GNNGUARD: Defending Graph Neural Networks against Adversarial Attacks

Xiang Zhang
Harvard University
xiang_zhang@hms.harvard.edu

Marinka Zitnik
Harvard University
marinka@hms.harvard.edu

Abstract

Deep learning methods for graphs achieve remarkable performance across a variety of domains. However, recent findings indicate that small, unnoticeable perturbations of graph structure can catastrophically reduce performance of even the strongest and most popular Graph Neural Networks (GNNs). Here, we develop GNNGUARD, a general algorithm to defend against a variety of training-time attacks that perturb the discrete graph structure. GNNGUARD can be straightforwardly incorporated into any GNN. Its core principle is to detect and quantify the relationship between the graph structure and node features, if one exists, and then exploit that relationship to mitigate negative effects of the attack. GNNGUARD learns how to best assign higher weights to edges connecting similar nodes while pruning edges between unrelated nodes. The revised edges allow for robust propagation of neural messages in the underlying GNN. GNNGUARD introduces two novel components, the neighbor importance estimation, and the layer-wise graph memory, and we show empirically that both components are necessary for a successful defense. Across five GNNs, three defense methods, and four datasets, including a challenging human disease graph, experiments show that GNNGUARD outperforms existing defense approaches by 15.3% on average. Remarkably, GNNGUARD can effectively restore state-of-the-art performance of GNNs in the face of various adversarial attacks, including targeted and non-targeted attacks, and can defend against attacks on heterophily graphs.

1 Introduction

Deep learning on graphs and Graph Neural Networks (GNNs), in particular, have achieved remarkable success in a variety of application areas [1–5]. The key to the success of GNNs is the neural message passing scheme [6] in which neural messages are propagated along edges of the graph and typically optimized for performance on a downstream task. In doing so, the GNN is trained to aggregate information from neighbors for every node in each layer, which allows the model to eventually generate representations that capture useful node feature as well as topological structure information [7]. While the aggregation of neighbor nodes' information is a powerful principle of representation learning, the way that GNNs exchange that information between nodes makes them vulnerable to adversarial attacks [8].

Adversarial attacks on graphs, which carefully rewire the graph topology by selecting a small number of edges or inject carefully designed perturbations to node features, can contaminate local node neighborhoods, degrade learned representations, confuse the GNN to misclassify nodes in the graph, and can catastrophically reduce the performance of even the strongest and most popular GNNs [9, 10]. The lack of GNN robustness is a critical issue in many application areas, including those where adversarial perturbations can undermine public trust [11], interfere with human decision making [12], and affect human health and livelihoods [13]. For this reason, it is vital to develop GNNs that are

robust against adversarial attacks. While the vulnerability of machine learning methods to adversarial attacks has raised many concerns and has led to theoretical insights into robustness [14] and the development of effective defense techniques [9, 12, 15], adversarial attacks and defense on graphs remain poorly understood.

Present work. Here, we introduce GNNGUARD[1], an approach that can defend any GNN model against a variety of training-time attacks that perturb graph structure (Figure 1). GNNGUARD takes as input an existing GNN model. It mitigates adverse effects by modifying the GNN's neural message passing operators. In particular, it revises the message passing architecture such that the revised model is robust to adversarial perturbations while at the same time the model keeps it representation learning capacity. To this end, GNNGUARD develops two key components that estimate neighbor importance for every node and coarsen the graph through an efficient memory layer. The former component dynamically adjusts the relevance of nodes' local network neighborhoods, prunes likely fake edges, and assigns less weight to suspicious edges based on network theory of homophily [16]. The latter components stabilizes the evolution of graph structure by preserving, in part the memory from a previous layer in the GNN.
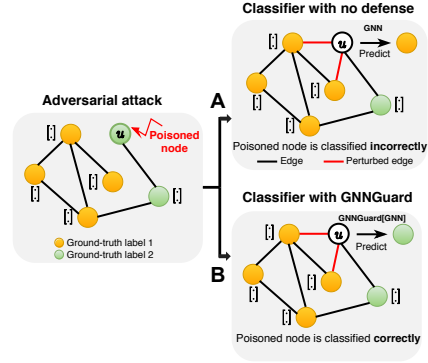

Figure 1: **A**. Small, adversarial perturbations of the graph structure and node features lead GNN to misclassify target u. **B**. The GNN, when integrated with GNNGUARD, correctly predicts u's label.

We compare GNNGUARD to three state-of-the-art GNN defenders across four datasets and under a variety of attacks, including direct targeted, influence targeted, and non-targeted attacks. Experiments show that GNNGUARD improves state-of-the-art methods by up to 15.3% in defense performance. Importantly, unlike existing GNN defenders [17–20], GNNGUARD is a general approach and can be effortlessly combined with any GNN architecture. To that end, we integrate GNNGUARD into five GNN models. Remarkably, results show that GNNGUARD can effectively restore state-of-the-art performance of even the strongest and most popular GNNs [3, 21, 7, 22, 23], thereby demonstrating broad applicability and relevance of GNNGUARD for graph machine-learning. Finally, GNNGUARD is the first technique that shows a successful defense on heterophily graphs [24]. In contrast, previous defenders, *e.g.*, [17–20], focused on homophily graphs [16]. Results show that GNNGUARD can be easily generalized to graphs with abundant structural equivalences where connected nodes can have different node features yet similar structural roles within their local topology [25].

2   Related Work

Adversarial attacks in continuous and discrete space. Adversarial attacks on machine learning have received increasing attention in recent years [14, 26, 27]. The attackers add small perturbations on the samples to completely alter the output of the machine learning model. The deliberately manipulated perturbations are often designed to be unnoticeable. Modern studies have shown that machine leaning models, especially deep neural networks, are highly fragile to adversarial attacks [13, 28, 29]. The majority of existing works focus on grid data or independent samples [30] whilst a few work investigate adversarial attack on graphs.

Adversarial attacks on graphs. Based on the goal of the attacker, adversarial attacks on graphs [31, 32] can be divided into poisoning attacks (*e.g.*, Nettack [8]) that perturb the graph in training-time and evasion attacks (*e.g.*, RL-S2V [32]) that perturb the graph in testing-time. GNNGUARD is designed to improve robustness of GNNs against poisoning attacks. There are two types of poisoning attacks: a targeted attack and a non-targeted attack [33]. The former deceives the model to misclassify a specific node (*i.e.*, target node) [8] while the latter degrades the overall performance of the trained model [30]. The targeted attack can be categorized into direct targeted attack where the attacker perturbs edges touching the target node and the influence targeted attack where the attacker only manipulates edges of the target node's neighbors. Nettack [8] generates perturbations by modifying graph structure (*i.e.*, structure attack) and node attributes (*i.e.*, feature attack) such that perturbations maximally destroy

---

[1]Code and datasets are available at https://github.com/mims-harvard/GNNGuard.

downstream GNN's predictions. Bojcheshki *et al.* [34] derive adversarial perturbations that poison the graph structure. Similarly, Zügner *et al.* [30] propose a non-targeted poisoning attacker by using meta-gradient to solve bi-level problem. In contrast, our GNNGUARD is a defense approach that inspects the graph and recovers adversarial perturbations.

Defense on graphs. While deep learning on graphs has shown exciting results in a variety of applications [6, 23, 35–37], little attention has been paid to the robustness of such models, in contrast to an abundance of research for image (*e.g.*, [38]) and text (*e.g.*, [39]) adversarial defense. We briefly overview the state-of-the-art defense methods on graphs. GNN-Jaccard [17] is a defense approach that pre-processes the adjacency matrix of the graph to identify the manipulated edges. While GNN-Jaccard can defend targeted adversarial attacks on known and already existing GNNs, there has also been work on novel, robust GNN models. For example, RobustGCN [19] is a novel GNN that adopts Gaussian distributions as the hidden representations of nodes in each convolutional layer to absorb the effect of an attack. Similarly, GNN-SVD [18] uses a low-rank approximation of adjacency matrix that drops noisy information through an SVD decomposition. Tang *et al.* [20] improve the robustness of GNNs against poisoning attack through transfer learning but has a limitation that requires several unperturbed graphs from the similar domain during training. However, all these approaches have drawbacks (see Section 4.3) that prevent them from realizing their potential for defense to the fullest extent. For instance, none of them consider how to defend heterophily graphs against adversarial attacks. GNNGUARD eliminates these drawbacks, successfully defending targeted and non-targeted poisoning attacks on any GNN without decreasing its accuracy.

3 Background and Problem Formulation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ denote a graph where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges and $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}, \mathbf{x}_u \in \mathbb{R}^M$ is the M-dimensional node feature for node $u \in \mathcal{V}$. Let $N = |\mathcal{V}|$ and $E = |\mathcal{E}|$ denote the number of nodes and edges, respectively. Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ denote an adjacency matrix whose element $\mathbf{A}_{uv} \in \{0, 1\}$ indicates existence of edge e_{uv} that connects node u and v. We use $\mathcal{N}_u$ to denote immediate neighbors of node u, including the node itself ($u \in \mathcal{N}_u$). We use $\mathcal{N}_u^*$ to indicate u's neighborhood, excluding the node itself ($u \notin \mathcal{N}_u^*$). Without loss of generality, we consider node classification task, wherein a GNN f classifies nodes into C labels. Let $\hat{y}_u = f_u(\mathcal{G})$ denote prediction for node u, and let $y_u \in \{1, \ldots, C\}$ denote the associated ground-truth label for node u. To degrade the performance of f, an adversarial attacker perturbs edges in $\mathcal{G}$, resulting in the perturbed version of $\mathcal{G}$, which we call $\mathcal{G}' = (\mathcal{V}, \mathcal{E}', \mathbf{X})$ ($\mathbf{A}'$ is adjacency matrix of $\mathcal{G}'$).

Background on graph neural networks. Graph neural networks learns compact, low-dimensional representations, *i.e.*, embeddings, for nodes such that representation capture nodes' local network neighborhoods as well as nodes' features [6, 3, 40]. The learned embeddings can be used for a variety of downstream tasks [3]. Let $\boldsymbol{h}_u^k \in \mathbb{R}^{D_k}$ denote the embedding of node u in the k-th layer of GNN, $k = \{1, \ldots, K\}$. The D_k stands for the dimension of $\boldsymbol{h}_u^k$. Note that $\boldsymbol{h}_u^0 = \boldsymbol{x}_u$. The computations in the k-th layer consist of a message-passing function MSG, an aggregation function AGG, and an update function UPD. This means that a GNN f can be specified as $f = (\text{MSG}, \text{AGG}, \text{UPD})$ [6, 36]. Given a node u and its neighbor $v \in \mathcal{N}_u$, the messaging-passing function MSG specifies what neural message $\boldsymbol{m}_{uv}^k$ needs to be propagated from v to u. The message is calculated by $\boldsymbol{m}_{uv}^k = \text{MSG}(\boldsymbol{h}_u^k, \boldsymbol{h}_v^k, \boldsymbol{A}_{uv})$, where MSG receives node embeddings of u and v along with their connectivity information e_{uv}. This is followed by the aggregation function AGG that aggregates all messages received by u. The aggregated message $\hat{\boldsymbol{m}}_u^k$ is computed by $\hat{\boldsymbol{m}}_u^k = \text{AGG}(\{\boldsymbol{m}_{uv}^k; v \in \mathcal{N}_u^*\})$. Lastly, the update function UPD combines u's embedding $\boldsymbol{h}_u^k$ and the aggregated message $\hat{\boldsymbol{m}}_u^k$ to generate the embedding for next layer as $\boldsymbol{h}_u^{k+1} = \text{UPD}(\boldsymbol{h}_u^k, \hat{\boldsymbol{m}}_u^k)$. The final node representation for u is $\boldsymbol{h}_u^K$, *i.e.*, the output of the K-th layer.

Background on poisoning attacks. Attackers try to fool a GNN by corrupting the graph topology during training [41]. The attacker carefully selects a small number of edges and manipulates them through perturbation and rewiring. In doing so, the attacker aims to fool the GNN into making incorrect predictions [20]. The attacker finds optimal perturbation $\boldsymbol{A}'$ through optimization [30, 8]:

$$\underset{\boldsymbol{A}' \in \mathcal{P}_\Delta^{\mathcal{G}}}{\text{argmin}} \ \mathcal{L}_{\text{attack}}(f(\boldsymbol{A}', \boldsymbol{X}; \Theta^*), \boldsymbol{y}) \quad \text{s.t.} \quad \Theta^* = \underset{\Theta}{\text{argmin}} \ \mathcal{L}_{\text{predict}}(f(\boldsymbol{A}', \boldsymbol{X}; \Theta), \boldsymbol{y}) \tag{1}$$

where $\boldsymbol{y}$ denotes ground-truth labels, $\mathcal{L}_{\text{attack}}$ denotes the attacker's loss function, and $\mathcal{L}_{\text{predict}}$ denotes GNN's loss. The Θ^* refers to optimal parameters and $f(\boldsymbol{A}', \boldsymbol{X}; \Theta^*)$ is prediction of f with parameters

Θ^* on the perturbed graph $\boldsymbol{A}'$ and node features $\boldsymbol{X}$. To ensure that attacker perturbs only a small number of edges, a budget Δ is defined to constrain the number of perturbed edges: $||\boldsymbol{A}' - \boldsymbol{A}||_0 \leqslant \Delta$ and $\mathcal{P}_\Delta^{\mathcal{G}}$ are perturbations that fit into budget Δ. Let $\mathcal{T}$ be target nodes that are intended to be mis-classified, and let $\mathcal{A}$ be attacker nodes that are allowed to be perturbed. We consider three types of attacks. (1) <mark>Direct targeted attacks.</mark> The attacker aims to destroy prediction for target node u by manipulating the incident edges of u [8, 17]. Here, $\mathcal{T} = \mathcal{A} = \{u\}$. (2) <mark>Influence targeted attacks.</mark> The attacker aims to destroy prediction for target node u by perturbing the edges of u's neighbors. Here, $\mathcal{T} = \{u\}$ and $\mathcal{A} = \mathcal{N}_u^*$. (3) <mark>Non-targeted attacks.</mark> The attacker aims to degrade overall GNN classification performance [30, 42]. Here, $\mathcal{T} = \mathcal{A} = \mathcal{V}_{\text{test}}$ where $\mathcal{V}_{\text{test}}$ denotes the test set.

3.1 GNNGUARD: Problem Formulation

GNNGUARD is a defense mechanism that is easy to integrate into any GNN f, resulting in a new GNN f' that is robust to poisoning attacks. This means that f' can make correct predictions even when trained on poisoned graph $\mathcal{G}'$. Given a GNN $f = (\text{MSG}, \text{AGG}, \text{UPD})$, GNNGUARD will return a new GNN $f' = (\text{MSG}', \text{AGG}', \text{UPD}')$, where MSG' is the message-passing function, AGG' is the aggregation function, and UPD' is the update function. The f' solves the following defense problem.

Problem (Defense Against Poisoning Attacks on Graphs). *In a poisoning attack, the attacker injects adversarial edges in $\mathcal{G}$, meaning that the attack changes training data, which can decrease the performance of GNN considerably. Let $\mathcal{G}'$ denote the perturbed version of $\mathcal{G}$ that is poisoned by the attack. We seek GNN f' such that for any node $u \in \mathcal{G}'$:*

$$\min f_u'(\mathcal{G}') - f_u(\mathcal{G}), \tag{2}$$

where $f_u'(\mathcal{G}') = \hat{y}_u'$ is the prediction when GNN f' is trained on $\mathcal{G}'$. Here, $f_u(\mathcal{G}) = \hat{y}_u$ denotes a hypothetical prediction that the GNN would made if it had access to clean graph $\mathcal{G}$.

It is worth noting that, in this paper, we learn a defense mechanism for semi-supervised node classification. GNNGUARD is a general framework for defending any GNN on various graph mining tasks such as link prediction. Since there exists a variety of GNNs that achieve competitive performance on $\mathcal{G}$, an intuitive idea is to force $f_u'(\mathcal{G}')$ to approximate $f_u(\mathcal{G})$ and, in doing so, ensure that f' will make correct predictions on $\mathcal{G}'$. For this reason, we design f' to learn neural messages on $\mathcal{G}'$ that, in turn, are similar to the messages that a hypothetical f would learn on $\mathcal{G}$. However, since it is impossible to access clean graph $\mathcal{G}$, Eq. (2) can not be directly optimized. The key to restore the structure of $\mathcal{G}$ is to design a message-passing scheme that can detect fake edges, block them and then attend to true, unperturbed edges. To this end, the impact of perturbed edges in $\mathcal{G}'$ can be mitigated by manipulating the flow of neural messages and thus, the structure of $\mathcal{G}$ can be restored.

4 GNNGUARD

Next, we describe GNNGUARD, our GNN defender against poisoning attacks. Recent studies [31, 17] found that most damaging attacks add fake edges between nodes that have different features and labels. Because of that, the core defense principle of GNNGUARD is to detect such fake edges and alleviate their negative impact on prediction by remove them or assigning them lower weights in neural message passing. GNNGUARD has two key components: <mark>(1) neighbor importance estimation, and (2) layer-wise graph memory,</mark> the first component being an essential part of a robust GNN architecture while the latter is designed to smooth the defense.

4.1 Neighbor Importance Estimation

GNNGUARD estimates an importance weight for every edge e_{uv} to quantify how relevant node u is to another node v in the sense that it allows for successful routing of GNN's messages. In contrast to attention mechanisms (*e.g.*, GAT [21, 43]), GNNGUARD determines importance weights based on the hypothesis that similar nodes (*i.e.*, nodes with similar features or similar structural roles) are more likely to interact than dissimilar nodes [16]. To this end, we <mark>quantify similarity s_{uv}^k between u and its neighbor v in the k-th layer of GNN</mark> as follows:

$$s_{uv}^k = d(\boldsymbol{h}_u^k, \boldsymbol{h}_v^k), \qquad d(\boldsymbol{h}_u^k, \boldsymbol{h}_v^k) = (\boldsymbol{h}_u^k \odot \boldsymbol{h}_v^k)/(||\boldsymbol{h}_u^k||_2 ||\boldsymbol{h}_v^k||_2), \tag{3}$$

where d is a similarity function and $\odot$ denotes dot product. In this work, we use cosine similarity to calculate d [44]. <u>In homophily graphs, s_{uv}^k measures the similarity between node features; in</u>
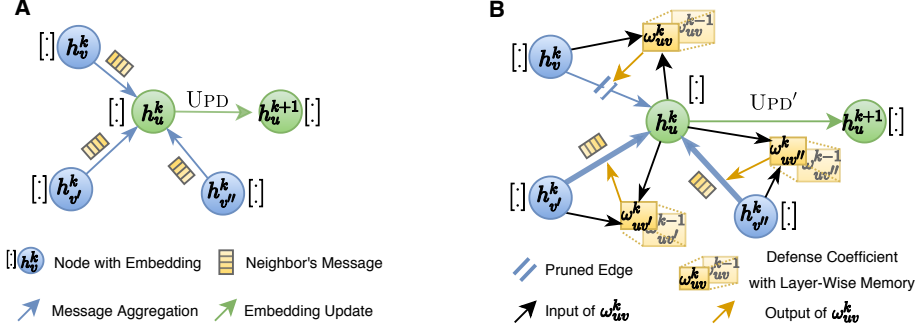
Figure 2: A. Illustration of neural message passing in u's local network neighborhood in the k-th layer of GNN f. **B.** The message flow in f', which is the GNN f endowed by GNNGUARD defense. We first calculate defense coefficients ω_{uv}^k based on node representations $\boldsymbol{h}_u^k$ and $\boldsymbol{h}_v^k$. The defense coefficients are then used to control the message stream such as blocking the message from v but strengthening messages from v' and v''. Thick blue arrow indicates the higher weights during message aggregation. To stabilize the evolution of graph structure, current defense coefficients (*e.g.*, ω_{uv}^k) keep a partial memory of the previous layer (*e.g.*, ω_{uv}^{k-1}).

heterophily graphs, it measures the similarity of nodes' structural roles. Larger similarity s_{uv}^k indicates that edge e_{uv} is strongly supported by node features (or local topology) of the edge's endpoints. We normalize s_{uv}^k at the node-level within u's neighborhood $\mathcal{N}_u$. The problem here is to specify what is the similarity of the node to itself. We normalize node similarities as:

$$\alpha_{uv}^k = \begin{cases} s_{uv}^k / \sum_{v \in \mathcal{N}_u^*} s_{uv}^k \times \hat{N}_u^k / (\hat{N}_u^k + 1) & \text{if} \quad u \neq v \\ 1/(\hat{N}_u^k + 1) & \text{if} \quad u = v, \end{cases} \tag{4}$$

where $\hat{N}_u^k = \sum_{v \in \mathcal{N}_u^*} ||s_{uv}^k||_0$. We refer to α_{uv}^k as an importance weight representing the contribution of node v towards node u in the GNN's passing of neural messages in poisoned graph $\mathcal{G}'$. In doing so, GNNGUARD assigns small importance weights to suspicious neighbors, which reduce the interference of suspicious nodes in GNN's operation. Further, to alleviate the impact of fake edges, we prune edges that are likely forged. Building on network homophily and findings [17] that fake edges tend to connect dissimilar nodes, we prune edges using importance weights. For that, we define a characteristic vector $\boldsymbol{c}_{uv}^k = [\alpha_{uv}^k, \alpha_{vu}^k]$ describing edge e_{uv}. Although $s_{uv}^k = s_{vu}^k$, it is key to note that $\alpha_{uv}^k \neq \alpha_{vu}^k$ because of self-normalization in Eq. (4). GNNGUARD calculates edge pruning probability for e_{uv} through a non-linear transformation as $\sigma(\boldsymbol{c}_{uv}^k \boldsymbol{W})$. Then, it maps the pruning probability to a binary indicator $1_{P_0} : \sigma(\boldsymbol{c}_{uv}^k \boldsymbol{W})$ where P_0 is a user-defined threshold:

$$1_{P_0}(\sigma(\boldsymbol{c}_{uv}^k \boldsymbol{W})) = \begin{cases} 0 & \text{if} \quad \sigma(\boldsymbol{c}_{uv}^k \boldsymbol{W}) < P_0 \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

Finally, we prune edges by updating importance weight α_{uv}^k to $\hat{\alpha}_{uv}^k$ as follows:

$$\hat{\alpha}_{uv}^k = \alpha_{uv}^k 1_{P_0}(\sigma(\boldsymbol{c}_{uv}^k \boldsymbol{W})), \tag{6}$$

meaning that the perturbed edges connecting dissimilar nodes will likely be ignored by the GNN.

4.2 Layer-Wise Graph Memory

Neighbor importance estimation and edge pruning change the graph structure between adjacent GNN layers. This can destabilize GNN training, especially if a considerable number of edges gets pruned in a single layer (*e.g.*, due to the weight initialization). To allow for robust estimation of importance weights and smooth evolution of edge pruning, we use layer-wise graph memory. This unit, applied at each GNN layer, keeps partial memory of the pruned graph structure from the previous layer (Figure 2). We define layer-wise graph memory as follows:

$$\omega_{uv}^k = \beta \omega_{uv}^{k-1} + (1 - \beta)\hat{\alpha}_{uv}^k, \tag{7}$$

where ω_{uv}^k represents defense coefficient for edge e_{uv} in the k-th layer and β is a memory coefficient specifying memory, *i.e.*, the amount of information from the previous layer that should be kept in the current layer. Memory coefficient $\beta \in [0, 1]$ is a learnable parameter and is set to $\beta = 0$ in the first GNN layer, meaning that $\omega_{uv}^0 = \hat{\alpha}_{uv}^0$. Using defense coefficients, GNNGUARD controls information flow across all neural message passing layers. It strengthens messages from u's neighbors with higher defense coefficients and weakens messages from u's neighbors with lower defense coefficients.

4.3 Overview of GNNGUARD

GNNGUARD is shown in Algorithm 1. The method is easy to plug into an existing GNN to defend the GNN against poisoning attacks. Given a GNN $f = (\text{MSG}, \text{AGG}, \text{UPD})$, GNNGUARD formulates a revised version of it, called $f' = (\text{MSG}', \text{AGG}', \text{UPD}')$. In each layer, f' takes current node representations and (possibly attacked) graph $\mathcal{G}'$. It estimates importance weights $\hat{\alpha}_{uv}$ and generates defense coefficients ω_{uv} by combining importance weights from the current layer and defense coefficients from the previous layer. In summary, aggregation function AGG' in layer k is: $\text{AGG}' = \text{AGG}(\{\omega_{uv}^k \odot \boldsymbol{m}_{uv}^k; v \in \mathcal{N}_u^*\})$. The update function UPD' is: $\text{UPD}' = \text{UPD}(\omega_{uu}^k \odot \boldsymbol{h}_u^k, \text{AGG}(\{\omega_{uv}^k \odot \boldsymbol{m}_{uv}^k; v \in \mathcal{N}_u^*\}))$. The message function MSG' remains unchanged $\text{MSG}' = \text{MSG}$ as neural messages are specified by the original GNN f. Taken together, the guarded f' attends differently to different node neighborhoods and propagates neural information only along most relevant edges. Our derivations here are for undirected graphs with node features but can be extended to directed graphs and edge features (*e.g.*, include them into calculation of characteristic vectors).

Algorithm 1: GNNGUARD.

Input: GNN model of interest $f = (\text{MSG}, \text{AGG}, \text{UPD})$; Poisoned graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}', \boldsymbol{X})$, ($\boldsymbol{A}'$ is adjacency matrix of $\mathcal{E}'$); Trainable parameters $\Theta, \boldsymbol{W}$, and β

Initialize parameters $\Theta, \boldsymbol{W}$, and β; initialize node representations $\boldsymbol{h}_u^0 = \boldsymbol{x}_u \; \forall u \in \mathcal{V}$

for layer $k \leftarrow 1$ **to** K **do**

    for $u \in \mathcal{V}$ **do**

        Calculate α_{uv}^k using Eq. (4) for all $v \in \mathcal{N}_u$ // Neighbor Importance Estimation

        $\boldsymbol{c}_{uv}^k = [\alpha_{uv}^k, \alpha_{vu}^k]$

        $\hat{\alpha}_{uv}^k = \alpha_{uv}^k 1_{P_0}(\sigma(\boldsymbol{c}_{uv}^k \boldsymbol{W}))$ using Eq. (6)

        $\omega_{uv}^k = \beta\omega_{uv}^{k-1} + (1-\beta)\hat{\alpha}_{uv}^k$ using Eq. (7)   // Layer-Wise Graph Memory

        $\boldsymbol{m}_{uv}^k = \text{MSG}'(\boldsymbol{h}_u^k, \boldsymbol{h}_v^k, \boldsymbol{A}_{uv}')$ using Section 4.3   // Neural Message Passing

        $\hat{\boldsymbol{m}}_u^k = \text{AGG}'(\{\omega_{uv}^k \odot \boldsymbol{m}_{uv}^k; v \in \mathcal{N}_u^*\})$ using Section 4.3

        $\boldsymbol{h}_u^{k+1} = \text{UPD}'(\omega_{uu}^k \odot \boldsymbol{h}_u^k, \hat{\boldsymbol{m}}_u^k)$ using Section 4.3

    end

end

Any GNN model. State-of-the-art GNNs use neural message passing comprising of MSG, AGG, and UPD functions. As we demonstrate in experiments, GNNGUARD can defend such GNN architectures against adversarial attacks. GNNGUARD works with many GNNs, including Graph Convolutional Network (GCN) [3], Graph Attention Network (GAT) [21], Graph Isomorphism Network (GIN) [7], Jumping Knowledge (JK-Net) [22], GraphSAINT [23], GraphSAGE [40], and SignedGCN [45].

Computational complexity. GNNGUARD is practically efficient because it exploits the sparse structure of real-world graphs. The time complexity of neighbor importance estimation is $\mathcal{O}(D_k E)$ in layer k, where D_k is the embedding dimensionality and E is the graph size, and the complexity of layer-wise graph memory is $\mathcal{O}(E)$. This means that time complexity of GNNGUARD grows linearly with the size of the graph as node embeddings are low-dimensional, $D_k \ll E$. Finally, the time complexity of a GNN endowed with GNNGUARD is on the same order as that of the GNN itself.

Further related work on adversarial defense for graphs. We briefly contrast GNNGUARD with existing GNN defenders. Compared to GNN-Jaccard [17], which examines fake edges as a GNN preprocessing step, GNNGUARD dynamically updates defense coefficients at every GNN layer for defense. In contrast to RobustGCN [19], which is limited to GCN, a particular GNN variant, and is challenging to use with other GNNs, GNNGUARD provides a generic mechanism that is easy to use with many GNN architectures. Further, in contrast to GNN-SVD [18], which uses only graph structure for defense, GNNGUARD takes advantage of information encoded in both node features and graph structure. Also, [18] is designed specifically for the Nettack attacker [8] and so is less versatile. Another technique [20] uses transfer learning to detect fake edges. While that is an interesting idea, it requires a large number of clean graphs from the same domain to successfully train the transfer model. On the contrary, GNNGUARD takes advantage of correlation between node features and graph structure and does not need any external data. Further, recent studies (*e.g.*, [46, 47]) focus on theoretical certificates for GNN robustness instead of defense mechanisms. That is an important but orthogonal direction to this paper, where the focus is on a practical adversarial defense framework.

Table 1: Defense performance (multi-class classification accuracy) against direct targeted attacks.

Model	Dataset	No Attack	Attack	GNN-Jaccard	RobustGCN	GNN-SVD	GNNGUARD
GCN	Cora	0.826	0.250	0.525	0.215	0.475	**0.705**
	Citeseer	0.721	0.175	0.435	0.230	0.615	**0.720**
	ogbn-arxiv	0.667	0.235	0.305	0.245	0.370	**0.425**
	DP	0.682	0.215	0.340	0.315	0.395	**0.430**
GAT	Cora	0.827	0.245	0.295	0.215	0.365	**0.625**
	Citeseer	0.718	0.265	0.575	0.230	0.575	**0.765**
	ogbn-arxiv	0.669	0.210	0.355	0.245	0.445	**0.520**
	DP	0.714	0.205	0.320	0.315	0.335	**0.445**
GIN	Cora	0.831	0.270	0.375	0.215	0.375	**0.645**
	Citeseer	0.725	0.285	0.570	0.230	0.570	**0.755**
	ogbn-arxiv	0.661	0.315	0.425	0.245	0.475	**0.640**
	DP	0.719	0.245	0.410	0.315	0.405	**0.460**
JK-Net	Cora	0.834	0.305	0.445	0.215	0.425	**0.690**
	Citeseer	0.724	0.275	0.615	0.230	0.610	**0.775**
	ogbn-arxiv	0.678	0.335	0.375	0.245	0.325	**0.635**
	DP	0.726	0.220	0.335	0.315	0.360	**0.450**
Graph SAINT	Cora	0.821	0.225	0.535	0.235	0.460	**0.695**
	Citeseer	0.716	0.195	0.470	0.350	0.395	**0.770**
	ogbn-arxiv	0.683	0.245	0.365	0.245	0.315	**0.375**
	DP	0.739	0.205	0.315	0.295	0.330	**0.485**

5 Experiments

We start by describing the experimental setup. We then present how GNNGUARD compares to existing GNN defenders (Section 5.1), provide an ablation study and a case study on citation network (Section 5.2), and show how GNNGUARD can be used with heterophily graphs (Section 5.3).

Datasets. We test GNNGUARD on four graphs. We use two citation networks with undirected edges and binary features: Cora [48] and Citeseer [49]. We also consider a directed graph with numeric node features, ogbn-arxiv [50], representing a citation network of CS papers published between 1971 and 2014. We use a Disease Pathway (DP) [51] graph with continuous features describing a system of interacting proteins whose malfunction collectively leads to diseases. The task is to predict for every protein node what diseases the protein might cause. Details are in Appendix D.

Setup. (1) <u>Generating adversarial attacks.</u> We compare our model to baselines under three kinds of adversarial attacks: direct targeted attack (Nettack-Di [8]), influence targeted attack (Nettack-In [8]), and non-targeted attack (Mettack [30]). In Mettack, we set the perturbation rate as 20% (*i.e.*, $\Delta = 0.2E$) with 'Meta-Self' training strategy. In Nettack-Di, $\Delta = \hat{N}_u^0$. In Nettack-In, we perturb 5 neighbors of the target node and set $\Delta = \hat{N}_v^0$ for all neighbors. In the targeted attack, we select 40 correctly classified target nodes (following [8]): 10 nodes with the largest classification margin, 20 random nodes, and 10 nodes with the smallest margin. We run the whole attack and defense procedure for each target node and report average classification accuracy. (2) <u>GNNs.</u> We integrate GNNGUARD with five GNNs (GCN [3], GAT [21], GIN [7], JK-Net [22], and GraphSAINT [23]) and present the defense performance against adversarial attacks. (3) <u>Baseline defense algorithms.</u> We compare GNNGUARD to three state-of-the-art graph defenders: GNN-Jaccard [17], RobustGCN [19], and GNN-SVD [18]. Hyperparameters and model architectures are in Appendix E.

5.1 Results: Defense Against Targeted and Non-Targeted Attacks

(1) <u>Results for direct targeted attacks.</u> We observe in Table 1 that Nettack-Di is a strong attacker and dramatically cuts down the performance of all GNNs (cf. "Attack" vs. "No Attack" columns). However, the proposed GNNGUARD outperforms state-of-art defense methods by 15.3% in the accuracy on average. Further, it successfully restores the performance of GNNs to the level comparable to when there is no attack. We also observe that RobustGCN fails to defend against Nettack-Di, possibly because the Gaussian layer in RobustGCN cannot absorb big effects when all fake edges are in the vicinity of a target node. In contrast, GNN-SVD works well here because it is sensitive to high-rank noise caused by the perturbation of many edges that are incident to a single node. (2) <u>Results for influence targeted attacks.</u> As shown in Table 2, GNNGUARD achieves the best classification accuracy comparing to other baseline defense algorithms. Taking a closer look at the results, we we can find that Nettack-In is relatively less threaten than Nettack-Di indicating part of the perturbed information was scattered during neural message passing. (3) <u>Results for non-targeted attacks.</u> Table 2 shows that Mettack has a considerable negative impact on GNN performance, decreasing the

Table 2: Defense performance (multi-class classification accuracy) against influence targeted (top) and non-targeted (bottom) attacks. Tables with full results for other GNN models are in Appendix A (influence targeted attacks) and Appendix B (non-targeted attacks).

Model	Dataset	No Attack	Attack	GNN-Jaccard	RobustGCN	GNN-SVD	GNNGUARD
GIN	Cora	0.831	0.525	0.635	0.605	0.615	**0.775**
	Citeseer	0.725	0.480	0.675	0.575	0.630	**0.845**
	ogbn-arxiv	0.661	0.570	0.605	0.620	0.525	**0.710**
	DP	0.719	0.505	0.585	0.565	0.605	**0.695**
GIN	Cora	0.831	0.588	0.702	0.571	0.692	**0.722**
	Citeseer	0.725	0.565	0.638	0.583	0.615	**0.711**
	ogbn-arxiv	0.661	0.424	0.459	0.436	0.459	**0.486**
	DP	0.719	0.537	0.559	0.528	0.513	**0.571**

Table 3: Ablation study on ogbn-arxiv dataset. 'Memory' denotes layer-wise graph memory (Section 4.2) while 'pruning' denotes edge pruning operation (Section 4.1).

Model	No Defense	GNNGUARD w/o pruning	GNNGUARD w/o memory	Full GNNGUARD
GCN	0.235	0.350	0.405	**0.425**
GAT	0.210	0.315	0.475	**0.520**
GIN	0.315	0.540	0.610	**0.640**
JK-Net	0.335	0.565	0.625	**0.635**
GraphSAINT	0.245	0.305	0.360	**0.375**

accuracy of even the strongest GNN by 18.7% on average. Moreover, we see that GNNGUARD achieves a competitive performance and outperforms baselines in 19 out of 20 settings. In summary, experiments show the GNNGUARD consistently outperforms all baseline defense techniques. Further, GNNGUARD can defend a variety of GNNs against different types of attacks, indicating that GNNGUARD is a powerful GNN defender against adversarial poisoning.

5.2 Results: Ablation Study and Inspection of Defense Mechanism

(1) <u>Ablation study.</u> We conduct an ablation study to evaluate the necessity of every component of GNNGUARD. For that, we took the largest dataset (ogbn-arxiv) and the most threatening attack (Nettack-Di) as an example. Results are in Table 3. We observe that full GNNGUARD behaves better and has smaller standard deviation than limited GNNGUARD w/o layer-wise graph memory, suggesting that graph memory can contribute to defense performance and stabilize model training. (2) <u>Node classification on clean datasets.</u> In principle, we don't know if the input graph has been attacked or not. Because of that, it is important that a successful GNN defender can deal with poisoned graphs and also does not harm GNN performance on clean datasets. Appendix C shows classification accuracy of GNNs on clean graphs. Across all datasets, we see that, when graphs are not attacked, GNNs with turned-on GNNGUARD achieve performance comparable to that of GNNs alone, indicating that GNNGUARD will not weaken learning ability of GNNs when there is no attack. (3) <u>Defense under different attack intensity.</u> We investigate defense performance as a function of attack strength. Table 4 shows attack and defense results on Cora under Mettack with increasing attack rates. It is expected that GCN accuracy decays as attacks intensify. Nevertheless, GNNGUARD effectively defends GCN and can do so especially under strong attack. GCN with GNNGUARD outperforms GCN with no defense by 19.8% when 25% of the edges are attacked.

A case study of attack and defense. We report an example of attack and defense illustrating how GNNGUARD works. Let's examine the paper "TreeP: A Tree Based P2P Network Architecture" by Hudzia *et al.* [52] that received four citations. The topic/label of this paper (*i.e.*, node u in ogbn-arxiv graph $\mathcal{G}$) and its cited works (*i.e.*, neighbors) is *Internet Technology (IT)*. GIN [7] trained on clean ogbn-arxiv graph makes a correct prediction for the paper with high confidence, $f_u(\mathcal{G}) = 0.536$. Then, we poison the paper using Nettack-Di attacker, which adds four fake citations between the paper and some very dissimilar papers from the *Artificial Intelligence (AI)* field. We re-trained GIN on perturbed graph $\mathcal{G}'$ and found the resulting classifier misclassifies the paper [52] into topic *AI* with confidence of $f_u(\mathcal{G}') = 0.201$, which is high on this prediction task with 40 distinct topics/labels. This fragility is especially worrisome as the attacker has only injected four fake citations and was already able to easily fool a state-of-the-art GNN. We then re-trained GIN with GNNGUARD defense on the same perturbed graph and, remarkably, the paper [52] was correctly classified to *IT* with high confidence ($f'_u(\mathcal{G}') = 0.489$) even after the attack. This example illustrates how easily an adversary can fool a GNN on citation networks.

Table 4: Attack and defense accuracy on Cora dataset.

Attack Rate (% edges)	No Defense	GNNGUARD
5%	0.771	**0.776**
10%	0.716	**0.749**
15%	0.651	**0.739**
20%	0.578	**0.714**
25%	0.531	**0.729**
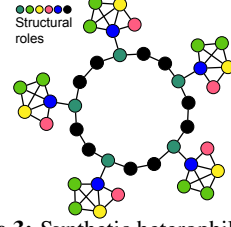
Figure 3: Synthetic heterophily graph. Node colors indicate structural roles.

Table 5: Performances of heterophily graphs. N/A means the method does not apply to the heterophily setting.

Model	No Attack	Attack	GNN-Jaccard	RobustGCN	GNN-SVD	GNNGUARD
GCN	0.834	0.385	N/A	0.525	0.595	**0.715**
GAT	0.851	0.325	N/A	0.575	0.635	**0.770**
GIN	0.891	0.450	N/A	0.575	0.650	**0.775**
JK-Net	0.889	0.425	N/A	0.575	0.640	**0.735**
GraphSAINT	0.876	0.415	N/A	0.575	0.625	**0.755**

5.3 Results: Defense of Heterophily Graphs

GNNGUARD with heterophily. Next, we evaluate GNNGUARD on graphs with structural roles, a prominent type of heterophily. To measure the local topology of nodes, we use graphlet degree vectors [53] which reflect nodes' structural properties, e.g., triangles, betweenness, stars, etc. To do so, we revise Eq. (3) by replacing embeddings for nodes u and v (*i.e.*, $\boldsymbol{h}_u^k$ and $\boldsymbol{h}_v^k$) with their graphlet degree vectors (*i.e.*, $\bar{\boldsymbol{h}}_u^k$ and $\bar{\boldsymbol{h}}_v^k$), yielding the learned similarity s_{uv}^k that quantifies structural similarity between u and v. The graphlet degree vectors are calculated using the orbit counting algorithm [54], are independent of node attributes, and provide a highly constraining measure of local graph topology. We test whether the revised GNNGUARD can defend GNNs trained on graphs with heterophily.

Experiments. We synthesize cycle graphs with attached house shapes (see an example in Figure 3), where labels are defined by nodes' structural roles [25]. The synthetic graphs contain 1,000 nodes (no node features, but each node has a 73-dimensional graphlet vector), 3,200 undirected edges, and 6 node labels (*i.e.*, distinct structural roles). We use the strongest performing attacker Nettack-Di to manipulate each graph. Results are shown in Table 5. We find that GNNGUARD achieves the highest accuracy of 77.5%. In contrast, GNN performance without any defense is at most 45.0%. GNNGUARD outperforms the strongest baseline by 19.2%, which is not surprising as existing GNN defenders cannot defend graphs with heterophily. Taken together, these results show the effectiveness of GNNGUARD, when used together with an appropriate similarity function, for graphs with either homophily or heterophily.

6 Conclusion

We introduce GNNGUARD, an algorithm for defending graph neural networks (GNN) against poisoning attacks, including direct targeted, influence targeted, and non-targeted attacks. GNNGUARD mitigates adverse effects by modifying neural message passing of the underlying GNN. This is achieved through the estimation of neighbor relevance and the use of graph memory, which are two critical components that are vital for a successful defense. In doing so, GNNGUARD can prune likely fake edges and assign less weight to suspicious edges, a principle grounded in network theory of homophily. Experiments on four datasets and across five GNNs show that GNNGUARD outperforms existing defense algorithms by a large margin. Lastly, we show how GNNGUARD can leverage structural equivalence and be used with heterophily graphs.

Broader Impact

Impacts on graph ML research. Graphs are universal structures of real-world complex systems. Because of strong representation learning capacity, GNNs have brought success in areas, ranging from disease diagnosis [10] and drug discovery [35] to recommendation system [55]. However, recent studies found that many GNNs are highly vulnerable to adversarial attacks [56]. Adversarial attackers inject imperceptible changes into graphs, thereby fooling downstream GNN classifiers into making incorrect predictions [9]. While there is a rich body of literature on adversarial attacks and defense on non-graph data (*e.g.*, text [39], and images [56]), much less is known about graphs. In an effort towards closing this gap, this paper introduces GNNGUARD, a powerful GNN defender that can be straightforwardly integrated into any existing GNN. Because GNNGUARD works with any GNN model, its impact on graph ML research is potentially more substantial than that of introducing another, albeit presumably more robust, GNN model.

A variety of impactful application areas. GNNGUARD can be used in a wide range of applications by simply integrating GNNGUARD with a GNN model of user choice that is most suitable in a particular application, as we demonstrate in this paper. Further positive impacts of GNNGUARD include the following. First, we envision that GNNGUARD will help users (*e.g.*, governments, companies, and individuals) avoid potential losses that are caused by misjudgments made by attacked GNNs (*e.g.*, in the face of a massive attack on a financial network) [9]. Second, it would be interesting to explore the possibility of deploying GNNGUARD for key GNN applications in biomedical domain, where, for example, a GNN diagnostics system could predict false diagnosis if it was trained on the attacked knowledge graph [57]. Finally, our model has implications for fairness and explainability of GNNs [36]), which is key to increase users' trust in GNN predictions. Lastly, GNNGUARD can be used for debugging GNN models and understanding of black-box GNN optimization.

The need for thoughtful use of GNNGUARD. It is possible to think of a situation where one would use GNNGUARD to get insights into black-box GNN optimization and then use those insights to improve existing attack algorithms, thereby identifying and potentially exploiting new, currently unknown vulnerabilities of GNNs. Because of this possibility and the fact that GNNs are becoming increasingly popular in real-world ML systems, it is important to conduct research to get insights into possible attacks and defense of GNNs.

Acknowledgments and Disclosure of Funding

References

[1] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: methods, applications and evaluations. Bioinformatics, 36(4), 2020.

[2] Haitham Ashoor, Xiaowen Chen, Wojciech Rosikiewicz, Jiahui Wang, Albert Cheng, Ping Wang, Yijun Ruan, and Sheng Li. Graph embedding and unsupervised learning predict genomic sub-compartments from hic chromatin interaction data. Nature Communications, 11(1), 2020.

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

[4] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. TKDE, 2020.

[5] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. TNNLS, 2020.

[6] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In ICML, 2017.

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

[8] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In KDD, 2018.

[9] Gean T Pereira and André CPLF de Carvalho. Bringing robustness against adversarial attacks. Nature Machine Intelligence, 1(11), 2019.

[10] Xintian Han, Yuxuan Hu, Luca Foschini, Larry Chinitz, Lior Jankelson, and Rajesh Ranganath. Deep learning models for electrocardiograms are susceptible to adversarial attack. Nature Medicine, 2020.

[11] SE Kreps and DL Kriner. Model uncertainty, political contestation, and public trust in science: Evidence from the COVID-19 pandemic. Science Advances, page eabd4563, 2020.

[12] Walt Woods, Jack Chen, and Christof Teuscher. Adversarial explanations for understanding image classification decisions and improved neural network robustness. Nature Machine Intelligence, 1(11), 2019.

[13] Samuel G Finlayson, John D Bowers, Joichi Ito, Jonathan L Zittrain, Andrew L Beam, and Isaac S Kohane. Adversarial attacks on medical machine learning. Science, 363(6433), 2019.

[14] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. Adversarial attacks and defenses in deep learning. Engineering, 2020.

[15] M Hutson. Ai can now defend itself against malicious messages hidden in speech. Nature, 2019.

[16] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. Annual Review of Sociology, 2001.

[17] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In IJCAI, 2019.

[18] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In WSDM, 2020.

[19] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In KDD, 2019.

[20] Xianfeng Tang, Yandong Li, Yiwei Sun, Huaxiu Yao, Prasenjit Mitra, and Suhang Wang. Transferring robustness for graph neural network against poisoning attacks. In WSDM, 2020.

[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In ICLR, 2018.

[22] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In ICML, 2018.

[23] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. ICLR, 2020.

[24] Dingxiong Deng, Fan Bai, Yiqi Tang, Shuigeng Zhou, Cyrus Shahabi, et al. Label propagation on k-partite graphs with heterophily. TKDE, 2019.

[25] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In KDD, 2018.

[26] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. ICML, 2020.

[27] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. KDD, 2020.

[28] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In CVPR, 2018.

[29] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In NIPS, 2018.

[30] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In ICLR, 2019.

[31] Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, and Jiliang Tang. Adversarial attacks and defenses on graphs: A review and empirical study. arXiv:2003.00653, 2020.

[32] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. ICML, 2018.

[33] Xuanqing Liu, Si Si, Xiaojin Zhu, Yang Li, and Cho-Jui Hsieh. A unified framework for data poisoning attack to graph-based semi-supervised learning. NeurIPS, 2019.

[34] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In ICML, 2019.

[35] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics, 34(13), 2018.

[36] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In NeurIPS, 2019.

[37] Pablo Gainza, Freyr Sverrisson, Frederico Monti, Emanuele Rodola, D Boscaini, MM Bronstein, and BE Correia. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. Nature Methods, 17(2), 2020.

[38] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. ICLR, 2015.

[39] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. EMNLP, 2017.

[40] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In NIPS, 2017.

[41] Xiao Zang, Yi Xie, Jie Chen, and Bo Yuan. Graph universal adversarial attacks: A few bad actors ruin graph learning models. arXiv:2002.04784, 2020.

[42] Kaidi Xu, Hongge Chen, Sijia Liu, Pin Yu Chen, Tsui Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. In IJCAI, 2019.

[43] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. Attention models in graphs: A survey. ACM TKDD, 13(6), 2019.

[44] Octavian-Eugen Ganea. Non-Euclidean Neural Representation Learning of Words, Entities and Hierarchies. PhD thesis, ETH Zurich, 2019.

[45] Tyler Derr, Yao Ma, and Jiliang Tang. Signed graph convolutional networks. In ICDM, 2018.

[46] Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In KDD, 2019.

[47] Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. In NeurIPS, 2019.

[48] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. Information Retrieval, 3(2), 2000.

[49] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. AI Magazine, 29(3), 2008.

[50] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. NeurIPS, 2020.

[51] Monica Agrawal, Marinka Zitnik, and Jure Leskovec. Large-scale analysis of disease pathways in the human interactome. In Pacific Symposium on Biocomputing, volume 23, 2018.

[52] Benoit Hudzia, M-Tahar Kechadi, and Adrian Ottewill. Treep: A tree based p2p network architecture. In ICCC, 2005.

[53] Tijana Milenković and Nataša Pržulj. Uncovering biological network function via graphlet degree signatures. Cancer informatics, 6, 2008.

[54] Tomaz Hocevar and Janez Demsar. A combinatorial approach to graphlet counting. Bioinformatics, 30(4):559–565, 2014.

[55] Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction. In CIKM, 2019.

[56] Zhenglong Zhou and Chaz Firestone. Humans can decipher adversarial images. Nature Communications, 10(1), 2019.

[57] Maya Rotmensch, Yoni Halpern, Abdulhakim Tlimat, Steven Horng, and David Sontag. Learning a health knowledge graph from electronic medical records. Scientific Reports, 7(1), 2017.

[58] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. Deeprobust: A pytorch library for adversarial attacks and defenses. arXiv:2005.06149, 2020.

Appendices to "GNNGUARD: Defending Graph Neural Networks against Adversarial Attacks"

Xiang Zhang
Harvard University
xiang_zhang@hms.harvard.edu

Marinka Zitnik
Harvard University
marinka@hms.harvard.edu

Appendix A Defense Performance Against Influence Targeted Attacks

Results are shown in Table 6. We find that the proposed GNNGUARD achieves the best defensive performance against influence targeted attack across five GNN models and four datasets. In particular, GNNGUARD outperforms state-of-the-art defense models by 8.77% on average. Furthermore, compared to the case where the GNN is attacked without any defense, GNNGUARD brings a significant accuracy improvement of 22.6% on average. Remarkably, results show that even most recently published GNNs (*e.g.*, GraphSAINT [23]) are sensitive to adversarial perturbations of the graph structure (cf. "Attack" vs. "No Attack" columns in Table 6), yet GNNGUARD can successfully defend GNNs against influence targeted attacks and can restore their performance to levels comparable to learning on clean, non-attacked graphs.

Table 6: Defense performance (multi-class classification accuracy) against influence targeted attacks.

Model	Dataset	No Attack	Attack	GNN-Jaccard	RobustGCN	GNN-SVD	GNNGUARD
GCN	Cora	0.826	0.410	0.520	0.605	0.425	**0.665**
	Citeseer	0.721	0.435	0.675	0.575	0.615	**0.745**
	ogbn-arxiv	0.667	0.545	0.615	0.620	0.445	**0.725**
	DP	0.682	0.475	0.550	0.565	0.460	**0.655**
GAT	Cora	0.827	0.425	0.550	0.605	0.450	**0.635**
	Citeseer	0.718	0.510	0.675	0.575	0.615	**0.815**
	ogbn-arxiv	0.669	0.635	0.525	0.620	0.505	**0.675**
	DP	0.714	0.470	0.540	0.565	0.570	**0.645**
GIN	Cora	0.831	0.525	0.635	0.605	0.615	**0.775**
	Citeseer	0.725	0.480	0.675	0.575	0.630	**0.845**
	ogbn-arxiv	0.661	0.570	0.605	0.620	0.525	**0.710**
	DP	0.719	0.505	0.585	0.565	0.605	**0.695**
JK-Net	Cora	0.834	0.525	0.665	0.605	0.625	**0.755**
	Citeseer	0.724	0.485	0.675	0.575	0.610	**0.865**
	ogbn-arxiv	0.678	0.545	0.580	0.620	0.475	**0.720**
	DP	0.726	0.495	0.635	0.565	0.590	**0.685**
Graph SAINT	Cora	0.821	0.405	0.495	0.610	0.395	**0.645**
	Citeseer	0.716	0.460	0.665	0.590	0.605	**0.735**
	ogbn-arxiv	0.683	0.525	0.595	0.615	0.570	**0.705**
	DP	0.739	0.435	0.615	0.645	0.575	**0.675**

Appendix B Defense Performance Against Non-Targeted Attacks

Results are shown in Table 7. To evaluate how harmful non-targeted attacks can be for GNNs, we first give results without attack and under attack (without defense), *i.e.*, "Attack" vs. "No Attack" columns in Table 7. We also show defense performance of GNNGUARD relative to state-of-the-art GNN defense techniques. First, we find that non-targeted attacks can have a considerable negative impact on the performance of the GNNs. The accuracy of even the strongest GNN is reduced by 18.7% on average. In addition, results show that our GNNGUARD outperforms baselines in most experiments and improves upon baselines considerably. Experiments indicate the proposed GNN defender can successfully mitigate negative impacts brought forward by non-targeted attacks on graphs.

Table 7: Defense performance (multi-class classification accuracy) against non-targeted attacks.

Model	Dataset	No Attack	Attack	GNN-Jaccard	RobustGCN	GNN-SVD	GNNGUARD
GCN	Cora	0.826	0.578	0.684	0.571	0.678	**0.714**
	Citeseer	0.721	0.601	0.646	0.583	0.668	**0.681**
	ogbn-arxiv	0.667	0.410	0.409	0.436	0.413	**0.444**
	DP	0.682	0.487	0.513	0.528	0.493	**0.539**
GAT	Cora	0.827	0.566	0.691	0.571	0.681	**0.718**
	Citeseer	0.718	0.676	0.667	0.583	0.680	**0.699**
	ogbn-arxiv	0.669	0.420	0.428	0.436	0.433	**0.432**
	DP	0.714	0.519	0.548	0.528	0.534	**0.566**
GIN	Cora	0.831	0.588	0.702	0.571	0.692	**0.722**
	Citeseer	0.725	0.565	0.638	0.583	0.615	**0.711**
	ogbn-arxiv	0.661	0.424	0.459	0.436	0.459	**0.486**
	DP	0.719	0.537	0.559	0.528	0.513	**0.571**
JK-Net	Cora	0.834	0.615	**0.726**	0.571	0.683	0.713
	Citeseer	0.724	0.574	0.647	0.583	0.679	**0.698**
	ogbn-arxiv	0.678	0.433	0.419	0.436	0.443	**0.457**
	DP	0.726	0.486	0.537	0.528	0.541	**0.587**
Graph SAINT	Cora	0.821	0.657	0.617	0.659	0.647	**0.705**
	Citeseer	0.716	0.628	0.596	0.637	0.652	**0.659**
	ogbn-arxiv	0.683	0.394	0.428	0.563	0.533	**0.583**
	DP	0.739	0.473	0.572	0.499	0.524	**0.537**

Appendix C Classification Accuracy on Clean (*i.e.*, Non-attacked) Datasets with and without GNNGUARD

Next, we want to investigate whether the GNN defender can harm the performance of the underlying GNN if the defender is used on clean, non-attacked graphs. Note that this is a practically important question, as in practice, users might not know a priori whether malicious agents have altered their graph datasets. Because of that, it is essential that a successful GNN defender does not decrease the predictive performance of the GNN in cases when GNNGUARD is turned on, but there is no attack. From results in the main paper and Appendix A-B, we already know that GNNGUARD can defend GNNs when they are attacked. Here, we show that GNNGUARD does not hinder GNNs even when they are not attacked.

Results are shown in Table 8. We observe that GNNs, trained on clean datasets, yield approximately the same performance irrespective of whether a GNN integrates GNNGUARD defense or not. These results suggest that the use of GNNGUARD does not reduce GNN's expressive power or its representation capacity when there are no adversarial attacks.

Appendix D Further Details on Datasets

GNNGUARD implementation as well as all datasets and the relevant data loaders are available at https://github.com/mims-harvard/GNNGuard. We provide further dataset statistics in Table 9.

Table 8: Classification accuracy on clean (*i.e.*, non-attacked) datasets with and without GNNGUARD.

	Cora-CLEAN		Citeseer-CLEAN		ogbn-arxiv-CLEAN		DP-CLEAN	
	w/o	w	w/o	w	w/o	w	w/o	w
GCN	0.826	0.817	0.721	0.716	0.667	0.683	0.682	0.681
GAT	0.827	0.829	0.718	0.719	0.669	0.674	0.714	0.717
GIN	0.831	0.832	0.725	0.726	0.661	0.671	0.719	0.716
JK-Net	0.834	0.829	0.724	0.727	0.678	0.682	0.726	0.731
GraphSAINT	0.821	0.819	0.716	0.721	0.683	0.669	0.739	0.727

Table 9: Dataset statistics. N, E, M, and C denote the number of nodes, edges, node feature dimensionality, and the number of labels/classes, respectively.

Dataset	N	E	M	C	Node features
Cora	2,485	5,069	1,433	7	Binary
Citeseer	2,110	3,668	3,703	6	Binary
ogbn-arxiv	31,971	71,669	128	40	Continuous
DP	22,552	342,353	73	519	Continuous
Synthesized	1,000	3,200	-	6	-

The new, Disease Pathway (DP) [51] dataset describes a system of interacting human proteins whose malfunction collectively leads to a variety of diseases. Nodes in the network represent human proteins and edges indicate protein-protein interactions. The raw dataset is available at `http://snap.stanford.edu/pathways`. The task is to predict for every protein node what diseases (*i.e.*, labels/classes) that protein might cause. The dataset has 73-dimensional continuous node features representing graphlet-orbit counts (*i.e.*, the number of occurrences of higher-order network motifs), which we normalize via z-scores. This is a multi label node classification dataset. We select 10 most-common labels (diseases), reformulate the task as 10 independent balanced binary classification problems and report average performance across multiple independent runs. The first four datasets are homophily graphs while the last synthesized graph is heterophily graph with structural equivalence. We randomly split the dataset into training (10%), validation (10%), and test set (80%) following the experimental setup in [8].

Appendix E   Further Details on Hyperparameter Setting

To select hyperparameters and GNN model architectures, we closely follow original authors' guidelines and relevant papers on GNNs (GCN [3], GAT [21], GIN [7], JK-Net [22], and Graph-SAINT [23]), baseline defense algorithms (GNN-Jaccard [17], RobustGCN [19], and GNN-SVD [18]), and models for generating adversarial attacks (Nettack-Di [8], Nettack-In [8], and Mettack [30]).

We use PyTorch DeepRobust package (`https://github.com/DSE-MSU/DeepRobust`) [58] to implement adversarial attack models and baseline defense algorithms, and PyTorch Geometric package (`https://github.com/rusty1s/pytorch_geometric`) to implement and train GNN models. In all experiments, we set the number of epochs to 200 and use early stopping (we stop training if validation accuracy does not increase for 10 consecutive epochs). We repeat every experiment 5 times and report average performance across independent runs. We set $P_0 = 0.5$, $K = 2$, $D_2 = 16$, and dropout rate as 0.5, optimize cross-entropy loss using Adam optimizer and learning rate of 0.01. For other parameters, we follow the setup in [8].