_{i,b} = k$-means $(x_{j,b})$
    $V^*_{kb} = \sum_{i=1}^{k} \sum_{x_{jb} \in C_{ib}} \|x_{jb} - \mu_{ib}\|^2$
  end for
  $Gap(k) = \frac{1}{B} \sum_{i=1}^{B} \log(V^*_{kb}) - \log(V_k)$
  $v' = \frac{1}{B} \sum_{i=1}^{B} \log(V^*_{kb})$
  $sd(k) = (\frac{1}{B} \sum_{i=1}^{B} (\log(V^*_{kb}) - v')^2)^{\frac{1}{2}}$
  $s'_k = \sqrt{\frac{1+B}{B}} sd(k)$
end for
$\hat{k} = min(k)\ s.t.\ Gap(k) - Gap(k+1) + s'_{k+1} \geq 0$
$C_i, \mu_i = \hat{k}$-means$(\mathbf{s}^i)$
$\mathcal{V}^i_{cus} = \{C_i | \max\{\text{Avg}(C_1), \text{Avg}(C_2), ..., \text{Avg}(C_{\hat{k}})\}\}$
if $|\mathcal{V}^i_{cus}| > msize$ **then**
  $\mathcal{V}^i_{cus} = \text{sort}(\mathcal{V}^i_{cus})$
  $\mathcal{V}^i_{cus} = \mathcal{V}^i_{cus}[: msize]$
end if

1) Node importance score learning. The goal is to measure the node importance so that the trigger can be placed on the important nodes. Specifically, we leverage both the edges and node features in G^i to decide the node importance.

First, we define two networks: EdgeView$(\cdot)$ and NodeView$(\cdot)$. EdgeView$(\cdot)$ characterizes the node importance from the edge view, and the extracted node importance scores from $\mathbf{A}^i$ are denoted by $\mathbf{e}^i \in \mathbb{R}^{|\mathcal{V}^i|}$. Instead, NodeView$(\cdot)$ characterizes the node importance from the node feature view, and the extracted node importance scores from $\mathbf{X}^i$ are denoted by $\mathbf{n}^i \in \mathbb{R}^{|\mathcal{V}^i|}$. Formally,

$$\mathbf{e}^i = \text{EdgeView}(\mathbf{A}^i), \quad \mathbf{n}^i = \text{NodeView}(\mathbf{X}^i), \tag{2}$$

where $\mathbf{e}^i$ and $\mathbf{n}^i$ are constrained to have a value range $(0, 1)$.

We then calculate the nodes' importance scores, denoted by $\mathbf{s}^i$, as the element-wise product $\odot$ of vectors $\mathbf{e}^i$ and $\mathbf{n}^i$ as below:

$$\mathbf{s}^i = \mathbf{e}^i \odot \mathbf{n}^i. \tag{3}$$

2) Trigger location learning. With nodes' importance scores, we design two schemes to decide the trigger location in each graph: *Definable-Trigger* and *Customized-Trigger*.

Definable-Trigger: It *predefines* a trigger node size n_{tri} used by all backdoored graphs. Specifically, this scheme first ranks $\mathbf{s}^i$ in a descending order and selects the nodes $\mathcal{V}^i_{def}$ from G^i with the top n_{tri} values as the trigger location.

Customized-Trigger: One drawback of Definable-Trigger is that all backdoored graphs use the same trigger size, but the graph size varies in practice. This would cause $\mathcal{V}^i_{def}$ misses important nodes if G^i is a large graph but n_{tri} is small, or includes non-important nodes if G^i is a small graph but n_{tri} is large. To address it, we further develop the Customized-Trigger scheme, which automatically *learns* the best local trigger size of each graph during the FedGL training. The learnt most important nodes for G^i are stored in $\mathcal{V}^i_{cus}$.

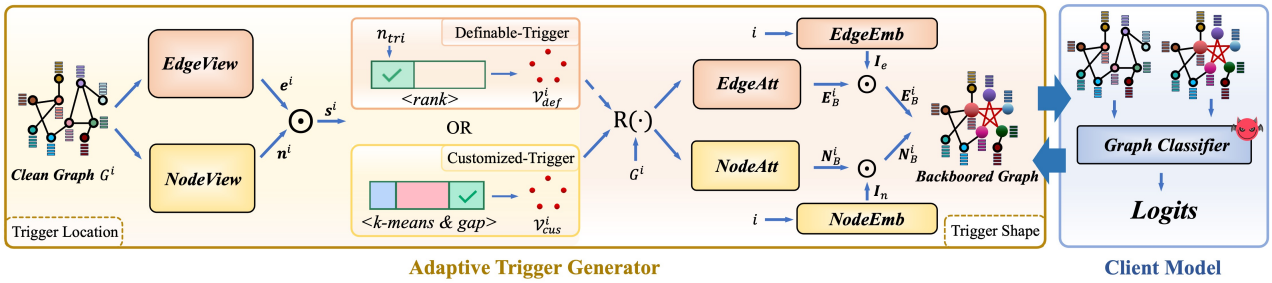

Figure 2: Pipeline of our proposed Opt-GDBA on FedGL (a client i perspective).

Our main idea is to adopt the Gap statistics [60] based on k-means clustering[1]. The algorithm details are shown in Algorithm 1. At a high-level, given the node scores $\mathbf{s}^i$, we first use the Gap statistics to estimate the number of clusters $\hat{k}$. Then we employ k-means to divide the nodes into $\hat{k}$ clusters based on their scores $\mathbf{s}^i$. Finally, the nodes in the cluster with the largest average score are treated as the most important nodes $\mathcal{V}^i_{cus}$, whose positions are put the local trigger. *Note that, to ensure the stealthiness of the attack, we require the trigger size not exceed a threshold (e.g., n^*_{tri}).*

3) Trigger shape learning. Given the location of the local trigger ($\mathcal{V}^i_{def}$ or $\mathcal{V}^i_{cus}$), we learn the trigger shape through two sub-modules: *edge/node attention*, and *local trigger differentiation*. The former determines the edges and node features in the trigger, which is inspired by [76], while the latter promotes divergence of different local triggers so that the attack effectiveness can be enhanced when these local triggers are combined for backdoored testing. We denote $G^i_B = (\mathcal{V}^i, \mathcal{E}^i_B, \mathbf{X}^i_B)$ as an initial backdoored graph with an empty trigger shape and the corresponding adjacency matrix $\mathbf{A}^i_B$. For brevity, we use $\mathcal{V}^i_{def}$ for illustration.

Edge/node attention. We introduce two attention networks: *edge attention network* EdgeAtt$(\cdot)$ that focuses on understanding the connectivity between nodes $\mathcal{V}^i_{def}$ in the trigger, and *node attention network* NodeAtt$(\cdot)$ that aims to improve the flexibility of the trigger by also incorporating node features. We denote the trigger's edge attention matrix as $\mathbf{E}^i_{tri} \in \mathbb{R}^{|\mathcal{V}^i_{def}|\times|\mathcal{V}^i_{def}|}$, and trigger's node feature attention matrix as $\mathbf{N}^i_{tri} \in \mathbb{R}^{|\mathcal{V}^i_{def}|\times d}$. Formally,

$$\mathbf{E}^i_{tri} = \text{EdgeAtt}(\mathbf{A}^i_B, \mathcal{V}^i_{\text{def}}); \tag{4}$$

$$\mathbf{N}^i_{tri} = \text{NodeAtt}(\mathbf{X}^i_B, \mathcal{V}^i_{\text{def}}). \tag{5}$$

Local trigger differentiation. To further enable distinct malicious clients to possess personalized and controllable local triggers, we also propose to incorporate the client index i into the trigger shape generation. Specifically, we first use an edge embedding function EdgeEmb$(\cdot)$ to convert the client index i into $\mathbf{I}_e \in \mathbb{R}^{|\mathcal{V}^i_{def}|\times|\mathcal{V}^i_{def}|}$ and a node embedding function NodeEmb$(\cdot)$ to convert it into $\mathbf{I}_n \in \mathbb{R}^{|\mathcal{V}^i_{def}|\times d}$. We then multiply $\mathbf{I}_e$ (and $\mathbf{I}_n$) with the attention matrix $\mathbf{E}^i_{tri}$ (and $\mathbf{N}^i_{tri}$) to integrate the unique information of the client index. The equations can be expressed as follows:

$$\begin{aligned} \mathbf{I}_e &= \text{EdgeEmb}(i), \quad \mathbf{E}^i_{tri} = \mathbf{E}^i_{tri} \odot \mathbf{I}_e, \\ \mathbf{I}_n &= \text{NodeEmb}(i), \quad \mathbf{N}^i_{tri} = \mathbf{N}^i_{tri} \odot \mathbf{I}_n. \end{aligned} \tag{6}$$

To further discretize the connectivity status between nodes, we convert the continuous edge attention matrix $\mathbf{E}^i_{tri}$ to be binary, i.e., $\mathbf{E}^i_{tri} = \mathbb{1}(\mathbf{E}^i_{tri} \geq 0.5)$, where $\mathbb{1}(p)$ is an indicator function that returns 1 if p is true, and 0 otherwise.

The trigger location $\mathcal{V}^i_{\text{def}}$ as well as trigger shape $\mathbf{E}^i_{tri}, \mathbf{N}^i_{tri}$ decides the optimized trigger, which we denote as $\tilde{\kappa}^i = (\mathcal{V}^i_{\text{def}}, \mathbf{E}^i_{tri}, \mathbf{N}^i_{tri})$. The optimized backdoored graph for a graph G^i is then generated by $\tilde{G}^i_B = R(G^i, \tilde{\kappa}^i)$. Algorithm 2 summarizes the adaptive trigger generator for generating a backdoored graph.

Algorithm 2 Adaptive Trigger Generator

Input: A clean graph G^i, trigger node size n_{tri} or n^*_{tri}.
Output: Backdoored graph $\tilde{G}^i_B$.

```
s^i = Node_score(G^i) // Node importance score
if Definable-Trigger then
    V^i_def = rank(s^i, n_tri) // Trigger location
    G̃^i_B = Trigger_shape_learning(G^i, V^i_def)
else if Customized-Trigger then
    V^i_cus = k-means_gap(s^i, msize) // Trigger location
    G̃^i_B = Trigger_shape_learning(G^i, V^i_cus)
end if
```

3.2 FedGL Training with Optimized Backdoor

We now show the entire backdoored FedGL training with the optimized backdoored graphs (algorithm details are in Algorithm 3). It involves alternatively and iteratively training the (backdoored) local model, optimizing the adaptive trigger generator, and updating the shared global backdoored model.

Training the (backdoored) local model: We denote the optimized backdoored graphs in each malicious client i as $\tilde{\mathcal{G}}^i_B = \{R(G^i_j, \tilde{\kappa}^i_j), y_B\}$. Then each malicious client i trains its local backdoored model θ^i_B via minimizing the loss on both the optimized backdoored graphs $\tilde{\mathcal{G}}^i_B$ and clean graphs $\mathcal{G}^i_C$:

$$\theta^i_B = \arg\min_{\theta^i_B} L(\tilde{\mathcal{G}}^i_B \cup \mathcal{G}^i_C; \theta). \tag{7}$$

For each benign client j, it updates the local model via minimizing the loss on all its clean graphs $\mathcal{G}^j$ as:

$$\theta^j = \arg\min_{\theta^j} L(\tilde{\mathcal{G}}^j; \theta). \tag{8}$$

[1]K-mean is a widely-adopted clustering algorithm that is efficient and effective. By integrating with gap statistics, K-means can also efficiently determine the optimal number of clusters. We admit there are other more advanced/complicated/effective clustering algorithms. Note that our purpose is not to pick the best clustering algorithm, but the one that is suitable to achieve our goal, i.e., learning the local trigger size.

Algorithm 3 Backdoored FedGL training with Opt-GDBA

Input: Total clients C with clean graphs $\{\mathcal{G}^i\}_{i\in C}$, malicious clients $\tilde{C}$, training iterations $iter$, initial global model θ_1, malicious clients' initial generator model $\{\omega_1^i\}_{i\in\tilde{C}}$.
Output: Backdoored global model θ_{iter}.

for each iteration t in [1,iter] **do**
  for each client $i \in C$ **do**
    if $i \in \tilde{C}$ **then**:
      Client i divides $\mathcal{G}^i$ into $\mathcal{G}_C^i$ and to-be-backdoored $\mathcal{G}_o^i$
      $\tilde{\mathcal{G}}_B^i$ = Generator$(\mathcal{G}_o^i; \omega_t^i)$ using Algorithm 2
      $\theta_t^i = \arg\min_{\theta^i} L(\tilde{\mathcal{G}}_B^i \cup \mathcal{G}_C^i; \theta_t)$
      $\omega_t^i = \arg\min_{\omega^i} L(\tilde{\mathcal{G}}_B^i; \theta_t^i)$
    else
      $\theta_t^i = \arg\min_{\theta^i} L(\mathcal{G}^i; \theta_t)$
    end if
  end for
  Server randomly selects C_t clients for aggregation
  $\theta_{t+1} = \frac{1}{|C_t|}\sum_{i\in C_t}\theta_t^i$
end for

Optimizing the adaptive trigger generator: We denote the parameters of the trigger generator per malicious client i as ω^i, which includes the parameters of all networks in Section 3.1. Each malicious client i also optimizes its generator ω^i to ensure the generated backdoored graphs be more effective and diverse. Specifically,

$$\omega^i = \arg\min_{\omega^i} L(\tilde{\mathcal{G}}_B^i; \theta_B^i). \tag{9}$$

Updating the shared global model: The server averages the backdoored local models $\{\theta_B^i\}$ and benign local models $\{\theta^j\}$ of the selected clients to update the global model θ.

4 Attack Results

4.1 Experimental Setup

Datasets and training/testing sets: We evaluate our attack on six benchmark real-world graph datasets for graph classification. ***Dataset description and statistics and training/testing sets about the datasets are presented in Table 8 in the full report.***
Attack baselines: We compare our Opt-GDBA with Rand-GCBA and Rand-GDBA (details are in Section 2.2). Their main difference lies in the way to inject the trigger.

- **Rand-GCBA [84]:** All malicious clients use a shared trigger with the same shape but random location. To force the trigger yields the most effective attack, we assume it be a *complete subgraph*.
- **Rand-GDBA [84]:** Each malicious client generates its local trigger. Following [94], each client generates the trigger using the Erdős-Rényi (ER) random graph model [18], where the number of edges e_{tri} with a trigger node size n_{tri} can be controlled. These triggers are then attached to random nodes in the to-be-backdoored graphs. To further enhance this attack, we also maintain the diversity of local triggers among malicious clients. To do so, we store a set of generated local triggers via the ER model, and assign different triggers to different malicious clients.
  For fair comparison, we make sure the total number of edges in all triggers in Rand-GCBA and Rand-GDBA are same. This can be realized by forcing $\rho^c * e_{tri}^c = \rho^d * e_{tri}^d$, where ρ^c and ρ^d are the ratio of malicious clients, and e_{tri}^c and e_{tri}^d are the number of trigger edge in Rand-GCBA and Rand-GDBA, respectively.
- **Our Opt-GDBA:** Each to-be-backdoored training graph generates an individual trigger via the proposed adaptive trigger generator, where the trigger location and shape are learnt. Note that the trigger node size n_{tri} is predefined in the Definable-Trigger scheme (same as Rand-GCBA and Rand-GDBA), but is automatically learnt in the Customized-Trigger scheme.

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 Opt-GDBA, for each testing graph, we learn the important nodes that determine the trigger location, and then generate the complete graph based on them. In contrast, the trigger location of Rand-GCBA and Rand-GDBA in testing graphs is random. *Note that [84] uses a different way to combine local triggers and injects a much larger global trigger in Rand-GDBA.* ***The discussion and results are shown in Table 10 in the report***.
Parameter setting: During FedGL training, we use a total of $C = 40$ clients ($C = 20$ on MUTAG due to less data) and evenly distribute the training graphs in each dataset to the clients. The total number of iterations is 200 in all datasets, except the larger RDT-M5K that is 400. In each iteration, the server randomly selects 50% of the total clients for training. The clients use the *de facto* Graph Isomorphism Network (GIN) [86] as the graph classifier. We use its open-source code (https://github.com/weihua916/powerful-gnns) in our experiments. By default, 50% of malicious clients' training graphs are randomly sampled to inject the backdoor trigger, and the target label is 1. All testing graphs are chosen for backdoored testing.
There are several hyperparameters that can affect all attacks' performance on FedGL: fraction of malicious clients ρ and trigger node size n_{tri} or a threshold size n_{tri}^* in Customized-Trigger scheme. We set $\rho = 20\%$ and $n_{tri} = 4$ by default, and set $n_{tri}^* = 5$ in all experiments. We will also study the impact of these hyperparameters. Our Opt-GDBA contains many modules and we will also study the importance of individual module.
Evaluation metrics: We adopt four metrics for evaluation: the main task accuracy (MA) and backdoor accuracy (BA) on testing graphs; the average trigger node size (also use n_{tri}) and edge size (also use e_{tri}) of all backdoored training graphs. A more effective attack would achieve a higher MA and higher BA, and a more stealthy attack would have a lower n_{tri} and e_{tri} given a close MA/BA.

4.2 Experimental Results

4.2.1 Main results of the compared attacks. Table 1 shows the comparison results of the attacks in the default setting (***more comprehensive comparison and more results are shown in Table 9-Table 14 in the full report.*** We have the below key observations:

(1) **Main task performance is marginally sacrificed under all attacks:** All attacks achieve a close MA, compared to the MA without attack (i.e., the differences between them in all cases are $\leq$ 3%). This verifies these attacks only slightly affect the performance of the main task.
(2) **Rand-GDBA outperforms Rand-GCBA on attacking FedGL:** Similar to the conclusion of backdoor attacks on image data [79], distributed backdoors for graph data are also superior to centralized backdoors on attacking FedGL. Specifically, the BA of

Table 1: Results of all the compared attacks in the default setting. The gain is between Opt-GDBA and Rand-GDBA.

Datasets (MA without attack)	Customized-Trigger ($n^*_{tri}=5$) (MA / BA)	n_{tri}	e_{tri}	n_{tri}	Opt-GDBA (MA / BA)	e_{tri}	Rand-GDBA (MA / BA)	e_{tri}	Rand-GCBA (MA / BA)	e_{tri}
BITCOIN (MA=0.73)	0.72 / 0.99 (↑0.36)	4.29	4.20	4	0.72 / 0.99 (↑0.36)	3.08	0.71 / 0.63	4	0.72 / 0.57	6
MUTAG (MA=0.74)	0.72 / 0.95 (↑0.43)	3.51	2.31	4	0.71 / 0.85 (↑0.33)	3.41	0.71 / 0.52	4	0.73 / 0.48	6
PROTEINS (MA=0.73)	0.72 / 0.90 (↑0.39)	3.75	2.56	4	0.72 / 0.90 (↑0.39)	2.07	0.70 / 0.51	4	0.71 / 0.33	6
DD (MA=0.73)	0.72 / 0.86 (↑0.46)	3.19	1.57	4	0.72 / 0.78 (↑0.38)	2.97	0.72 / 0.40	4	0.72 / 0.33	6
COLLAB (MA=0.75)	0.73 / 0.86 (↑0.32)	4.68	4.51	4	0.73 / 0.84 (↑0.30)	3.34	0.73 / 0.54	4	0.71 / 0.37	6
RDT-M5K (MA=0.53)	0.52 / 0.90 (↑0.33)	4.51	3.59	4	0.52 / 0.89 (↑0.32)	3.27	0.52 / 0.57	4	0.52 / 0.40	6

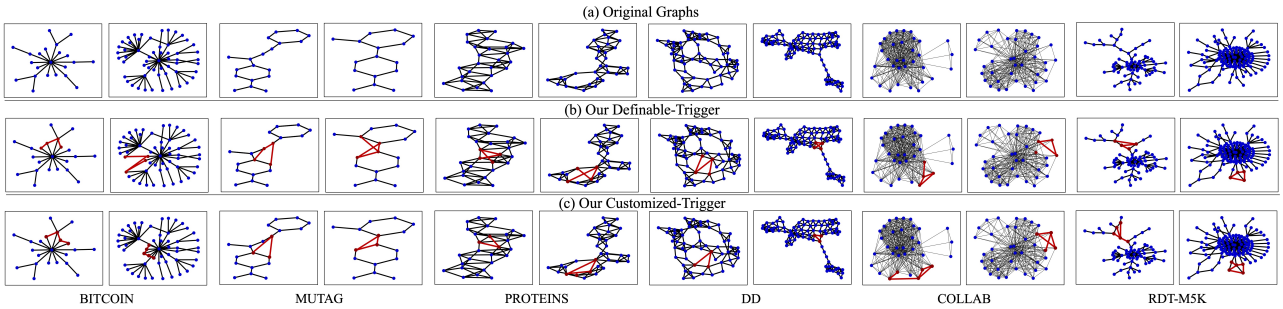

Figure 3: Examples of original clean graphs on the six datasets and their corresponding backdoored ones by our Opt-GDBA.

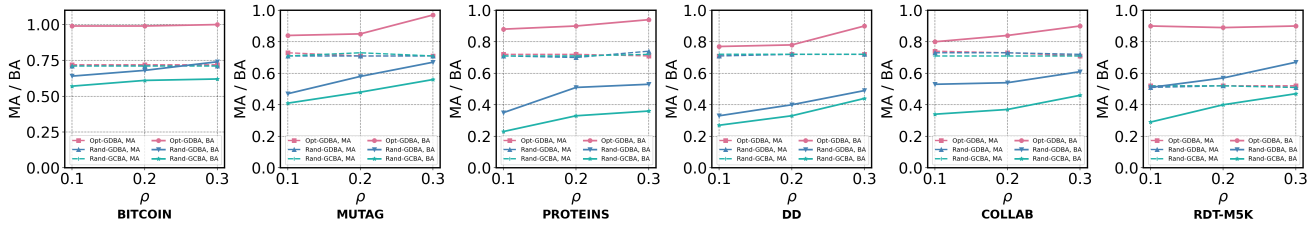

Figure 4: MA/BA vs. ρ on all compared attacks in all datasets.

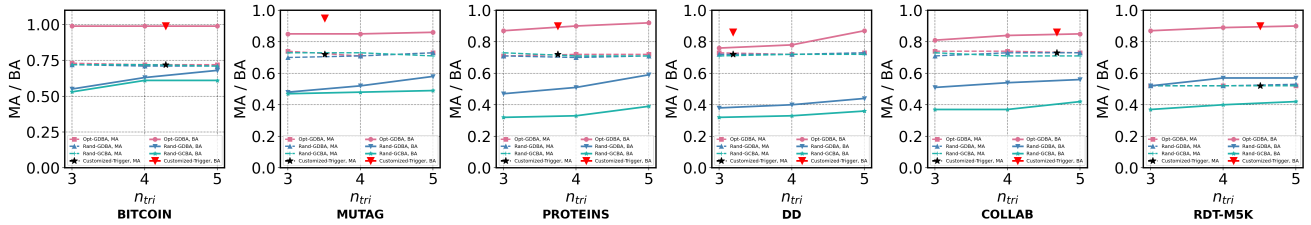

Figure 5: MA/BA vs. n_{tri} ($n^*_{tri}=5$) on all compared attacks in all datasets.

Table 2: Comparing the two trigger location learning schemes in our Opt-GDBA w.r.t. the average trigger edge size e_{tri}.

Datasets	BITCOIN			MUTAG			PROTEINS			DD			COLLAB			RDT-M5K		
ρ vs. e_{tri}	10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%
Definable-Trigger	4.95	5.09	5.35	5.83	5.88	5.63	3.56	3.36	3.44	5.42	5.04	5.22	5.85	5.72	5.99	5.39	5.45	5.59
Customized-Trigger	3.88	4.20	4.35	3.52	2.31	2.32	2.68	2.56	2.54	1.64	1.57	1.79	4.50	4.51	4.72	3.42	3.59	3.86

Rand-GDBA is higher than that of Rand-GCBA, with a gain of 4% to 18%. This implies the diverse local triggers indeed can promote the backdoor attack. This observation hence implies the importance of using distributed backdoors.

(3) **Our Opt-GDBA outperforms Rand-GDBA in terms of both attack effectiveness and stealthiness:** Both our Opt-GDBA with Definable-Trigger and Customized-Trigger schemes yield impressive attack results, with BA exceeding 85% and 90% in almost all cases, respectively. Under a same trigger node size

n_{tri} (Definable-Trigger) or a smaller learnt n_{tri} (Customized-Trigger), the BA of Opt-GDBA consistently and significantly surpasses that of Rand-GDBA. Particularly, the gain is from 30% to 46%. In addition, the average number of injected edges of Opt-GDBA is less than that of Rand-GDBA, showing Opt-GDBA is a more stealthy attack than Rand-GDBA. These findings underscore that the graph-dependent triggers learnt by our trigger generator are not only better memorized during the FedGL training, but also uncover important locations in the clean graphs. Figure 3 shows example triggers generated by Opt-GDBA on the six datasets. We can see most of the triggers are attached to the important/central nodes in the raw graphs.

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

Impact of the fraction ρ of malicious clients: Figure 4 shows the MA/BA results vs. $\rho = 10\%, 20\%, 30\%$. We can see MA is stable w.r.t. different ρ, and BA (slightly) increases with a larger ρ. For instance, on MUTAG, when ρ is from 10% to 30%, the BA of Rand-GCBA, Rand-GDBA, and our Opt-GDBA can be increased from 41% to 56%, from 43% to 67% and from 84% to 97% with the Definable-Trigger, respectively. This shows MA is marginally affected, but the attack becomes stronger with more malicious clients.

Impact of the trigger size n_{tri}: Figure 5 shows the MA/BA results vs. n_{tri} (=3,4,5). We can see a larger trigger size corresponds to a larger BA, which implies a stronger attack. This is because the trigger can be injected to a larger region of the clean graph. Still, the MA is the very stable in terms of different trigger sizes.

Impact of the trigger location learning scheme: Our Opt-GDBA uses two schemes to decide the trigger location: Customized-Trigger automatically learns it, while Definable-Trigger predefines it. Table 2 shows the comparison results. We can see the trigger outputted by Customized-Trigger has an average number of edges < 5 in all cases. In contrast, Definable-Trigger yields $\gtrapprox 5$ edges in most cases. Note that the MA and BA of the two schemes are close. This hence reflects the Customized-Trigger scheme can further locate the "more important" region in a graph to attach the trigger.

Global trigger vs. local triggers: From Table 1, we know the MA performance is marginally affected by the backdoor attacks with respect to different trigger sizes. Recall that, during testing, we use the combined local triggers to form a global trigger (a complete subgraph), which is injected into all testing graphs. In this experiment, we also explore the BA performance of our Opt-GDBA where each client uses the local triggers generated by its own trigger generator. Specifically, we use $\rho = 20\%$ and the total number of malicious clients is $20\% * 50\% * 40 = 4$. Figure 6 compares the BA produced by the global trigger vs. local triggers per malicious client with Definable-Trigger Opt-GDBA with $n_{tri} = 4$. For instance, "Local triggers 1" means the local triggers are generated via malicious client 1's trigger generator on the corresponding testing graphs.

We observe that though the backdoored FedGL training does not involve the global trigger, the BA achieved by the global trigger is even larger than that by the local triggers. One possible reason could be that the federated training might memorize the combined effect of local triggers. This phenomenon further reinforces the FedGL framework is more vulnerable to distributed backdoors.

4.2.3 Persistence and stealthiness of the triggers. In this experiment, we explore the persistence and stealthiness of the backdoor triggers generated by our Opt-GDBA.

(Persistence) Finetuning the backdoored FedGL model with clean graphs cannot remove the backdoor effect: A straightforward strategy to mitigate the backdoor effect is to finetune the FedGL model only with clean graphs. To simulate this, we extend the FedGL training with an extra 200 iterations (e.g., from 200 to 400) which only involves training with clean graphs. Table 4 shows the results. We can see the BA with finetuning is close to that without finetuing in all ρ and when the trigger node size is not large (i.e., $n_{tri} = 3, 4$). This shows the backdoor effect is persistent.

(Stealthiness) Similarity between the backdoored graphs and clean graphs is large: We further quantitatively compare the structure similarity between the generated backdoored graphs and the clean graphs, where we use the metrics NetSim and DeltaCon proposed in [73]. Table 3 shows the similarity results over all training graphs. We observe the backdoored graphs and their clean counterparts are structurally close (except BITCOIN where one possible reason could be the BITCOIN graph is very sparse).

The above results imply that empirical defenses based on finetuning and structure similarity test are hard to detect or remove the backdoor trigger. Also, empirical defenses are always broken by advanced/adaptive attacks [94]. Hence, it is necessary to develop certified defenses for backdoored FedGL. More details see Section 5.

4.2.4 Ablation study. In this experiment, we examine the contribution of each module in our trigger generator. The modules include Trigger-Location (based on the Edge-View module), Trigger-Shape, Customized-Trigger, and Definable-Trigger. For simplicity, we test on BITCOIN, and the other datasets show similar observations. The results are summarized in Table 5 with $\rho = 20\%$, and $n_{tri} = 4$ in Definable-Trigger and $n^*_{tri} = 5$ in Customized-Trigger.

(a) The whole generator as a reference. **(b)** We exclude only the Edge-View sub-module in Trigger-Location, indicating that trigger locations in graphs are computed solely based on the rank of the node features. **(c)** We remove Trigger-Location and decide the trigger location in each graph randomly. Compared with **(a)**, the significant BA reductions of 16% for Customized-Trigger, and 15% and 18% for Definable-Trigger in **(b)** and **(c)** demonstrate that the Trigger-Location module excels at selecting important nodes in the graphs. **(d)** We remove Trigger-Shape and use the ER model [18] to decide the trigger shape. The reductions of 27% and 22% in BA underscore the superior effectiveness of our Trigger-Shape module in learning trigger shapes. **(e)** We remove both the Trigger-Shape and Trigger-Location modules, resorting to a random method for trigger location selection and an ER model for trigger shape determination. The substantial 36% reduction in BA proves the strong competition of our method for backdoor attacks on FedGL.

5 Certified Defense for FedGL

In this section, we design certified defenses for (backdoored) FedGL. Suppose we have learnt a backdoored FedGL for graph classification. We aim to build a certifiably robust defense mechanism such that the graph classifier: 1) provably predicts the correct label for clean testing graphs injected with arbitrary trigger with a bounded size; and 2) provably predicts a non-target label for backdoored testing

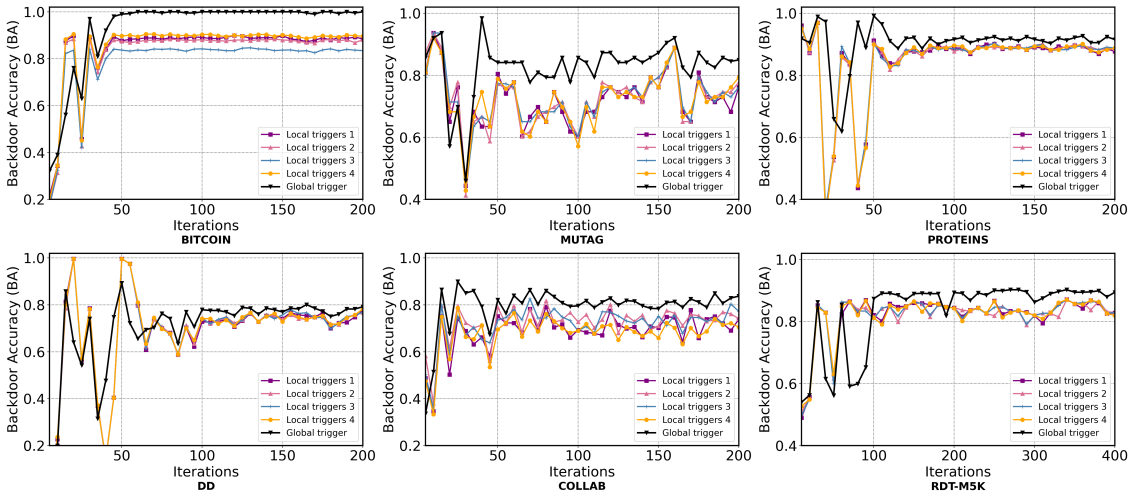

Figure 6: Comparing the backdoor performance with global trigger vs. local triggers generated by malicious clients.

Table 3: Structure similarity between the generated backdoored graphs by our Opt-GDBA and the clean graphs.

Datasets	BITCOIN				MUTAG				PROTEINS				DD				COLLAB				RDT-M5K			
$n_{tri}(n^*_{tri})$	3	4	5	5*	3	4	5	5*	3	4	5	5*	3	4	5	5*	3	4	5	5*	3	4	5	5*
NetSim (↑)	0.73	0.55	0.52	0.54	0.99	0.90	0.82	0.87	0.93	0.88	0.80	0.86	1.00	0.99	0.99	0.99	1.00	0.99	0.99	0.99	0.99	0.98	0.97	0.98
DeltaCon (↑)	0.80	0.65	0.63	0.64	0.96	0.93	0.89	0.92	0.95	0.94	0.89	0.91	1.00	0.99	0.99	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.98	0.98

Table 4: Finetuning the backdoored FedGL model by extending the training on clean graphs.

Datasets	ρ	10%			20%			30%		
	n_{tri}	3	4	5	3	4	5	3	4	5
BITCOIN	BA	0.97	0.99	0.98	0.99	0.99	0.99	0.99	1.00	0.95
	BA-FT	0.95	0.97	0.96	0.94	0.95	0.96	0.96	0.95	0.73
MUTAG	BA	0.83	0.84	0.82	0.87	0.85	0.86	0.95	0.97	0.94
	BA-FT	0.82	0.82	0.81	0.85	0.84	0.84	0.92	0.92	0.65
PROTEINS	BA	0.82	0.91	0.88	0.87	0.92	0.90	0.94	0.94	0.96
	BA-FT	0.77	0.90	0.82	0.87	0.89	0.76	0.94	0.92	0.65
DD	BA	0.78	0.77	0.80	0.76	0.78	0.87	0.83	0.80	0.92
	BA-FT	0.70	0.74	0.76	0.72	0.76	0.84	0.75	0.73	0.61
COLLAB	BA	0.80	0.80	0.82	0.81	0.84	0.85	0.85	0.90	0.91
	BA-FT	0.79	0.77	0.80	0.79	0.81	0.80	0.82	0.85	0.77
RDT-M5K	BA	0.88	0.90	0.85	0.87	0.89	0.90	0.90	0.89	0.92
	BA-FT	0.87	0.85	0.72	0.85	0.94	0.75	0.88	0.86	0.60

Table 5: Impact of different modules in our adaptive trigger generator on the (MA/BA) performance on Bitcoin. T-L: Trigger-Location; T-S: Trigger-Shape; E-V: Edge-View; Cus-T: Customized-Trigger; Def-T: Definable-Trigger.

Models	T-L	T-S	T-L w/o E-V	Cus-T	Def-T
(a)	✓	✓		**0.72 / 0.99**	**0.72 / 0.99**
(b)		✓	✓	0.72 / 0.83	0.72 / 0.84
(c)		✓		-	0.71 / 0.81
(d)	✓			0.72 / 0.72	0.72 / 0.77
(e)				-	0.71 / 0.63

graphs. This has the implication that benign clients' performance are provably kept, while the malicious clients' backdoored effect are provably removed during testing.

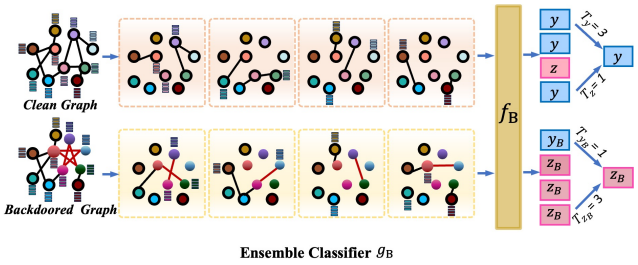

Figure 7: Overview of our proposed certified defense.

Our defense includes three key steps: 1) dividing a (clean or backdoored) testing graph into multiple subgraphs; 2) building a majority vote-based ensemble graph classifier on the subgraphs; and 3) deriving the robustness guarantees of the ensemble graph classifier against arbitrary trigger. Figure 7 overviews our defense.

5.1 Graph Division into Subgraphs

Recall that a backdoor attack can arbitrarily perturb the edges $\mathcal{E}$ and node features $\mathbf{X}$ in a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ as the trigger can be put in any location with any shape. To defend against this attack, our main idea is to design a deterministic function h[2] to divide G into different subgraphs, such that each edge and node (feature) in G is deterministically mapped into only one subgraph.

Hash function as the mapping: We use the cryptographic hash function (e.g., MD5) as our mapping function. It takes input as a bit string and outputs an integer (e.g., 128-bit long with the integer

[2]We emphasize that the function should be independent of the graph structure and node features. Otherwise, an attacker may possibly "reverse engineer" the function to find the relation between the output and input.

range $[0, 2^{128}-1]$). Here, we propose to use node indexes and stringify them as the input to the hash function. For instance, for a node v, we denote its string as $\text{str}(v)$. Given the $\text{str}(v)$ of every node $v \in \mathcal{V}$, we propose mapping the nodes and edges using the hash function h and dividing a graph G into multiple (e.g., T) subgraphs.

Node feature and edge division: First, we divide node features into T groups using the hash function h. Specifically, we compute the hash value $\big(h[\text{str}(u)] \bmod T + 1\big)$ for every node $u \in \mathcal{V}$, where mod is the modulo function. We use $\mathcal{V}^t$ to denote the set of nodes whose group index is t, i.e., $\mathcal{V}^t = \{u \in \mathcal{V} \mid h[\text{str}(u)] = t\}$, $t = 1, 2, \cdots T$. Correspondingly, we use $\mathbf{X}^t \in \mathbb{R}^{|\mathcal{V}| \times d}$ to denote the node features in the tth group. With such grouping, we observe that nodes not in the tth group do not have features in $\mathbf{X}^t$. To mitigate this, we simply set features of these nodes to be zeros. I.e., $\mathbf{X}^t_v = \mathbf{X}_v$, if $v \in \mathcal{V}^t$; and $\mathbf{X}^t_v = \mathbf{0}$, otherwise.

Next, we divide edges into T groups. Specifically, we compute $h[\text{str}(u) + \text{str}(v)] \bmod T + 1$ for every edge $(u, v) \in \mathcal{E}, u \leq v$, where "+" means the string concatenation. We then let $h[\text{str}(v) + \text{str}(u)] = h[\text{str}(u) + \text{str}(v)]$. This is to ensure an undirected edge has a same hash value. We use $\mathcal{E}^t$ to denote the set of edges whose group index is t, i.e., $\mathcal{E}^t = \{(u, v) \in \mathcal{E} \mid h[\text{str}(u) + \text{str}(v)] = t\}$.

Then, we construct T subgraphs, i.e., $G^t = (\mathcal{V}, \mathcal{E}^t, \mathbf{X}^t)$ with $t = 1, 2, \cdots, T$, for a graph G. Notice the node features and edges are non-overlapped between different subgraphs. That is, $\mathbf{X}^i \cap \mathbf{X}^j = \emptyset, \mathcal{E}^i \cap \mathcal{E}^j = \emptyset, \forall i, j \in \{1, 2, \cdots, T\}, i \neq j$. This is a requirement to enable deriving our robustness guarantee. Note also that the subgraph does not need to be connected, as a graph classifier can still predict a label for a graph with disconnected components.

5.2 Majority Vote-based Ensemble Classifier

Let the backdoored graph classifier be f_B. Given a clean testing graph G (with true label y), we construct T subgraphs $\{G^t\}$ using our graph division strategy and introduce a majority vote-based ensemble graph classifier g_B to classify these T subgraphs. Specifically, we denote by T_l the number of subgraphs classified as the label l by f_B, i.e., $T_l = \sum_{t=1}^{T} \mathbb{1}(f_B(G^t) = l)$. Then, we define g_B as:

$$g_B(G) = \arg\max_{l \in \mathcal{Y}} T_l, \tag{10}$$

which returns a smaller index when ties exist. Let $y = g_B(G)$ by assuming the ensemble classifier accurately predicts the clean graph.

Similarly, for a backdoored testing graph G_B (with the target label y_B), we construct T subgraphs $\{G_B^t\}$ using the graph division strategy and denote by T_{l_B} the number of subgraphs classified as the label l_B by f_B, i.e., $T_{l_B} = \sum_{t=1}^{T} \mathbb{1}(f_B(G_B^t) = l_B)$. Then, we have:

$$g_B(G_B) = \arg\max_{l_B \in \mathcal{Y}} T_{l_B}. \tag{11}$$

Likewise, we let $y_B = g_B(G_B)$ by assuming the backdoored testing graph successfully triggers the backdoor.

5.3 Certified Robustness Guarantees

With our graph division strategy and ensemble classifier, we can derive the robustness guarantee for clean graphs against backdoor trigger and backdoored graphs. ***Proofs are in the full report.***

5.3.1 Certified robustness w.r.t. clean graph. Assume we have a backdoored graph $\tilde{G}$ generated from the clean graph G. We use $\tilde{G}^1, \tilde{G}^2, \cdots, \tilde{G}^T$ to denote the T subgraphs from $\tilde{G}$ via the graph division strategy. Moreover, we denote by $\tilde{T}_l = \sum_{t=1}^{T} \mathbb{1}(f_B(\tilde{G}^t) = l), \forall l \in \mathcal{Y}$, and $g_B(\tilde{G}) = \arg\max_{l \in \mathcal{Y}} \tilde{T}_l$. We aim to ensure $g_B(G) = g_B(\tilde{G})$ when the perturbation size induced by the backdoor trigger is bounded by a threshold (call *certified perturbation size*), where the perturbation size is defined as the sum of the perturbed number of nodes (whose features can be arbitrarily modified) and edges w.r.t. G. Formally, we state the theorem below:

Theorem 1 (Certified perturbation size w.r.t. clean graph). *Given a backdoored graph classifier f_B and our ensemble graph classifier g_B. Given a clean testing graph G with a label y and its T subgraphs $\{G^t\}_{t=1}^T$ produced by our graph division strategy. Suppose T_y and T_z are the largest and second largest frequency outputted by f_B on predicting $\{G^t\}_{t=1}^T$. Let m be the perturbation size induced by an arbitrary backdoor trigger and the respective backdoored graph is $\tilde{G}$. Then $g_B(G) = g_B(\tilde{G}) = y$, when m satisfies:*

$$m \leq m^* = \lfloor \frac{T_y - T_z + \mathbb{1}(y < z) - 1}{2} \rfloor. \tag{12}$$

We have below remarks of the theoretical result from Theorem 1:

- It can be applied for *any* backdoored FedGL model.
- It holds for *any* backdoored attack with a trigger that arbitrarily perturbs m^* edges and nodes.
- It does not restrict the trigger to be connected.
- The robustness guarantee is true with a probability 100%.

Next, we further show our certified robustness guarantee is tight.

Theorem 2 (Tightness of m^*). *For any m satisfying $m > m^*$, there exists a base classifier $f_B' \neq f_B$ that will make g_B misclassify $\tilde{G}$. That being said, it is impossible to derive a larger certified perturbation size than m^* in Theorem 1, without using extra information on f_B.*

5.3.2 Certified robustness w.r.t. backdoored graphs. For a backdoored graph G_B, we consider its robustness against our defense strategy. We have the below theorem.

Theorem 3 (Certified (non-)backdoored graph). *Let f_B and g_B be defined as Theorem 1. Given a backdoored testing graph G_B with a target label y_B and its T subgraphs $\{G_B^t\}_{t=1}^T$ produced by our graph division. Let T_{y_B} and T_{z_B} be the largest and second largest frequency outputted by f_B on predicting $\{G_B^t\}_{t=1}^T$. Then if $T_{y_B} > T_{z_B} - \mathbb{1}(y_B < z_B)$, G_B is a certified backdoored graph for our defense, otherwise it is a certified non-backdoored graph.*

A certified backdoored graph means it provably evades our defense, while a certified non-backdoored graph means our defense provably predicts it as a non-target label.

6 Certified Defense Results

6.1 Experimental Setup

Parameter setup: We first train the backdoored FedGL model under our Opt-DGBA (with a specified ρ, n_{tri} in Definable-Trigger or n^*_{tri} in Customized-Trigger). The trained backdoored graph classifier f_B is then for both normal testing (i.e., on cleaning testing graphs) and backdoor testing (i.e., on the backdoored testing graphs generated by our Opt-GBA). Here, we only select successfully backdoored testing graphs for evaluation. Unless otherwise specified, we use $\rho = 20\%$, $n_{tri} = 4$, or $n^*_{tri} = 5$.

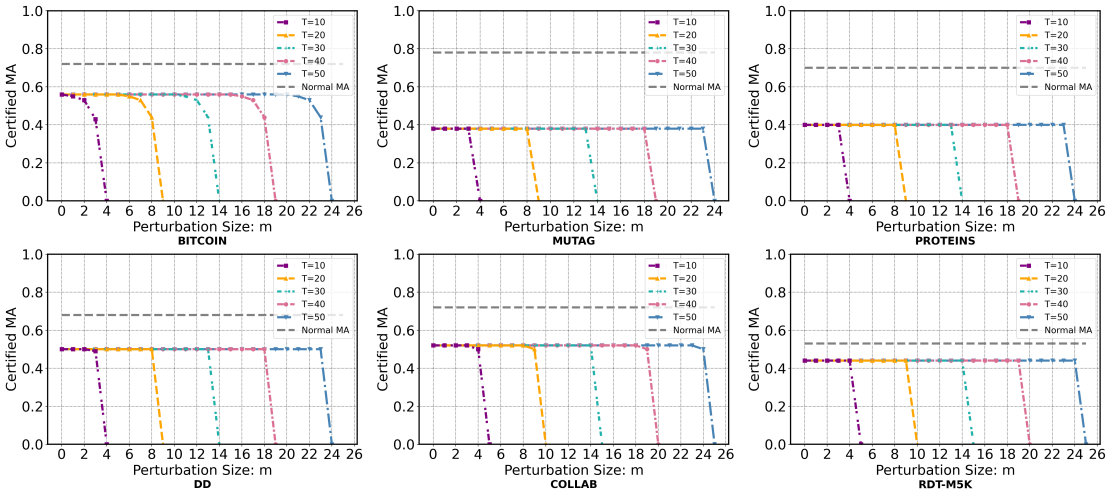

Figure 8: Certified MA vs. T. 100 testing graphs are randomly sampled (and all testing graphs in MUTAG) for evaluation. Normal MA (under no attack and defense) is also reported for reference.

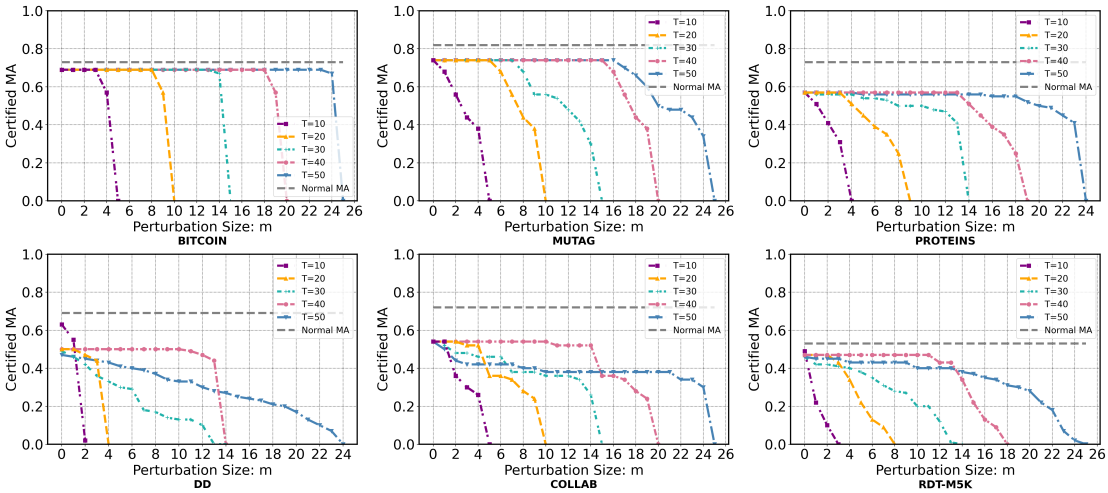

Figure 9: Certified MA vs. T, where we finetune the backdoored FedGL model with augmented subgraphs that are generated from the benign clients' training graphs.

To apply our defense, for each (clean/backdoored) testing graph, we use our graph division strategy to divide it into T subgraphs and majority vote-based ensemble classifier g_B to predict these subgraphs. The key hyperparameter in our defense is the number of subgraphs T. By default we set $T = 30$. We also test its impact on the defense performance.

Evaluation metrics: We use the certified accuracy [5, 55, 64] on the testing graphs for evaluation.

- **Certified MA at perturbation size m:** the fraction of the clean testing graphs that are provably classified as the true label against an arbitrary trigger whose size is m.
- **Certified BA:** the fraction of the backdoored testing graphs that are provably classified as the target label against our defense.

6.2 Experimental Results

6.2.1 Results on certified MA. In this experiment we show the results on certified MA against the backdoored FedGL model trained under our Opt-GDBA. ***More results are shown in the full report.***

Certified MA vs. T: Figure 8 shows the certified MA at perturbation size m vs. different T, where we test on all 63 testing graphs in MUTAG and randomly sampled 100 testing graphs in the other datasets. For reference, we also report the normal MA without our defense. In general, we observe our defense is provably more robust (i.e., larger certified MA with larger certified perturbation size) when T is larger. For instance, on BITCOIN, when $T = 30, 50$, certified MA are 44% and 58% with $m = 13$, and the maximum certified perturbation size are $m^* = 13$ and 23, respectively. The reason is our majority vote-based ensemble classifier could tolerate more perturbed subgraphs when T increases. As an arbitrary node/edge

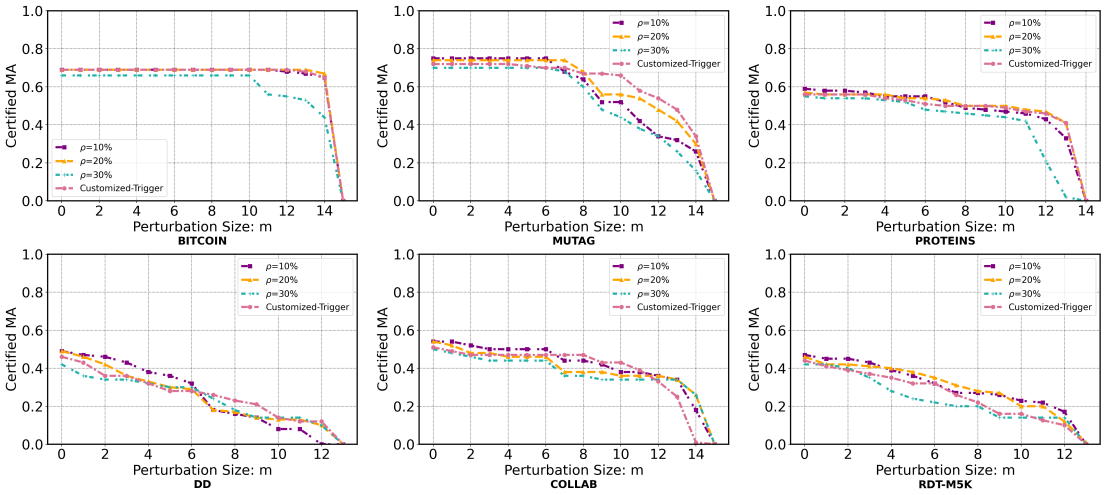

Figure 10: Certified MA vs. ratio ρ of malicious clients ($T = 30$, $n_{tri} = 4$)

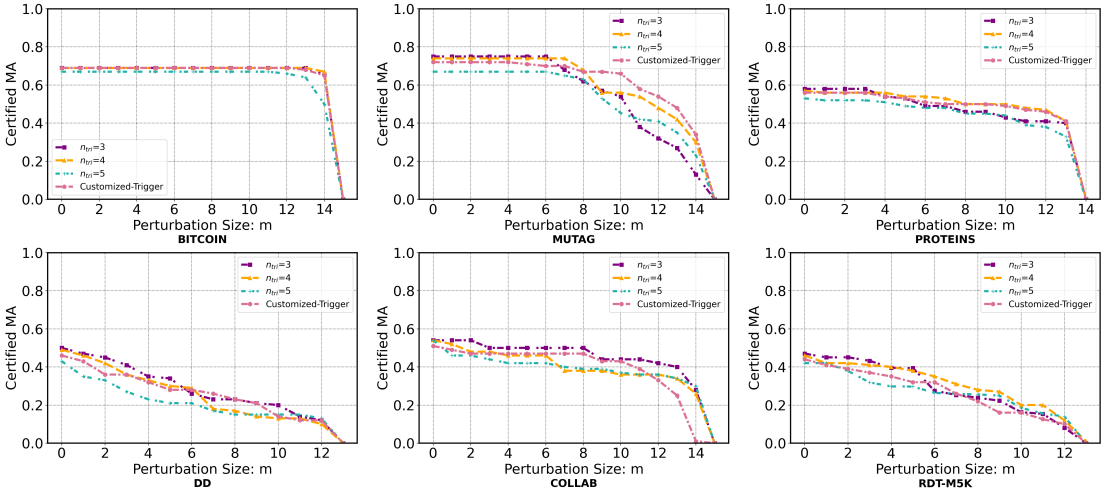

Figure 11: Certified MA vs. trigger node size n_{tri} ($T = 30$, $\rho = 20\%$).

perturbation induced by the trigger can affect at most one clean subgraph's prediction, a larger T implies being robust to a larger m.

Though effective, we still see a large gap between certified MA and normal MA on datasets such as MUATG and PROTEINS. This is due to the number of accurately predicted subgraphs (i.e, T_y in Equation (12)) after graph division is not large enough on these datasets. We note that the backdoored FedGL training only uses the whole training graph. To enhance the certified MA, we propose to finetune the backdoored FedGL model with extra subgraphs created from benign clients' training graphs. Specifically, in each benign client, we use $T = \{10, 20, ..., 50\}$ to generate a set of subgraphs for each training graph and pick one subgraph from each T. These subgraphs have the same label as the raw graph. Figure 9 shows the results. We observe the finetuned model with clean subgraphs yield a significantly higher certified MA on the relatively sparser/smaller datasets (e.g., BITCOIN, MUTAG, and PROTEINS). This implies the finetuned model learns a correct mapping between the subgraphs and the true label, and hence improves the accuracy of subgraphs created from the testing graphs. In contrast, we see a drop of certified MA on relatively denser/larger datasets (e.g., DD, COLLAB, and RDT-M5K). This is possibly because the finetuned model is hard to associate both the large dense graphs and their generated much smaller and sparser subgraphs with the true label.

The above results suggest that, in practice, to enhance the certified MA of a backdoored FedGL model, we could augment the training graphs with their subgraphs on small/sparse datasets, but may not on large/denser datasets.

Certified MA vs. ρ: In this experiment, we assess the defense performance on Opt-GDBA attacked FedGL models with varying ρ of malicious clients. Figure 10 shows the results with $T = 30$ and $n_{tri} = 4$. We observe the certified MA and certified perturbation size are similar with different ρ's (except a slight drop when $\rho = 30\%$). This is primarily because the Opt-GDBA's MA and BA are relatively stable across different ρ, as shown in Table 1. This ensures the

Table 6: Certified BA and MA of backdoored graphs under our defense (in all T, ρ, and n_{tri}).

Datasets	BITCOIN	MUTAG	PROTEINS	DD	COLLAB	RDT
Certified BA	0	0	0	0	0	0
MA	1.00	1.00	1.00	1.00	0.96	0.92

number of correctly predicted subgraphs via our ensemble classifier is also close in different ρ's, and so do the certified MA and certified perturbation size.

Certified MA vs. n_{tri}: This experiment explores the defense performance with varying trigger node sizes used by Opt-GDBA. Figure 11 shows the results. Similarly, our defense achieves similar certified MA in general, with large n_{tri} slightly reduces the certified MA.

6.2.2 Results on certified BA. In this experiment, we evaluate the robustness of the backdoored testing graphs generated by our Opt-GDBA under our defense. Table 6 shows the results of certified BA. We observe the certified BA is 0 in all T, ρ, and n_{tri}. Recall our graph division strategy ensures the divided subgraphs have non-overlapping edges and node features. The above results can be attributed to two aspects: the trigger in the backdoored testing graph is separated into: 1) a few subgraphs that are still classified as the *target label*, but the other majority subgraphs are mostly classified as a non-target label (actually the true label in most cases); or 2) a large number of the subgraphs that makes it difficult to form any effective trigger in the subgraphs. In either case, the number of successful backdoored subgraphs is a minority. Hence, with the majority voting, all the backdoored testing graphs are misclassified as a *non-target label*. The results imply the backdoored testing graphs generated by our Opt-GDBA are completely broken by our graph division + ensemble classifier based defense.

We also calculate MA on the generated backdoored graphs (which have correct predictions without backdoor) under our defense. We obtain $\geq$ 92% MA in all datasets, where 4 datasets are 100%. *This means our defense does not/marginally affect clean labels, so the FedGL's utility is still maintained.* This is because the proposed defense is mainly designed to affect the backdoored effect in the backdoored subgraphs, but not affect the utility of clean subgraphs.

7 Related Work

Backdoor attacks on centralized learning for non-graph data and defenses: Extensive works have shown centralized machine learning models for non-graph data, such as image [10, 11, 22, 36, 40, 53, 59, 62, 72, 89], text [9, 15, 45, 48, 87], audio [17, 23, 50, 56], and video [83, 96], are vulnerable to backdoor attacks. A backdoored model produces attacker-desired behaviors when the same trigger is injected into testing data. Gu et al. [22] proposed the first backdoor attack, called BadNet, on image classifiers. The attack injects a trigger (e.g., a sticker with yellow square) into "STOP" sign from the U.S. stop signs database and changes their labels to the "SPEED" sign. The trained backdoored image classifier then predicts a "STOP" sign with the same sticker trigger to be the "SPEED" sign.

Many empirical defenses [10, 16, 24, 38–40, 44, 66, 68] have been proposed to mitigate backdoor attacks. For instance, Wang et al. [66] proposed Neural Cleanse to detect and reverse engineer the trigger. However, all these defenses are broken by adaptive attacks [71]. These two works [63, 71] proposed provable defenses against backdoor attacks in the image domain. However, they are shown to have insufficient effectiveness against backdoor attacks. In addition, they cannot be applied to inputs with different sizes.

Backdoor attacks on federated learning for non-graph data and defenses: Backdoor attacks on FL are categorized as centralized backdoor attack (CBA) [4, 21, 67, 95], where all malicious clients use a shared trigger, and distributed backdoor attack (DBA) [79], where each malicious client uses its own defined trigger. For instance, Bagdasaryan et al. [2] designed the first CBA via model replacement. Inspired by the distributed learning property of FL, Xie et al. [79] designed the first DBA which is shown to be more persistent and stealthy than CBAs on FL.

Many empirical defenses [7, 42, 43, 49, 57, 93] have been proposed, and can be performed in the FL stage of pre-aggregation [42, 49], in-aggregation [43, 57, 93], and post-aggregation [7]. However, they can only defend against known attack techniques, and an adversary aware of the existence of these defenses can break them [67]. Existing certified defenses [6, 8, 78, 80] for FL can only tolerate a very small number of malicious clients and/or incur a large computation/communication cost for clients.

Backdoor attacks on centralized graph learning and defenses: Unlike non-graph data that can be represented via Cartesian coordinates and have fixed input size, graphs cannot do so and different graphs often have varying sizes, making the trigger hard to be defined. To address this, two recent works [76, 94] propose to use *subgraph* as a trigger. Zhang et al. [94] use a random subgraph as the trigger shape, which is generated by random graph generation models (such as the Erdős-Rényi [18], Small World [70], and Preferential Attachment [3]), and pick random nodes as the trigger location. Instead of using a random trigger shape, Xi et al. [76] designed a trigger generator to learn to generate the trigger shape for each graph using its edge and node feature information. However, the trigger randomly chooses nodes as the trigger location.

Zhang et al. [94] proposed a certified defense for a backdoored graph classifier by extending randomized ablation [35] for image classifiers. Specifically, they built a randomized subgraph sampling based defense mechanism to ensure the backdoored graph classifier provably predicts the same label for a testing graph if the injected trigger has a size less than a threshold. However, their defense is limited to edge perturbation and their robustness guarantee could be incorrect with a certain probability.

Backdoor attacks on federated graph learning: Xu et al. [84] is the only work studying backdoor attacks on FedFL. It is inspired by [79, 94] with random subgraph as a trigger. We showed their backdoor performance is not good enough with a smaller trigger.

Majority-voting based ensemble for certified defenses: The key insight of this type of defense is to ensure only a bounded number of corrupted votes/predictions (each prediction is treated as a vote) are changed with a bounded adversarial perturbation. This idea has been used in certified defenses against adversarial patch attacks [33, 77], data poisoning attacks [30, 31, 34], and others [28, 46, 92]. The key difference among these methods is that they create problem-dependent voters for the majority vote. A closely relevant work to ours is [88], but they have two key differences. First, the studied problem is different. [88] proposes a majority-voting strategy for GL to defend against evasion attacks, while ours

for FedGL to defend against distributed backdoor attacks. Second, the graph division strategy is different. [88] divides a graph into *overlapped* subgraphs, which facilitates deriving the robustness guarantee against graph perturbation *or* node (feature) perturbation, but not both. Instead, our method can concurrently defend against both graph *and* node (feature) perturbations.

8 Conclusion

We study the robustness of FedGL from both the attacker's and defender's perspective. We first design an effective, stealthy, and persistent DBA on FedGL. Instead of using a random (centralized or distributed) trigger that is injected into random position in a graph, our attack develops a trigger generator that adaptively learns the important trigger location and shape per backdoored graph. Our attack results show existing empirical defenses based on backdoor detection or removal are ineffective. Then, we further develop a certified defense for backdoored FedGL model based on graph division and majority vote-based ensemble. We derive the certified robustness as well as its tightness w.r.t. clean graphs against arbitrary trigger and backdoored graphs generated by our attack.

Acknowledgments

We thank all anonymous reviewers for the constructive comments. Li is partially supported by the National Natural Science Foundation of China under Grant No. 62072208, Key Research and Development Projects of Jilin Province under Grant No. 20240302090GX. Hong is partially supported by the National Science Foundation under grant Nos. CNS-2302689, CNS-2308730, CNS-2319277 and CMMI-2326341. Wang is partially supported by the National Science Foundation under grant Nos. ECCS-2216926, CNS-2241713, CNS-2331302 and CNS-2339686. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

[1] Jinheon Baek, Wonyong Jeong, Jiongdao Jin, Jaehong Yoon, and Sung Ju Hwang. 2023. Personalized subgraph federated learning. In *ICML*.
[2] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *AISTATS*.
[3] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* (1999).
[4] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In *ICML*.
[5] Aleksandar Bojchevski, Johannes Gasteiger, and Stephan Günnemann. 2020. Efficient robustness certificates for discrete data: Sparsity-aware randomized smoothing for graphs, images and more. In *ICML*.
[6] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. Provably secure federated learning against malicious clients. In *AAAI*, Vol. 35. 6885–6893.
[7] Xiaoyu Cao, Jinyuan Jia, Zaixi Zhang, and Neil Zhenqiang Gong. 2023. Fedrecover: Recovering from poisoning attacks in federated learning using historical information. In *IEEE Symposium on Security and Privacy (SP)*.
[8] Xiaoyu Cao, Zaixi Zhang, Jinyuan Jia, and Neil Zhenqiang Gong. 2022. Flcert: Provably secure federated learning against poisoning attacks. *IEEE Transactions on Information Forensics and Security* 17 (2022), 3691–3705.
[9] Kangjie Chen, Yuxian Meng, Xiaofei Sun, Shangwei Guo, Tianwei Zhang, Jiwei Li, and Chun Fan. 2022. BadPre: Task-agnostic Backdoor Attacks to Pre-trained NLP Foundation Models. In *ICLR*.
[10] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv* (2017).
[11] Joseph Clements and Yingjie Lao. 2018. Hardware trojan attacks on neural networks. *arXiv preprint arXiv:1806.05768* (2018).
[12] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*. Springer, 1–15.
[13] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2023. Benchmarking Graph Neural Networks. *Journal of Machine Learning Research* 24, 43 (2023), 1–48.
[14] FedML supports several out-of-the-box deep learning algorithms for various data types, such as tabular, text, image, graphs, and Internet of Things (IoT) data. [n. d.]. https://aws.amazon.com/blogs/machine-learning/part-2-federated-learning-on-aws-with-fedml-health-analytics-without-sharing-sensitive-data/.
[15] Leilei Gan, Jiwei Li, Tianwei Zhang, Xiaoya Li, Yuxian Meng, Fei Wu, Yi Yang, Shangwei Guo, and Chun Fan. 2022. Triggerless Backdoor Attack for NLP Tasks with Clean Labels. In *ACL-HLT*.
[16] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. 2019. Strip: A defence against trojan attacks on deep neural networks. In *ACSAC*.
[17] Yunjie Ge, Qian Wang, Jiayuan Yu, Chao Shen, and Qi Li. 2023. Data Poisoning and Backdoor Attacks on Audio Intelligence Systems. *IEEE Communications Magazine* (2023).
[18] Edgar N Gilbert. 1959. Random graphs. *The Annals of Mathematical Statistics* 30, 4 (1959), 1141–1144.
[19] Michelle Goddard. 2017. The EU General Data Protection Regulation (GDPR): European regulation that has a global impact. *International Journal of Market Research* 59, 6 (2017), 703–705.
[20] Xueluan Gong, Yanjiao Chen, Jianshuo Dong, and Qian Wang. 2022. ATTEQ-NN: Attention-based QoE-aware Evasive Backdoor Attacks.. In *NDSS*.
[21] Xueluan Gong, Yanjiao Chen, Qian Wang, and Weihan Kong. 2022. Backdoor attacks and defenses in federated learning: State-of-the-art, taxonomy, and future directions. *IEEE Wireless Communications* (2022).
[22] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Proc. of Machine Learning and Computer Security Workshop*.
[23] Hanqing Guo, Xun Chen, Junfeng Guo, Li Xiao, and Qiben Yan. 2023. MASTERKEY: Practical Backdoor Attack Against Speaker Verification Systems. In *Annual International Conference on Mobile Computing and Networking*. 1–15.
[24] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. 2019. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763* (2019).
[25] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*.
[26] 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* (2021).
[27] 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 *AAAI*.
[28] Hanbin Hong, Binghui Wang, and Yuan Hong. 2022. Unicr: Universally approximated certified robustness via randomized smoothing. In *ECCV*.
[29] How AWS uses graph neural networks to meet customer needs . [n. d.]. https://www.amazon.science/blog/how-aws-uses-graph-neural-networks-to-meet-customer-needs/.
[30] Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. 2021. Intrinsic certified robustness of bagging against data poisoning attacks. In *AAAI*.
[31] Jinyuan Jia, Yupei Liu, Xiaoyu Cao, and Neil Zhenqiang Gong. 2022. Certified robustness of nearest neighbors against data poisoning and backdoor attacks. In *AAAI*.
[32] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[33] Alexander Levine and Soheil Feizi. 2020. (De) Randomized smoothing for certifiable defense against patch attacks. In *NeurIPS*.
[34] Alexander Levine and Soheil Feizi. 2020. Deep Partition Aggregation: Provable Defenses against General Poisoning Attacks. In *ICLR*.
[35] Alexander Levine and Soheil Feizi. 2020. Robustness certificates for sparse adversarial attacks by randomized ablation. In *AAAI*.
[36] Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. 2018. Hu-fu: Hardware and software collaborative attack framework against neural networks. In *ISVLSI*. IEEE.
[37] Wenlong Liao, Birgitte Bak-Jensen, Jayakrishnan Radhakrishna Pillai, Yuelong Wang, and Yusen Wang. 2021. A review of graph neural networks and their applications in power systems. *Journal of Modern Power Systems and Clean Energy* (2021).
[38] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *RAID*.
[39] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In *SIGSAC*.
[40] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning attack on neural networks. In *NDSS*.
[41] 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*.

[42] Thien Duc Nguyen, Phillip Rieger, Huili Chen, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Shaza Zeitouni, et al. 2022. FLAME: Taming Backdoors in Federated Learning. In *31st USENIX Security Symposium*.

[43] Mustafa Safa Ozdayi, Murat Kantarcioglu, and Yulia R Gel. 2021. Defending against backdoors in federated learning with robust learning rate. In *AAAIConference on Artificial Intelligence*. 9268–9276.

[44] Soumyadeep Pal, Ren Wang, Yuguang Yao, and Sijia Liu. 2023. Towards Understanding How Self-training Tolerates Data Backdoor Poisoning. *arXiv preprint arXiv:2301.08751* (2023).

[45] Xudong Pan, Mi Zhang, Beina Sheng, Jiaming Zhu, and Min Yang. 2022. Hidden trigger backdoor attack on {NLP} models via linguistic style manipulation. In *31st USENIX Security Symposium (USENIX Security 22)*.

[46] Hengzhi Pei, Jinyuan Jia, Wenbo Guo, Bo Li, and Dawn Song. 2023. Textguard: Provable defense against backdoor attacks on text classification. In *NDSS*.

[47] 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* (2022).

[48] Fanchao Qi, Mukai Li, Yangyi Chen, Zhengyan Zhang, Zhiyuan Liu, Yasheng Wang, and Maosong Sun. 2021. Hidden Killer: Invisible Textual Backdoor Attacks with Syntactic Trigger. In *ACL*.

[49] Phillip Rieger, Thien Duc Nguyen, Markus Miettinen, and Ahmad-Reza Sadeghi. 2022. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. *arXiv preprint arXiv:2201.00763* (2022).

[50] Nirupam Roy, Haitham Hassanieh, and Romit Roy Choudhury. 2017. Backdoor: Making microphones hear inaudible sounds. In *Annual International Conference on Mobile Systems, Applications, and Services*.

[51] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. 2020. Hidden trigger backdoor attacks. In *AAAI*.

[52] Sina Sajadmanesh and Daniel Gatica-Perez. 2021. Locally private graph neural networks. In *CCS*.

[53] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. 2022. Dynamic Backdoor Attacks Against Machine Learning Models. In *EuroSP*.

[54] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[55] Yan Scholten, Jan Schuchardt, Simon Geisler, Aleksandar Bojchevski, and Stephan Günnemann. 2022. Randomized message-interception smoothing: Gray-box certificates for graph neural networks. *NeurIPS* (2022).

[56] Cong Shi, Tianfang Zhang, Zhuohang Li, Huy Phan, Tianming Zhao, Yan Wang, Jian Liu, Bo Yuan, and Yingying Chen. 2022. Audio-domain position-independent backdoor attack via unnoticeable triggers. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*.

[57] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. 2019. Can you really backdoor federated learning? *arXiv* (2019).

[58] 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 *AAAI*.

[59] Guanhong Tao, Zhenting Wang, Shiwei Feng, Guangyu Shen, Shiqing Ma, and Xiangyu Zhang. 2023. Distribution preserving backdoor attack in self-supervised learning. In *2024 IEEE Symposium on Security and Privacy (SP)*.

[60] Robert Tibshirani, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* (2001).

[61] Traffic prediction with advanced Graph Neural Networks. [n. d.]. https://deepmind.google/discover/blog/traffic-prediction-with-advanced-graph-neural-networks/.

[62] Brandon Tran, Jerry Li, and Aleksander Madry. 2018. Spectral signatures in backdoor attacks. In *NeurIPS*.

[63] Binghui Wang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2020. On Certifying Robustness against Backdoor Attacks via Randomized Smoothing. In *CVPR Workshop*.

[64] Binghui Wang, Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. 2021. Certified robustness of graph neural networks against adversarial structural perturbation. In *KDD*.

[65] 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 *IEEE International Conference on Data Mining*.

[66] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *IEEE S&P*.

[67] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. 2020. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*.

[68] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. 2020. Practical detection of trojan neural networks: Data-limited and data-free cases. In *ECCV*. 222–238.

[69] 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 *KDD*.

[70] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* (1998).

[71] Maurice Weber, Xiaojun Xu, Bojan Karlaš, Ce Zhang, and Bo Li. 2023. Rab: Provable robustness against backdoor attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1311–1328.

[72] Emily Wenger, Josephine Passananti, Arjun Nitin Bhagoji, Yuanshun Yao, Haitao Zheng, and Ben Y Zhao. 2021. Backdoor attacks against deep learning systems in the physical world. In *CVPR*.

[73] Peter Wills and François G Meyer. 2020. Metrics for graph comparison: a practitioner's guide. *Plos one* 15, 2 (2020), e0228728.

[74] 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.

[75] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020), 4–24.

[76] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. 2021. Graph backdoor. In *USENIX Security*.

[77] Chong Xiang, Arjun Nitin Bhagoji, Vikash Sehwag, and Prateek Mittal. 2021. {PatchGuard}: A provably robust defense against adversarial patches via small receptive fields and masking. In *USENIX Security*.

[78] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. 2021. Crfl: Certifiably robust federated learning against backdoor attacks. In *ICML*.

[79] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. 2019. Dba: Distributed backdoor attacks against federated learning. In *ICLR*.

[80] Chulin Xie, Yunhui Long, Pin-Yu Chen, Qinbin Li, Sanmi Koyejo, and Bo Li. 2023. Unraveling the Connections between Privacy and Certified Robustness in Federated Learning Against Poisoning Attacks. In *CCS*.

[81] Han Xie, Jing Ma, Li Xiong, and Carl Yang. 2021. Federated graph classification over non-iid graphs. In *NeurIPS*, Vol. 34.

[82] Han Xie, Li Xiong, and Carl Yang. 2023. Federated node classification over graphs with latent link-type heterogeneity. In *ACM Web Conference*.

[83] Shangyu Xie, Yan Yan, and Yuan Hong. 2023. Stealthy 3D Poisoning Attack on Video Recognition Models. *IEEE Trans. Dependable Secur. Comput.* 20, 2 (2023), 1730–1743. https://doi.org/10.1109/TDSC.2022.3163397

[84] 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 *ACSAC*. 684–698.

[85] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.

[86] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.

[87] Shenao Yan, Shen Wang, Yue Duan, Hanbin Hong, Kiho Lee, Doowon Kim, and Yuan Hong. 2024. An LLM-Assisted Easy-to-Trigger Backdoor Attack on Code Completion Models: Injecting Disguised Vulnerabilities against Strong Detection. In *33rd USENIX Security Symposium (USENIX Security 24)*.

[88] Han Yang, Binghui Wang, Jinyuan Jia, et al. 2024. GNNCert: Deterministic Certification of Graph Neural Networks against Adversarial Perturbations. In *The Twelfth International Conference on Learning Representations*.

[89] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. 2019. Latent Backdoor Attacks on Deep Neural Networks. In *CCS*.

[90] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.

[91] Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu Ming Yiu. 2021. Subgraph federated learning with missing neighbor generation. *NeurIPS* (2021).

[92] Xinyu Zhang, Hanbin Hong, Yuan Hong, Peng Huang, Binghui Wang, Zhongjie Ba, and Kui Ren. 2024. Text-crs: A generalized certified robustness framework against textual adversarial attacks. In *IEEE SP*.

[93] Zaixi Zhang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2022. FLDetector: Defending federated learning against model poisoning attacks via detecting malicious clients. In *KDD*.

[94] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. 2021. Backdoor attacks to graph neural networks. In *SACMAT*.

[95] Zhengming Zhang, Ashwinee Panda, Linyue Song, Yaoqing Yang, Michael Mahoney, Prateek Mittal, Ramchandran Kannan, and Joseph Gonzalez. 2022. Neurotoxin: Durable backdoors in federated learning. In *ICML*.

[96] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. 2020. Clean-label backdoor attacks on video recognition models. In *CVPR*. 14443–14452.

[97] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81.

[98] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: a comprehensive graph neural network platform. *Proceedings of the VLDB Endowment* 12, 12 (2019).