Graph Agent Network: Empowering Nodes with Inference Capabilities for Adversarial Resilience

Ao Liu[1], Wenshan Li[2*], Tao Li[1], Beibei Li[1], Guangquan Xu[3,4], Pan Zhou[5], Wengang Ma[1], Hanyuan Huang[1]

[1]School of Cyber Science and Engineering, Sichuan University
[2]School of Cyber Science and Engineering, Chengdu University of Information Technology
[3]College of Information Science and Technology, Shihezi University
[4]Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University
[5]School of Cyber Science and Engineering, Huazhong University of Science and Technology
{aliu, litao, libeibei, mawengang941206}@scu.edu.cn, helenali@cuit.edu.cn, losin@tju.edu.cn, panzhou@hust.edu.cn, huanghanyuan@stu.scu.edu.cn

Abstract

End-to-end training with global optimization have popularized graph neural networks (GNNs) for node classification, yet inadvertently introduced vulnerabilities to adversarial edge-perturbing attacks. Adversaries can exploit the inherent opened interfaces of GNNs' input and output, perturbing critical edges and thus manipulating the classification results. Current defenses, due to their persistent utilization of global-optimization-based end-to-end training schemes, inherently encapsulate the vulnerabilities of GNNs. This is specifically evidenced in their inability to defend against targeted secondary attacks. In this paper, we propose the Graph Agent Network (GAgN) to address the aforementioned vulnerabilities of GNNs. GAgN is a graph-structured agent network in which each node is designed as an 1-hop-view agent. Through the decentralized interactions between agents, they can learn to infer global perceptions to perform tasks including inferring embeddings, degrees and neighbor relationships for given nodes. This empowers nodes to filtering adversarial edges while carrying out classification tasks. Furthermore, agents' limited view prevents malicious messages from propagating globally in GAgN, thereby resisting global-optimization-based secondary attacks. We prove that single-hidden-layer multilayer perceptrons (MLPs) are theoretically sufficient to achieve these functionalities. Experimental results show that GAgN effectively implements all its intended capabilities and, compared to state-of-the-art defenses, achieves optimal classification accuracy on the perturbed datasets.

Extended version — https://arxiv.org/abs/2306.06909

1 Introduction

Graph neural networks (GNNs) have become state-of-the-art models for node classification tasks by leveraging end-to-end global training paradigms to effectively learn and extract information from graph-structured data (Hamilton et al. 2017). However, this approach has also inadvertently introduced inherent vulnerabilities, making GNNs vulnerable to adversarial edge-perturbing attacks. These vulnerabilities arise from GNNs exposing a global end-to-end training interface, which while allowing for precise classification, also provides adversaries with opportunities to attack GNNs.

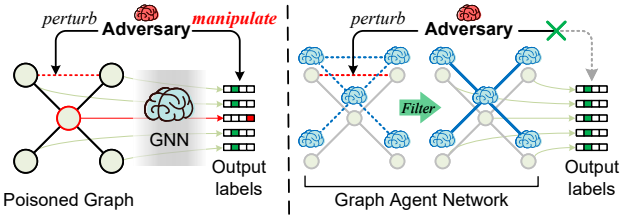

Figure 1: The principle of adversary exploiting the vulnerabilities caused by global optimization to attack GNNs, and the defense mechanism of GAgN against these attacks.

In response to the edge-perturbing attacks, existing defense mechanisms primarily rely on global-optimization-based defense methods (Sun et al. 2022). Their objective is to enhance the robustness of GNNs through adversarial training (Feng et al. 2019), aiming to increase the tolerant ability of perturbations to defend against potential adversarial perturbations. These approaches may still inherit some inherent vulnerabilities of GNNs, as their model frameworks still follow the global-optimization pattern of GNNs. Representative examples include: (1) RGCN (Zhu et al. 2019) replaces the hidden representations of nodes in each graph convolutional network (GCN) (Kipf and Welling 2017) layer to the Gaussian distributions, to further absorb the effects of adversarial changes. (2) GCN-SVD (Entezari et al. 2020) combines a singular value decomposition (SVD) filter prior to GCN to eliminate adversarial edges in the training set. (3) STABLE (Li et al. 2022) reforms the forward propagation of GCN by adding functions that randomly recover the roughly removed edges. Unfortunately, the computational universality of GNNs has been recently demonstrated (Loukas 2019), signifying that attributed graphs can be classified into any given label space, even those subject to malicious manipulation. This implies that defense approaches focused on en-

*Corresponding author

hancing GNNs' global robustness remain theoretically vulnerable, with potential weaknesses to secondary attacks. In fact, the vulnerability of existing global-optimization-based defenses has been theoretically proven (Liu et al. 2022). Adversaries can reduce the classification accuracy of these defenses once again by launching secondary attacks using these defenses as surrogate models. Experimental evidence demonstrates that even under the protection of state-of-the-art defensive measures (Geisler et al. 2020), secondary attacks targeting these defenses successfully mislead 72.8% of the nodes into making incorrect classifications once again.

To overcome the issues mentioned above, inspired by the natural filtering ability of decentralized intelligence (Saldanha et al. 2022), we propose a decentralized agent network called Graph Agent Networks (GAgN) whose principle is shown in Figure 1. GAgN empowers nodes with autonomous awareness, while limiting their perspectives to their 1-hop neighbors. As a result, nodes no longer rely solely on global-level end-to-end training data. Instead, they progressively gain the perception of the entire network through communication with their neighbors and accomplish self-classification.

Specifically, GAgN is designed to enhance the agent-based model (ABM) (Khodabandelu and Park 2021), ensuring improved compatibility with graph-based scenarios. In GAgN, agents are interconnected within the graph topology and engage in cautious communication to mitigate the impact of potential adversarial edges. As time progresses, these agents systematically exchange information to broaden their receptive fields, perceive the global information, and attain decentralized intelligence. An agent primarily includes two major abilities: 1) Storage: states, which are storable features, and actions, which are trainable functions for inference. 2) Communication: agents can receive states and actions from neighboring agents and integrate them to improve their own inference capabilities.

During the communication round, an agent organizes multiple restricted-view mini-datasets based on information received from its immediate neighbors, and updates its states and actions accordingly. After sufficient communications, the agent accumulates enough historical experience to perform specific tasks. These tasks include: 1) computing its own embedding, 2) estimating the possible degree of a given node, and 3) determining the neighboring confidence of two given nodes. The first function enables GAgN to perform node classification, whereas the second and third functions collaboratively contribute to filtering adversarial edges.

Furthermore, we rigorously demonstrate that all these functions can be accomplished by the single-hidden-layer multilayer perceptron (MLPs). This conclusion offers theoretical backing for substantially reducing GAgN's computational complexity. In practice, by instantiating nodes as lightweight agents, a GAgN can be constructed to carry out node classification tasks with adversarial resilience.

Our main contributions including:

- We propose a decentralized agent network, GAgN, which empowers nodes with the ability to autonomously perceive and utilize global intelligence to address the inherent vulnerabilities of GNNs and existing defense models.
- We theoretically prove that an agent can accomplish the relevant tasks using only three trainable matrices.
- We experimentally show that primary functions of GAgN have been effectively executed, which collectively yielding state-of-the-art accuracy in perturbed datasets.

2 The Proposed Method

We start with action spaces and discuss the state's structure and update strategies alongside other components. Figure 2 shows the workflow of GAgN.

Preliminaries Consider connected graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}|$ nodes. The degree of node v_i is $\deg_i$. The feature vector and one-hot label of v_i are $\mathbf{z}_i \in \mathbb{R}^{1\times d_z}$ and $\ell_i \in \mathbb{R}^{1\times d_L}$. The maximum degree is $\deg_{\max}$. The neighborhood N_i of v_i includes all v_j with $e_{i,j} \in \mathcal{E}$. The edge between v_i and v_j is $e_{i,j}$. Given a set of vectors, $\mathrm{Concat}_r(\cdot)$ and $\mathrm{Concat}_c(\cdot)$ output the row-wise and column-wise concatenated matrices.

2.1 Action Spaces

The action space consists of all potential actions a node can take in a reasoning task (Peng and Van De Panne 2017). In GAgN, it is designed as a set of perception functions with the same parameter structure but different specific parameters. The optimal parameters for agents in each communication round can be found within this action space. We use node v_i as an example to illustrate these functions.

Individual Function Parameters of individual functions are specific to each agent, reflecting the understanding gained through interactions with neighbors, without direct agent interactions. This includes:

▶ Neighbor feature aggregator $\mathcal{A}_i : \mathbb{R}^{(\deg_i+1)\times d_z} \to \mathbb{R}^{d_z}$ aggregates features from v_i's neighborhood into v_i. Similar to GAT, the node's own features are also aggregated. The trainable parameter of $\mathcal{A}_i$ is an attention vector $\mathbf{w}_i \in \mathbb{R}^{\deg_i+1}$, which captures the attention between v_i and its neighbors. $\mathcal{A}_i$ aggregates features of N_i as

$$\mathcal{A}_i(\mathrm{N}_i) = \mathbf{w}_i \mathrm{Concat}_r(\{\mathbf{z}_i\} \cup \{\mathbf{z}_j : v_j \in \mathrm{N}_i\})/\deg_i + 1. \quad (1)$$

$\mathbf{w}_i$ is initialized with minimal values and updated iteratively.

Communicable Functions Parameters of communicable functions are exchangeable, allowing agents to broaden their collective perception through sharing. Agents can learn from others' perceptions and integrate this knowledge into their own understanding. These functions encompass:

▶ Embedding function $\mathcal{M}_i : \mathbb{R}^{1\times d_z} \to \mathbb{R}^{1\times d_L}$ that embeds node features into the label space. We have proved that an effective embedding can be generated using a $d_z \times d_L$-dimensional trainable matrix (*c.f.* Corollary 1).

▶ Degree inference function $\mathcal{D}_i : \mathbb{R}^{1\times d_z} \to \mathbb{R}^{1\times \deg_{\max}}$ that predicts the probability distribution of a node's degree across the range of 1 to $\deg_{\max}$ (i.e., one-hot encoding) using a given feature. Instantiating $\mathcal{D}_i$ as a a $d_z \times \deg_{\max}$-dimensional trainable matrix can provide sufficient capacity for fitting (*c.f.* Theorem 2).

▶ Neighboring confidence function $\mathcal{N}_i : \mathbb{R}^{1\times 2d_z} \to \mathbb{R}$, a binary-classifier that infers whether two given node features

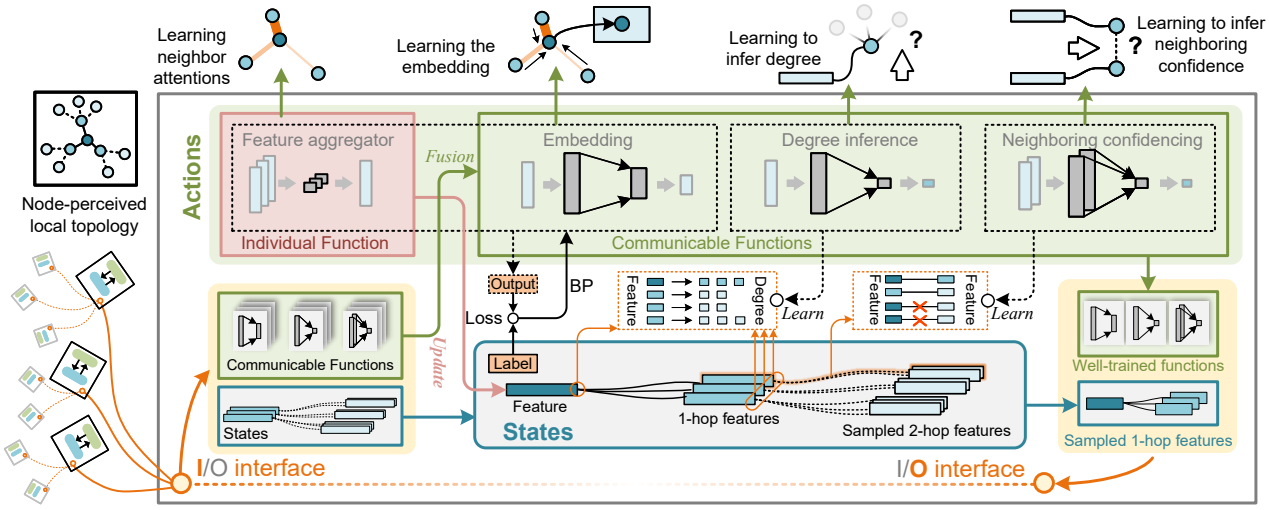

Figure 2: The internal structure and communication paradigm of an agent within GAgN.

are ground-truth neighbors, and the conclusion is provided in the form of a confidence value. We have proved that any two nodes on $\mathcal{G}$ can be inferred neighborhood relationship by a $2d_z \times 1$-dimensional trainable matrix (*c.f.* Theorem 3).

2.2 I/O Interface

Input In a single communication round, node v_i receives the following messages from its neighbors N_i. 1) Communicable functions of all nodes in N_i, i.e., sets of functions $\mathcal{M}_i^{\mathrm{rec}} = \{\mathcal{M}_j : v_j \in \mathrm{N}_i\}$, $\mathcal{D}_i^{\mathrm{rec}} = \{\mathcal{D}_j : v_j \in \mathrm{N}_i\}$, and $\mathcal{N}_i^{\mathrm{rec}} = \{\mathcal{N}_j : v_j \in \mathrm{N}_i\}$. 2) Features from the 1-hop neighbors $Z_i^{\mathrm{I}} = \{\mathbf{z}_j : v_j \in \mathrm{N}_i\}$. 3) Sampled features from the 2-hop neighbors $Z_i^{\mathrm{II}} = \bigcup_{v_j \in \mathrm{N}_i} \mathrm{S}(v_j) - Z_i^{\mathrm{I}}$, where $\mathrm{S}(\cdot)$ denotes the sample function.

Output Node v_i sends the following messages to its neighbors N_i. 1) Feature in the current communication round $\mathbf{z}_i$. 2) Sampled features from node v_i's neighbor. In this approach, v_i samples (i.e., operates $\mathrm{S}(\cdot)$) from its received immediate neighbor features Z^{I} and sends them out. The sampling function of v_i aims to sample the neighbors' features that have a relatively large difference from its own feature. That is, for a specific receiving nodes v_k, this approach reduces the similarity between the Z_k^{I} and Z_k^{II}, while reducing the amount of messages transmitted outward. Therefore, given the sampling size ρ, $\mathrm{S}(\cdot)$ is defined as

$$\mathrm{S}(v_i) = \begin{cases} \mathrm{N}_i, \rho \leq |\mathrm{N}_i| \\ \{\mathbf{z}_j : v_j \in N_i, \mathrm{rank}(||\mathbf{z}_j, \mathbf{z}_i||_2) \leq \rho\}, \rho > |\mathrm{N}_i| \end{cases}, \tag{2}$$

where $\mathrm{rank}(\cdot)$ denotes the rank, a smaller rank indicating a larger value. 3) Communicable functions $\mathcal{M}_i$, $\mathcal{D}_i$, and $\mathcal{N}_i$.

2.3 Decision Rules

The decision rules define how to choose actions in different states (Farmer and Foley 2009). In GAgN, they guide the training of function parameters in the action space, based on v_i's states after receiving neighbors' messages. These functions adjust parameters via the loss function using stochastic gradient descent (SGD).

Aggregator The loss function $J_{\mathcal{A}}(\cdot)$ of $\mathcal{A}_i$ assigns different weights to v_i's neighbors to achieve a more accurate embedding closer to its label. For the aggregated feature $\mathbf{z}_i^{\mathcal{A}} = \mathcal{A}_i(\mathrm{N}_i)$, the cross-entropy loss is: $J_{\mathcal{A}}(z_i^{\mathcal{A}}) = -\log \mathcal{M}_i(\mathbf{z}_i^{\mathcal{A}})\ell_i^{\top}$. Parameters of $\mathcal{A}_i$ are updated via SGD-based backpropagation. Notably, while $\mathcal{M}_i$ participates in forward propagation, updates for $\mathcal{A}_i$ and $\mathcal{M}_i$ occur asynchronously to prevent the leakage of agents' private understandings (i.e., $\mathcal{A}_i$'s parameters) to their neighbors before secure communication through end-to-end training. Consequently, the gradient for updating $\mathcal{A}_i$ is $\nabla_{\mathcal{A}_i}^{\mathrm{update}} = \nabla_{\mathcal{A}_i}\left(J_{\mathcal{A}}(\mathbf{z}_i^{\mathcal{A}})\right)$. To obtain $\nabla_{\mathcal{A}_i}^{\mathrm{update}}$, we perform iterative training while freezing the parameters of $\mathcal{M}_i$.

Embedding Func. During training, parameters of $\mathcal{A}_i$ are frozen before updating the parameters of $\mathcal{M}_i$, i.e., $\nabla_{\mathcal{M}_i}^{\mathrm{update}} = \nabla_{\mathcal{M}_i}\left(J_{\mathcal{A}}(\mathbf{z}_i^{\mathcal{A}})\right)$. $\mathcal{A}_i$ and $\mathcal{M}_i$ share the same forward propagation process. Upon explicit request for specific tasks, i.e., when the training of the corresponding model parameters is initiated, $\mathcal{A}_i$ and $\mathcal{M}_i$ independently perform back propagation based on $J_{\mathcal{A}}(z_i^{\mathcal{A}})$, thus maintaining the non-communicability of $\mathcal{A}_i$. This is illustrated in Figure 3.

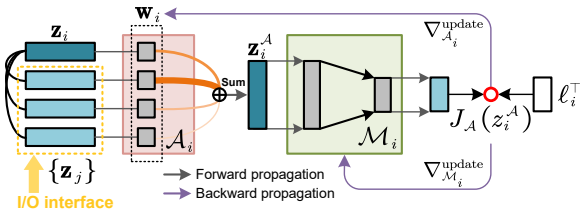

Figure 3: Independent training flow of $\mathcal{A}_i$ and $\mathcal{M}_i$.

Inference Func. Agents use the I/O interface to access neighboring features and degrees. $\mathcal{D}_i$ learns to associate these features with degrees. Neighbors' true degrees, provided as one-hot vectors, guide $\mathcal{D}_i$'s updates, mapping features to discrete degree distributions. The attention vector W_i adjusts the loss weight for each neighbor, reducing the impact of those with lower attention weights.

We denote the matrix representation of $\mathcal{D}_i$'s inference as $\mathcal{X}_i^{\mathcal{D}} = \sigma(\mathrm{Concat}_r(\mathcal{D}_i(\{\mathbf{z}_j : v_j \in \mathrm{N}_i\} \cup \mathbf{z}_i)))$, where σ is the nonlinear activation function, and the concatenated one-hot vector that encodes v_i's and its neighbors' degrees as the supervisor: $\mathcal{Y}_i^{\mathcal{D}} = \mathrm{OneHot}(\{d_j : v_j \in \mathrm{N}_i\} \cup \deg_i)$. Then, the loss function of $\mathcal{D}_i$ is defined as a Kullback-Leibler divergence (denoted as $\mathrm{KL}(\cdot)$) weighted by neighbor attention, i.e., $J_{\mathcal{D}}(\mathcal{X}_i^{\mathcal{D}}) = -\frac{\mathrm{KL}(\mathcal{X}_i^{\mathcal{D}} \| \mathcal{Y}_i^{\mathcal{D}}) \cdot \sum_j \mathbf{I}_j \mathbf{w}_{i,j}}{\deg_i + 1}$, where $\mathbf{I}$ is an all-ones vector with $\deg_{\max}$ elements, used to compute the sum of the elements in each row of a matrix.

Neighboring Confidence Func. v_i uses $\mathcal{N}_i$ to identify neighboring relationships based on node features. The training data for $\mathcal{N}_i$ includes features within v_i's view: $\mathbf{z}_i$ and $\{\mathbf{z}_j : v_j \in \mathrm{N}_i\}$. In a graph, distinguishable decision regions exist for similarity patterns between v_i and its first- and second-order neighbors (Perozzi et al. 2014). Thus, $\mathcal{N}_i$ uses samples from the current communication round to train on two categories: 1) feature pairs labeled as "neighbor"

$$\mathcal{X}_1^{\mathcal{N}} = \{\mathrm{Concat}_c(\mathbf{z}_i, \mathbf{z}_j), \mathrm{Concat}_c(\mathbf{z}_j, \mathbf{z}_i) : v_j \in Z_i^{\mathrm{I}}\}, \quad (3)$$

and 2) the feature pairs that are labeled as "non-neighbor" with training samples

$$\mathcal{X}_0^{\mathcal{N}} = \{\mathrm{Concat}_c(\mathbf{z}_i, \mathbf{z}_k), \mathrm{Concat}_c(\mathbf{z}_k, \mathbf{z}_i) : v_k \in Z_i^{\mathrm{II}}\}. \quad (4)$$

Then, according to the training samples $\mathcal{X}^{\mathcal{N}} = \mathcal{X}_1^{\mathcal{N}} \cup \mathcal{X}_0^{\mathcal{N}}$, the loss function of $\mathcal{D}_i$ is defined as the binary cross-entropy loss (denoted as $\mathrm{CE}(\cdot)$), i.e., $J_{\mathcal{N}}(\mathcal{X}^{\mathcal{N}}) = \mathrm{CE}(\mathcal{X}_1^{\mathcal{N}}, \mathcal{X}_0^{\mathcal{N}})$

All loss functions are differentiable, allowing functions to self-train based on their losses. Using SGD for parameter adjustment, these functions reduce the loss and improve inference capabilities.

2.4 Action Update Rules

Action update rules are essential for updating agents' actions (i.e., functions) based on functions received from their neighbors. Using these rules, agents can dynamically adjust their actions to adapt to the evolving environment and enhance collective performance.

Node v_i acquires the integrated feature of its neighbors' actions through weighted fusion of the transmitted actions in the current round. Based on these synthesized features, v_i updates its actions accordingly. The weighted fusion methods are detailed in the following subsections.

Embedding Func. These functions affects classification results, which adversaries may target, and any node in $\mathcal{V}$ could be compromised (some neighbors of v_i might be malicious). Thus, v_i cannot fully trust its neighbors' $\mathcal{M}_.$. Since attention weights are individual functions and resist malicious message accumulation, the weighted fusion of $\mathcal{M}_i^{\mathrm{rec}}$ using $\mathbf{w}_i$ can filter harmful gradients, enabling a safe update of $\mathcal{M}_i$. Specifically, the parameter of $\mathcal{M}_i$ is updated as

$$\boldsymbol{\theta}_{\mathcal{M}_i} \leftarrow \boldsymbol{\theta}_{\mathcal{M}_i} + \eta_{\mathcal{M}} \omega_{i,j} \sum_{\mathcal{M}_j \in \mathcal{M}_i^{\mathrm{rec}}} \left(\boldsymbol{\theta}_{\mathcal{M}_j} - \boldsymbol{\theta}_{\mathcal{M}_i}\right) / |\mathrm{N}_i|, \quad (5)$$

where $\omega_{i,j}$ is the j-th attention in $\mathbf{w_i}$ and $\eta_{\mathcal{M}}$ is the learning rate for updates that is usually the same as that for $\mathcal{M}_i$ itself.

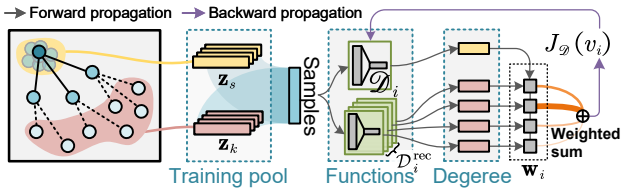

Figure 4: The training strategy for middleware function $\mathscr{D}_i$.

Degree Inference Func. For identical features, different degree inference functions may yield varying results. To harmonize these, v_i trains a middleware function $\mathscr{D}_i$ to ensure that inference results for $\mathbf{z}_k, \forall v_k \in \mathcal{V}$ by $\mathscr{D}_i$ closely match those from the received set $\mathcal{D}_i^{\mathrm{rec}}$. This involves initializing $\mathscr{D}_i$ randomly and designing a loss function based on spatial random sampling for training, aiming to integrate stored functions and update $\mathcal{D}_i$. The training involves: 1) Implementing minor random perturbations in Euclidean space to sample features $\mathbf{z}_s$ akin to $\mathbf{z}_i$, calculated as $\mathbf{z}_s = \mathbf{z}_i + \xi\mathbf{x}$, where $\mathbf{x}$ is an i.i.d. 0-1 Gaussian vector and ξ the sample range. 2) Using the features of nodes in Z_i^{II} as negative samples to broaden the sample diversity. 3) Applying $\mathscr{D}_i$ and $\mathcal{D}_i^{\mathrm{rec}}$ to separately infer these samples and utilizing the mean squared error (MSE) to evaluate the differences in inference outcomes. Training for $\mathscr{D}_i$ are depicted in Figure 4.

In summary, the loss function of $\mathscr{D}_i$ is

$$J_{\mathscr{D}}(v_i) = \sum_{\mathcal{D}_j \in \mathcal{D}_i^{\mathrm{rec}}} \omega_{i,j} \Big[-\sum_{Q} \mathbb{E}_{\mathbf{z}_s \sim P(\mathbf{z}_i)} \mathrm{MSE}\left(\mathscr{D}_i(\mathbf{z}_s), \mathcal{D}_j(\mathbf{z}_s)\right) - \sum\nolimits_{v_k \in Z_i^{\mathrm{II}}} \mathrm{MSE}\left(\mathscr{D}_i(\mathbf{z}_k), \mathcal{D}_j(\mathbf{z}_k)\right) \Big], \quad (6)$$

where P is a sampling distribution in the Euclidean space, Q defines the number of sample. After training that employing $J_{\mathscr{D}}(v_i)$ as the loss, the parameter of $\mathcal{D}_i$ is updated as $\boldsymbol{\theta}_{\mathcal{D}_i} \leftarrow \boldsymbol{\theta}_{\mathcal{D}_i} + \eta_{\mathcal{D}} \left(\boldsymbol{\theta}_{\mathcal{D}_i} - \boldsymbol{\theta}_{\mathscr{D}_i}\right)$.

Neighboring Confidence Func. From Eqs. (3) and (4), under a limited view, the training set always includes z_i when training $\mathcal{N}_i$ for any node v_i. This means $\mathcal{N}_i$ can only detect neighboring relationships for itself and a node with a given feature. The action update mechanism allows for generalization. By receiving neighboring functions $\mathcal{N}_i^{\mathrm{rec}}$ via the I/O interface, $\mathcal{N}_i$ can expand its scope to a richer dataset, learning from neighbors' inferential capabilities. 1) Construct a training set as per Eqs. (3) and (4). 2) Use both $\mathcal{N}_i$ and $\mathcal{N}_i^{\mathrm{rec}}$ to predict outputs of the same samples. 3) For a neighbor v_k, $\mathcal{N}_k$'s output measures the similarity in inferential ability with $\mathcal{N}_i$. Similarity implies alignment; otherwise, a higher loss prompts $\mathcal{N}_i$ to adjust towards $\mathcal{N}_k$. 4) Iterative training with all received $\mathcal{N}_i^{\mathrm{rec}}$ completes function fusion.

The evaluation of $\mathcal{N}_k$ for $\mathcal{N}_i$ is quantified by the absolute differences between their outputs. After all functions in $\mathcal{D}_i^{\text{rec}}$ provide evaluations, the loss is calculated by a weighted sum. This loss is then used to compute the gradient updates for a single update round of $\mathcal{N}_i$: $\boldsymbol{\theta}_{\mathcal{N}_i} \leftarrow \boldsymbol{\theta}_{\mathcal{N}_i} - \eta_{\mathcal{N}} \frac{\partial \left(\sum_{x_k \in \mathcal{X}^{\mathcal{N}}} \sum_{\mathcal{N}_j \in \mathcal{N}_i^{\text{rec}}} \omega_{i,j} \left| \mathcal{N}_i(x_k) - \mathcal{N}_j(x_k) \right| \right)}{\partial \boldsymbol{\theta}_{\mathcal{N}_i} \deg_i |\mathcal{X}_0^{\mathcal{N}} \cup \mathcal{X}_1^{\mathcal{N}}|}$. Through multiple rounds of training, the collective guidance of $\mathcal{N}_j, \forall v_j \in \mathrm{N}_i$ on $\mathcal{N}_i$'s gradient descent direction is achieved through continuous evaluations, progressively fusing their inferential abilities into $\mathcal{N}_i$.

2.5 Filtering Adversarial Edges

As communication between agents tends towards convergence, they initiate the detection of adversarial edges. Given that all agents possess the capacity for inference, an intuitive approach would be to allow inter-agent detection until a sufficient and effective number of adversarial edges are identified. However, this method necessitates the execution of $|\mathcal{V}| \times |\mathcal{V}|$ inference calculations, the majority of which are redundant and do not proportionately contribute to the global detection rate relative to the computational power consumed. Therefore, we propose a multilevel filtering method that significantly reduces computational load while preserving a high detection rate.

Specifically, the entire detection process is grounded in the functions trained from the agents' first-person perspectives, with each agent acting both as a detector and a detectee. The detection procedure is illustrated by taking node v_i as the detector and node v_x as the detectee:

1. For any node v_x, if there is a neighbor with insufficient attention, the edge formed by this neighbor relationship is considered as a suspicious edge.
2. Random proxy communication channels are established, introducing an unfamiliar neighbor v_i to v_x, with both parties exchanging information through I/O interfaces.
3. v_i infers the degree of v_x based on its own experience (i.e., applies $\mathcal{D}_i(\mathbf{z}_x)$); if the inferred result deviates from the actual outcome, the process advances to the subsequent detection step; otherwise, a new detectee is selected.
4. v_i estimates neighbor confidence between v_x and its neighbors using its experience (i.e., computing $\text{Concat}_c(\mathbf{z}_x, \mathbf{z}_u), \forall v_u \in \mathrm{N}_x$); edges with trust levels below a threshold are identified as adversarial edges.

Employing this approach, administrators possessing global model access rights are limited to establishing proxy connections, which entails creating new I/O interfaces for nodes without the ability to interfere in specific message passing interactions between them. As a result, distributed agents are responsible for conducting detection tasks, enabling individual intelligences to identify adversarial perturbations derived from global training.

2.6 Theoretical Analysis

We first establish the global equivalence.

Theorem 1 (Equivalence): Denote $\mathbb{M}(\mathcal{G}) = \{\mathbf{z}_i^{\mathbb{M}} = \mathcal{M}_i(\mathbf{z}_i) : v_i \in \mathcal{V}\}$ as the embedding result of all nodes, $\texttt{SAGE}(\mathcal{G}) = \{\mathbf{z}_i^{\text{S}}, ...\mathbf{z}_{|\mathcal{V}|}^{\text{S}}\}$ as the classification result of GraphSAGE, and G as the set of all attributed graphs. For any SAGE there exists $\mathbb{M}$ that

$$\mathbb{M}(\mathcal{G}) = \texttt{SAGE}(\mathcal{G}), \forall \mathcal{G} \in G. \tag{7}$$

We then propose the corollary to guide the design of the embedding function:

Corollary 1: If model $\mathcal{M}_i$ is a single-layer neural network with trainable parameters as a $d_z \times d_L$ matrix, then $\mathbb{M}_i$ is universally capable of embedding any feature space onto the corresponding label space.

For the structure of the degree inference function, we arrive at the following conclusions:

Theorem 2: Given an attribute graph $\forall \mathcal{G} \in G$, it is possible to map its feature matrix to any label matrix using only a single linear transformation (i.e., a trainable matrix) and a nonlinear activation.

For the neighboring confidence function's structure, we present the following findings:

Theorem 3: For any graph $\mathcal{G}$, there exists a fixed $2d_z$-dimensional weight vector $\mathbf{q} = [q_1, \ldots, q_{2d_z}]$, such that for any two nodes v_i and v_j in $\mathcal{G}$ and their true neighboring relationship ϕ (where ϕ is a relative minimum or converges to 1), $\text{Concat}_r(\mathbf{z}_i, \mathbf{z}_j)\mathbf{q}^\top = \phi$ is solvable.

3 Experiments

Datasets. Our approaches are evaluated on six real-world datasets widely used for studying graph adversarial attacks, including Cora, Citeseer, Polblogs, and Pubmed.

Baselines. The GAgN model protects non-defense GNNs from edge-perturbing attacks and outperforms other defensive approaches. Baseline models are evaluated as follows: *Comparison defending models.* GAgN is compared against: 1) RGCN, which uses Gaussian distributions for node representations to mitigate adversarial effects, 2) GNN-SVD, which applies truncated SVD for a low-rank adjacency matrix approximation, 3) Pro-GNN (Jin et al. 2020), which enhances GNN robustness through intrinsic node properties, 4) Jaccard (Wu et al. 2019), defending based on Jaccard similarity, 5) EGNN (Liu et al. 2021), filtering perturbations via ℓ_1- and ℓ_2-based smoothing. *Attack methods.* Experiments incorporate: 1) Metattack, a meta-learning based strategy, 2) G-EPA (Liu et al. 2022), a generalized edge-perturbing attack approach. The perturbation rate for attacks is set at 20%, a standard in the field unless otherwise specified.

Classification Accuracy We evaluate the global classification accuracy of the proposed GAgN against primary and secondary attacks. Our experimental design uses GCN and its defenses as surrogates for Metattack and G-EPA, where Metattack represents the most potent primary attack and G-EPA an effective secondary strategy. Accuracy results are detailed in Table 1. As GAgN closes global input-output interfaces, it cannot directly serve as a surrogate in secondary attacks. To compensate, we transfer perturbed graphs from

Surrogate		GCN (primary attack by Metattack)						Corresponding defenses (secondary attack by G-EPA)					
Dataset	r_p	RGCN	SVD	Pro	Jaccard	EGNN	GAgN	RGCN	SVD	Pro	Jaccard	EGNN	GAgN
Cora	20	58.67	57.01	63.94	72.51	69.02	**77.01**	49.17±0.89	47.91±2.18	55.04±1.11	63.11±1.84	59.52±1.39	**74.91±0.74**
Citeseer	20	62.53	57.29	56.24	66.21	64.94	**70.49**	54.03±2.00	48.79±1.77	47.74±1.59	57.71±1.57	56.44±1.94	**69.19±0.91**
Polblogs	20	58.36	54.87	73.10	69.87	75.42	**80.92**	49.86±1.17	46.37±2.71	64.60±1.35	61.37±2.07	66.92±1.54	**78.42±2.38**
Pubmed	20	71.20	81.24	82.82	76.39	79.06	**82.98**	62.70±1.34	72.74±0.94	74.07±0.75	67.89±0.73	70.56±0.79	**78.29±0.67**

Table 1: Classification accuracy (%) under primary and secondary attackm with the highest scores in **bold** and the second highest underlined. r_p is the perturbation rate. SVD and Pro is the abbreviation of GNN-SVD and Pro- GNN respectively.

baseline models under secondary attacks to the GAgN test set. Five datasets are conducted, presenting average accuracies and variation ranges to illustrate GAgN's resilience against secondary attacks. The diversity of the baselines likely covers potential vulnerabilities in defenses.

Our results indicate that GAgN achieves the best classification accuracy under both primary and secondary attacks. This is particularly noteworthy, as the lack of global input-output interfaces in GAgN makes it difficult for adversaries to exploit its vulnerabilities through global training, rendering it nearly impervious to secondary attacks.

Effectiveness of Aggregator

Symmetry of Attentions. In a well-trained GAgN, adjacent agents should learn similar attentions for their connecting edges (i.e., $\omega_{i,j} \approx \omega_{j,i}$) under the absence of collusion. To validate this, we first arbitrarily assign directions to the edges in the graph, defining the attention of any edge $e_{i,j}$ as $\omega_{i,j}$ from node v_i to v_j. This results in a new graph with reversed attentions for all edges. We use a line chart to compare forward and reverse attentions for the same edges after learning in the GAgN model. For a certain edge's smoothness, we sort the forward attention, obtain the sorted edge index, and use this index to find the corresponding reverse attention, examining variations in attentions. As shown in Figure 5, the variations between forward and reverse attentions are nearly identical, indicating that adjacent agents learn similar attentions. This confirms the effectiveness of the neighbor feature aggregator in the GAgN model. *To demonstrate consistency among adjacent agents in learning attention for the same edge, we selected the node with the highest degree (168) from Cora. Experimental results, presented in Appendix B.1, show the learning curves of neighboring agents' attentions on the same edge.*

Distribution of Attentions. Here we investigate the attention distribution learned by agents for both normal edges and adversarial edges. A significant distinction between the two would indicate that agents, through communication, have become aware that lower weights should be assigned to adversarial edges, thereby autonomously filtering out information from illegitimate neighbors introduced by such edges. As the previous set of experiments have demonstrated the symmetry of attention, we present the attention of a randomly selected agent on one end of the edge. The experimental results, as depicted in Figure 6, reveal a notable difference in the kernel density estimation (KDE) of attentions between normal edges and adversarial edges (generated by Metattack). This outcome substantiates that agents, by training the aggregator, have acquired the capability to autonomously filter out adversarial edges.

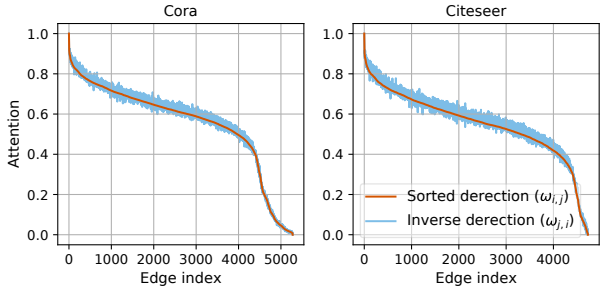

Figure 5: Attentions of random direction & inverse direction on the same edges.

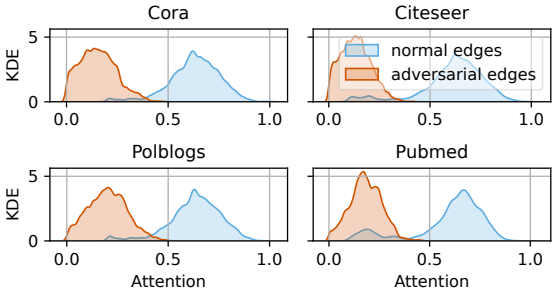

Figure 6: Distribution of attentions on normal and adversarial edges.

Effectiveness of Embedding After communication, the agent uses a well-trained embedding function to map its feature into the label space. We visualize the global embeddings to assess their effectiveness. If agents' embeddings, trained only on their labels under limited knowledge, still show global clustering, it indicates the embedding function's success. Figure 7 shows the t-SNE visualization of Cora and Citeseer embeddings (Van der Maaten and Hinton 2008), with points of different colors indicating distinct categories. The clear clustering of points validates the embedding function's efficacy. *Additionally, the visualization of the embedding process is presented in Appendix B.2, illustrating how GAgN progressively learns effective embeddings.*

Effectiveness of Degree Inference Each agent possesses the ability to infer the degree of any node based on the degree inference function. To validate the effectiveness of this

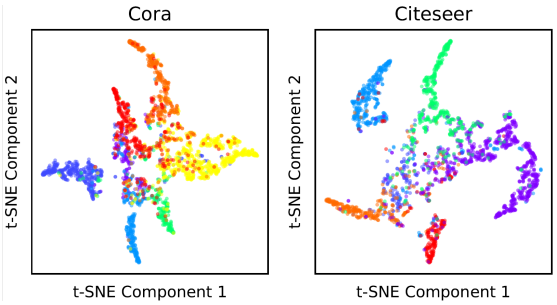

Figure 7: Well-trained agents generated embedding.

	Cora	Citeseer	Polblogs	Pubmed
L-N	90.35	89.21	92.41	81.08
S-N	95.31	94.19	96.32	83.45
D-N	92.64	91.08	92.57	83.06

Table 2: Degree inference accuracy (%)

inference, we select 150 representative nodes as test sets after training the GAgN model on the corresponding clean graph, following these rules: 1) the top 50 nodes with the highest degree, denoted as L-N; 2) the top 50 nodes with the lowest degree, denoted as S-N; 3) the 50 nodes furthest away from the inference node in terms of graph distance, denoted as D-N. Notably, to avoid training set leakage into the test set, an agent's 1-hop neighbors and select 2-hop neighbors are excluded from its test set. The experimental results, as shown in Table 3, demonstrate that even without data on the test nodes, agents can generalize their inference capabilities to the nodes of the graph under limited knowledge training, effectively inferring the degree of nodes.

Effectiveness of Neighboring Confidence To validate the neighboring confidence function on graph $\mathcal{G}$, $|\mathcal{V} \times \mathcal{V}|$ matrix operations are needed. To reduce computational complexity and ensure fair evaluation, we construct four representative graphs and use their neighbor relationships as test samples: 1) The original graph $\mathcal{G}_0$, with all labels set to 1. 2) A graph $\mathcal{G}_1$ with randomly distributed edges, labeled based on whether edges are original. 3) A randomly rewired graph $\mathcal{G}_2$, with all labels set to 0. 4) A graph $\mathcal{G}_3$ with only adversarial edges, all labeled 0. We randomly select 50 nodes to compute the confidence for neighbor relationships, treating this as a binary classification problem. Classification accuracy measures the function's effectiveness. Figure 8 shows that the neighboring confidence function performs well across all test sets, demonstrating its effectiveness.

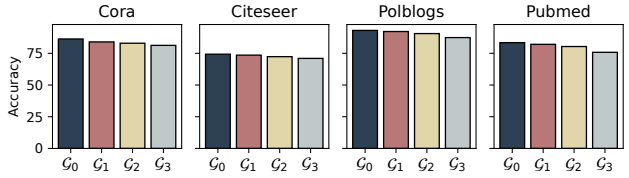

Figure 8: Accuracy of neighboring Confidence function.

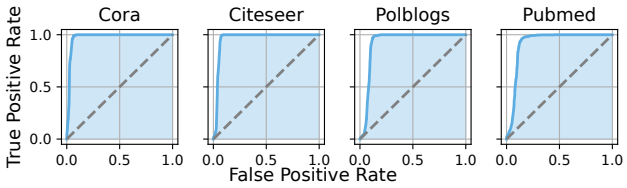

Figure 9: ROC curves with different attention threshold.

Dataset	Cora		Citeseer	
Metrics	TP (TPR)	FP (FPR)	TP (TPR)	FP (FPR)
Att.	928 (85.52%)	302 (5.56%)	782 (82.66%)	341 (7.20%)
Att.+$\mathcal{D}$+$\mathcal{N}$	901 (83.04%)	47 (0.87%)	753 (79.60%)	55 (1.16%)

Table 3: Results of ablation experiments

Ablation Experiments We employed a multi-module filtering approach to reduce computational complexity when filtering adversarial edges. To investigate the individual effects of these modules when operating independently, it is crucial to conduct ablation experiments. Section 3 has explored the distribution patterns of attention scores for normal and adversarial edges. Edges with low attention scores are initially considered suspicious, which implies that the attention threshold may influence the detection rate and false alarm rate. To quantitatively examine the impact of threshold selection on these rates, we utilize receiver operating characteristic (ROC) curves. As depicted in Figure 9, the detection rate is relatively insensitive to the attention threshold. This allows for a high detection rate while maintaining low false alarm rates.

After identifying suspicious edges based on attention, the degree inference and neighboring confidence functions help filter out normal edges, reducing false positives. We examine the effectiveness of these functions in detecting adversarial edges while retaining normal ones. True positives (TP) and true positive rate (TPR) measure detection accuracy, while false positives (FP) and false positive rate (FPR) measure normal edge misclassification. We use Metattack for testing. Table 3 shows results: "*Att.*" refers to detection by attention alone, and "*Att.*+$\mathcal{D}$+$\mathcal{N}$" refers to combined detection. Detection using only attention has a higher rate but also more false positives. Adding degree inference and neighboring confidence reduces false positives significantly without majorly affecting detection rates.

4 Conclusions

We proposed GAgN for addressing the inherent vulnerabilities of GNNs to edge-perturbing attacks. By adopting a decentralized interaction mechanism, GAgN facilitates the filtration of adversarial edges and thwarts global attacks. The theoretical sufficiency for GAgN further simplifies the model, while experimental results validate its effectiveness in resisting edge-perturbing attacks.

Acknowledgments

This work is supported in part by the National Key R&D Program of China (No. 2022YFB3102100, 2020YFB1805400), the National Science Foundation of China (No. U22B2027, 62032002, 62372313, 62172297, 62476107), the China Postdoctoral Science Foundation (No. 2024M752211), the Sichuan Science and Technology Program (No. 24QYCX0417), and the Youth Science Foundation of Sichuan (No. 25QNJJ4560).

References

Entezari, N.; Al-Sayouri, S. A.; Darvishzadeh, A.; and Papalexakis, E. E. 2020. All you need is low (rank) defending against adversarial attacks on graphs. In *WSDM*, 169–177.

Farmer, J. D.; and Foley, D. 2009. The economy needs agent-based modelling. *Nature*, 460(7256): 685–686.

Feng, F.; He, X.; Tang, J.; and Chua, T.-S. 2019. Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Trans. Knowl. Data. Eng.*, 33(6): 2493–2504.

Geisler, S.; Zügner, D.; Günnemann, S.; and x, x. 2020. Reliable Graph Neural Networks via Robust Aggregation. In *NeurIPS*, 13272–13284.

Hamilton, W.; Ying, Z.; Leskovec, J.; and x, x. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

Jin, W.; Ma, Y.; Liu, X.; Tang, X.; Wang, S.; and Tang, J. 2020. Graph structure learning for robust graph neural networks. In *ACM SIGKDD*, 66–74.

Khodabandelu, A.; and Park, J. 2021. Agent-based modeling and simulation in construction. *Autom. Constr.*, 131: 103882.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Li, K.; Liu, Y.; Ao, X.; Chi, J.; Feng, J.; Yang, H.; and He, Q. 2022. Reliable Representations Make A Stronger Defender: Unsupervised Structure Refinement for Robust GNN. In *ACM SIGKDD*.

Liu, A.; Li, B.; Li, T.; Zhou, P.; and Wang, R. 2022. AN-GCN: An Anonymous Graph Convolutional Network Against Edge-Perturbing Attacks. *IEEE Trans. Neural Netw. Learn. Syst.*

Liu, X.; Jin, W.; Ma, Y.; Li, Y.; Liu, H.; Wang, Y.; Yan, M.; and Tang, J. 2021. Elastic graph neural networks. In *ICML*, 6837–6849.

Loukas, A. 2019. What graph neural networks cannot learn: depth vs width. In *ICLR*.

Peng, X. B.; and Van De Panne, M. 2017. Learning locomotion skills using deeprl: Does the choice of action space matter? In *ACM SIGGRAPH*.

Perozzi, B.; Al-Rfou, R.; Skiena, S.; and x, x. 2014. Deepwalk: Online learning of social representations. In *ACM SIGKDD*, 701–710.

Saldanha, O. L.; Quirke, P.; West, N. P.; James, J. A.; Loughrey, M. B.; Grabsch, H. I.; Salto-Tellez, M.; Alwers, E.; Cifci, D.; Ghaffari Laleh, N.; et al. 2022. Swarm learning for decentralized artificial intelligence in cancer histopathology. *Nat. Med.*, 28(6): 1232–1239.

Sun, L.; Dou, Y.; Yang, C.; Zhang, K.; Wang, J.; Philip, S. Y.; He, L.; and Li, B. 2022. Adversarial attack and defense on graph data: A survey. *IEEE Trans. Knowl. Data Eng.*

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *JMLR*, 9(11): 2579–2605.

Wu, H.; Wang, C.; Tyshetskiy, Y.; Docherty, A.; Lu, K.; and Zhu, L. 2019. Adversarial examples for graph data: deep insights into attack and defense. In *IJCAI*.

Zhu, D.; Zhang, Z.; Cui, P.; and Zhu, W. 2019. Robust graph convolutional networks against adversarial attacks. In *ACM SIGKDD*, 1399–1407.