Enhancing Graph Neural Network-based Fraud Detectors against Camouflaged Fraudsters

Yingtong Dou[1], Zhiwei Liu[1], Li Sun[2], Yutong Deng[2], Hao Peng[3], Philip S. Yu[1]
[1]Department of Computer Science, University of Illinois at Chicago
[2]School of Computer Science, Beijing University of Posts and Telecommunications
[3]Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University
{ydou5,zliu213,psyu}@uic.edu,{l.sun,buptdyt}@bupt.edu.cn,penghao@act.buaa.edu.cn

ABSTRACT

Graph Neural Networks (GNNs) have been widely applied to fraud detection problems in recent years, revealing the suspiciousness of nodes by aggregating their neighborhood information via different relations. However, few prior works have noticed the camouflage behavior of fraudsters, which could hamper the performance of GNN-based fraud detectors during the aggregation process. In this paper, we introduce two types of camouflages based on recent empirical studies, i.e., the feature camouflage and the relation camouflage. Existing GNNs have not addressed these two camouflages, which results in their poor performance in fraud detection problems. Alternatively, we propose a new model named CAmouflage-REsistant GNN (CARE-GNN), to enhance the GNN aggregation process with three unique modules against camouflages. Concretely, we first devise a label-aware similarity measure to find informative neighboring nodes. Then, we leverage reinforcement learning (RL) to find the optimal amounts of neighbors to be selected. Finally, the selected neighbors across different relations are aggregated together. Comprehensive experiments on two real-world fraud datasets demonstrate the effectiveness of the RL algorithm. The proposed CARE-GNN also outperforms state-of-the-art GNNs and GNN-based fraud detectors. We integrate all GNN-based fraud detectors as an open-source toolbox[1]. The CARE-GNN code and datasets are available at https://github.com/YingtongDou/CARE-GNN.

CCS CONCEPTS

• **Security and privacy → Web application security**; • **Computing methodologies → Neural networks**.

KEYWORDS

Graph Neural Networks, Fraud Detection, Reinforcement Learning

ACM Reference Format:
Yingtong Dou[1], Zhiwei Liu[1], Li Sun[2], Yutong Deng[2], Hao Peng[3], Philip S. Yu[1]. 2020. Enhancing Graph Neural Network-based Fraud Detectors against Camouflaged Fraudsters. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411903

[1]https://github.com/safe-graph/DGFraud

1 INTRODUCTION

As Internet services thrive, they also incubate various kinds of fraudulent activities [14]. Fraudsters disguise as regular users to bypass the anti-fraud system and disperse disinformation [44] or reap end-users' privacy [32]. To detect those fraudulent activities, graph-based methods have become an effective approach in both academic [7, 21, 38] and industrial communities [2, 28, 50]. Graph-based methods connect entities with different relations and reveal the suspiciousness of these entities at the graph level, since fraudsters with the same goal tend to connect with each other [1].

Recently, as the development of Graph Neural Networks (GNNs) (e.g., GCN [17], GAT [34], and GraphSAGE [12]), many GNN-based fraud detectors have been proposed to detect opinion fraud [19, 25, 39], financial fraud [23, 24, 37], mobile fraud [41], and cyber criminal [48]. In contrast to traditional graph-based approaches, GNN-based methods aggregate neighborhood information to learn the representation of a center node with neural modules. They can be trained in an *end-to-end* and *semi-supervised* fashion, which saves much feature engineering and data annotation cost.

However, existing GNN-based fraud detection works only apply GNNs in a narrow scope while ignoring the camouflage behaviors of fraudsters, which have been drawing great attention from both researchers [8, 15, 16, 49] and practitioners [2, 19, 41]. Meanwhile, theoretical studies prove the limitations and vulnerabilities of GNNs when graphs have noisy nodes and edges [3, 4, 13, 33]. Therefore, failing to tackle the camouflaged fraudsters would sabotage the performance of GNN-based fraud detectors. Though some recent works [4, 9, 13, 25, 41] have noticed similar challenges, their solutions either fail to fit the fraud detection problems or break the end-to-end learning fashion of GNNs.

To demonstrate the challenges induced by camouflaged fraudsters during the neighbor aggregation of GNNs, as shown in Figure 1, we construct a graph with two relations and two types of entities. The relation can be any common attributes supposing to be shared by similar entities (e.g., the *User-IP-User* relation connects entities with the same IP address). There are two types of *camouflages* as follows: **1) Feature camouflage:** smart fraudsters may adjust their behaviors [8, 10], add special characters in reviews [19, 41] (so-called spamouflage), or employ deep language generation models [15] to gloss over explicit suspicious outcomes. Like Figure 1 shows, a fraudster can add some special characters to a fake review, which helps to bypass feature-based detectors [41]. **2)**

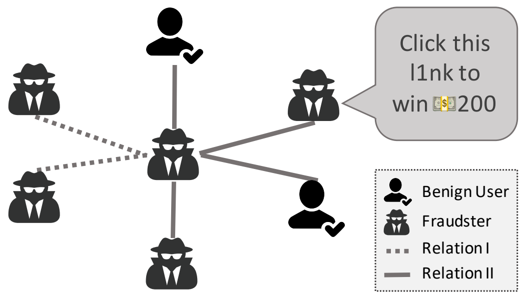

Figure 1: Two types of fraudster camouflage. (1) Feature camouflage: fraudsters add special characters to the text and make it delusive for feature-based spam detectors. (2) Relation camouflage: center fraudster connects to many benign entities under Relation II to attenuate its suspiciousness.

Relation camouflage: previous works [16, 49] show that crowd workers are actively committing opinion fraud on online social networks. They can probe the graphs used by defenders [43] and adjust their behavior to alleviate the suspiciousness [44]. Specifically, these crafty fraudsters camouflage themselves via connecting to many benign entities (i.e., posting regular reviews or connecting to reputable users). As Figure 1 shows, under Relation II, there are more benign entities than fraudsters.

Directly applying GNNs to graphs with camouflaged fraudsters will hamper the neighbor aggregation process of GNNs. As Figure 1 shows, if we aggregate neighbors with the intriguing reviews as node features, it will probably smooth out the suspiciousness of the center fraudster [13, 25]. Similarly, if we aggregate all neighbors under Relation II, where there are more dissimilar neighbors, it will eliminate the suspiciousness of the center fraudster.

Considering the agility of real-world fraudsters [8, 10], designing GNN-based detectors that exactly capture these camouflaged fraudsters is impractical. Therefore, based on the outcomes of two camouflages and the aggregation process of GNNs, we propose *three* neural modules to enhance the GNNs against the camouflages. **1)** For the feature camouflage, we propose a **label-aware similarity measure** to find the most similar neighbors based on node features. Specifically, we design a neural classifier as a similarity measure, which is directly optimized according to experts with domain knowledge (i.e., annotated data). **2)** For the relation camouflage, we devise a **similarity-aware neighbor selector** to select the similar neighbors of a center node within a relation. Furthermore, we leverage reinforcement learning (RL) to adaptively find the optimal neighbor selection threshold along with the GNN training process. **3)** We utilize the neighbor filtering thresholds learned by RL to formulate a **relation-aware neighbor aggregator** which combines neighborhood information from different relations and obtains the final center node representation.

We integrate above three modules together with general GNN frameworks and name our model as CAmouflage REsistant Graph Neural Network (CARE-GNN). Experimental results on two real-world fraud datasets demonstrate that our model boosts the GNN performance on graphs with camouflaged fraudsters. The proposed neighbor selector can find optimal neighbors and CARE-GNN outperforms state-of-the-art baselines under various settings.

Table 1: Glossary of Notations.

Symbol	Definition
$\mathcal{G}; \mathcal{V}; \mathcal{E}; \mathcal{X}$	Graph; Node set; Edge set; Node feature set
$y_v; Y$	Label for node v; Node label set
$r; R$	Relation; Total number of relations
$l; L$	GNN layer number; Total number of layers
$b; B$	Training batch number; Total number of batches
$e; E$	Training epoch number; Total number of epochs
$\mathcal{V}_{train}; \mathcal{V}_b$	Nodes in the training set; Node set at batch b
$\mathcal{E}_r^{(l)}$	Edge set under relation r at the l-th layer
$\mathbf{h}_v^{(l)}$	The embedding of node v at the l-th layer
$\mathbf{h}_{v,r}^{(l)}$	The embedding of node v under relation r at the l-th layer
$\mathcal{D}^{(l)}(v, v')$	The distance between node v and v' at the l-th layer
$S^{(l)}(v, v')$	The similarity between node v and v' at the l-th layer
$p_r^{(l)} \in P$	The filtering threshold for relation r at the l-th layer
$a_r^{(l)} \in A; \tau$	RL action space; Action step size
$G(\mathcal{D}_r^{(l)})$	Average neighbor distances for relation r at the l-th layer
$f(p_r^{(l)}, a_r^{(l)})$	RL reward function
$\text{AGG}_r^{(l)}$	Intra-relation aggregator for relation r at the l-th layer
$\text{AGG}_{all}^{(l)}$	Inter-relation aggregator at the l-th layer
$\mathbf{z}_v$	Final embedding for node v

We highlight the advantages of CARE-GNN as follows:

- **Adaptability.** CARE-GNN adaptively selects best neighbors for aggregation given arbitrary multi-relation graph.
- **High-efficiency.** CARE-GNN has a high computational efficiency without attention and deep reinforcement learning.
- **Flexibility.** Many other neural modules and external knowledge can be plugged into the CARE-GNN.

2 PROBLEM DEFINITION

In this section, we first define the multi-relation graph and the graph-based fraud detection problem. Then, we introduce how to apply GNN to fraud detection problems. All important notations in this paper are summarized in Table 1.

Definition 2.1. Multi-relation Graph. We define a multi-relation graph as $\mathcal{G} = \left\{\mathcal{V}, \mathcal{X}, \{\mathcal{E}_r\}|_{r=1}^{R}, Y\right\}$, where $\mathcal{V}$ is the set of nodes $\{v_1, \dots, v_n\}$. Each node v_i has a d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ represents a set of all node features. $e_{i,j}^r = (v_i, v_j) \in \mathcal{E}_r$ is an edge between v_i and v_j with a relation $r \in \{1, \cdots, R\}$. Note that an edge can be associated with multiple relations and there are R different types of relations. Y is the a set of labels for each node in $\mathcal{V}$.

Definition 2.2. Fraud Detection on Graph. For the fraud detection problem, the node v represents the target entity whose suspiciousness needs to be justified. For example, it can be a review on the review website [19, 29] or a transaction in the trading system [23, 37]. The node has a label $y_v \in \{0, 1\} \in Y$ where 0 represents *benign* and 1 represents *suspicious*. The relations R are rules, interactions, or shared attributes between nodes, e.g., two reviews from the same user [25] or transactions from the same devices [24]. The graph-based fraud detection problem is a semi-supervised binary node classification problem on the graph. Graph-based fraud detectors are trained based on the labeled node information along

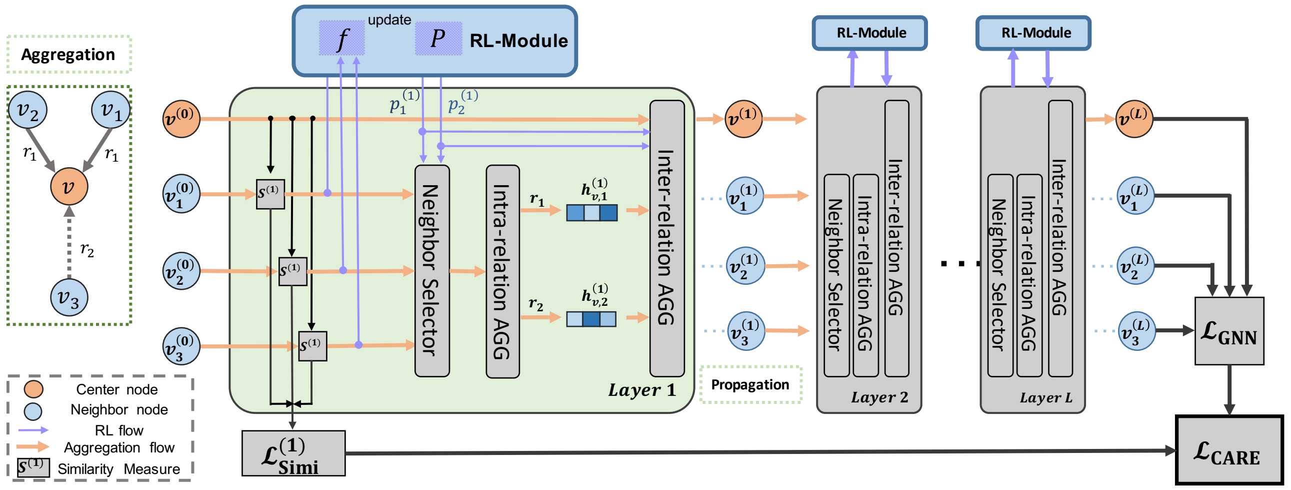

Figure 2: The aggregation process of proposed CARE-GNN at the training phase.

with the graph composed of multi-relations. The trained models are then used to predict the suspiciousness of unlabeled nodes.

Definition 2.3. GNN-based Fraud Detection. A Graph Neural Network (GNN) is a deep learning framework to embed graph-structured data via aggregating the information from its neighboring nodes [12, 17, 34]. Based on the defined multi-relation graph in Definition 2.2, we unify the formulation of GNNs from the perspective of neighbor aggregation (as shown in the left side of Figure 2):

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{h}_v^{(l-1)} \oplus \text{AGG}^{(l)}\left(\{\mathbf{h}_{v',r}^{(l-1)} : (v, v') \in \mathcal{E}_r^{(l)}\}\Big|_{r=1}^{R}\right)\right). \quad (1)$$

For a center node v, $\mathbf{h}_v^{(l)}$ is the hidden embedding at l-th layer and $\mathbf{h}_v^{(0)} = \mathbf{x}_i$ is the input feature. $\mathcal{E}_r^{(l)}$ denotes edges under relation r at the l-th layer. $\mathbf{h}_{v',r}^{(l-1)}$ is the embedding of neighboring node v' under relation r. AGG represents the aggregation function that mapping the neighborhood information from different relations into a vector, e.g., mean aggregation [12] and attention aggregation [34]. $\oplus$ is the operator that combines the information of v and its neighboring information, e.g., concatenation or summation [12].

For fraud detection problems, we first construct a multi-relation graph based on domain knowledge. Then, the GNN is trained with partially labeled nodes supervised by binary classification loss functions. Instead of directly aggregating the neighbors for all relations, we separate the aggregation part as *intra-relation* aggregation and *inter-relation* aggregation process. During the intra-relation aggregation process, the embedding of neighbors under each relation is aggregated simultaneously. Then, the embeddings for each relation are combined during the inter-relation aggregation process. Finally, the node embeddings at the last layer are used for prediction.

3 PROPOSED MODEL

3.1 Model Overview

The proposed CARE-GNN has three neural modules and its pipeline is shown in Figure 2. For a center node v, we first compute its neighbor similarities based with proposed label-aware similarity measure (Section 3.2). Then we filter the dissimilar neighbors under each relation with the proposed neighbor selector (Section 3.3). The neighbor selector is optimized using reinforcement learning during training the GNN (purple module in Figure 2). At the aggregation step, we first use the intra-relation aggregator to aggregate neighbor embeddings under each relation. Then, we combine embeddings across different relations with the inter-relation aggregator (Section 3.4). The optimization steps and the algorithm procedure are presented in Section 3.5 and Algorithm 1, respectively.

3.2 Label-aware Similarity Measure

Previous studies have introduced various fraudster camouflage types from behavior [8, 10] and semantic [15, 41] perspectives. Those camouflages could make the features of fraudsters and benign entities similar to each other, and further mislead GNNs to generate uninformative node embeddings. To tackle those node feature camouflages, we deem that an effective similarity measure is needed to filter the camouflaged neighbors before applying GNNs. Previous works have proposed unsupervised similarity metrics like *Cosine Similarity* [25] or Neural Networks [45]. However, many fraud problems like financial fraud and opinion fraud require extra domain knowledge to identify fraud instances. For example, in opinion fraud, unsupervised similarity measures could not identify the camouflaged fake reviews, which are even indistinguishable by humans [15]. Therefore, we need a parameterized similarity measure to compute node similarity with supervised signals from domain experts (e.g., high-fidelity data annotations).

For the parameterized similarity measure, AGCN [20] employs a *Mahalanobis distance* plus a *Gaussian kernel*, and DIAL-GNN [6] uses the parameterized *cosine similarity*. However, those two types of measures suffer from high time complexity $O(|\mathcal{V}|\bar{D}d)$, where $\bar{D}$ is the average degree of nodes which is extremely high in real-world graphs (see Table 2) and d is the feature dimension.

Label-aware Similarity Measure. Inspired by LAGCN [4] which uses a Multi-layer Perceptron (MLP) as the edge label predictor, we employ a one-layer MLP as the node label predictor at each layer and use the l_1-distance between the prediction results of two nodes as their similarity measure. For a center node v under relation r at the l-th layer and edge $(v, v') \in \mathcal{E}_r^{(l-1)}$, the distance between v and v' is the l_1-distance of two embeddings:

$$\mathcal{D}^{(l)}(v, v') = \left\| \sigma\left(MLP^{(l)}(\mathbf{h}_v^{(l-1)})\right) - \sigma\left(MLP^{(l)}(\mathbf{h}_{v'}^{(l-1)})\right) \right\|_1, \quad (2)$$

and we can define the similarity measure as:

$$S^{(l)}(v, v') = 1 - \mathcal{D}^{(l)}(v, v'), \quad (3)$$

where each layer has its own similarity measure. The input of MLP at the l-th layer is the node embedding at the previous layer, and the output of MLP is a scalar which is then fed into a non-linear activation function σ (we use tanh in our work). To save the computational cost, we only take the embedding of the node itself as the input instead of using combined embeddings like the LAGCN [4]. Therefore, taking the $S_r^{(1)}(v, v')$ as an example where the input is the raw feature, the time complexity of the proposed similarity measure reduces significantly from $O(|\mathcal{V}|\bar{D}d)$ to $O(|\mathcal{V}|d)$ since it predicts the node label solely based on its feature.

Optimization. To train the similarity measure together with GNNs, a heuristic approach is to append it as a new layer before the aggregation layer of GCN [20]. However, if the similarity measure could not effectively filter the camouflaged neighbors at the first layer, it will hamper the performance of following GNN layers. Consequently, the MLP parameters cannot be well-updated through the back-propagation process. To train the similar measure with a direct supervised signal from labels, like [35], we define the cross-entropy loss of the MLP at l-layer as:

$$\mathcal{L}_{\text{Simi}}^{(l)} = \sum_{v \in \mathcal{V}} -\log\left(y_v \cdot \sigma\left(MLP^{(l)}(\mathbf{h}_v^{(l)})\right)\right). \quad (4)$$

During the training process, the similarity measure parameters are directly updated through the above loss function. It guarantees similar neighbors can be quickly selected within the first few batches and help regularize the GNN training process.

3.3 Similarity-aware Neighbor Selector

Given the similarity scores between the center node and its neighbors with Eq. (3), we should select similar neighbors (i.e., filter camouflaged ones) to improve the capability of GNNs. According to the relation camouflage, fraudsters may connect to different amounts of benign entities under different relations [44]. However, since data annotation is costly for real-world fraud detection problems, computing the number of similar neighbors under each relation through data labeling is impossible. We should devise an adaptive filtering/sampling criteria to select an optimal amount of similar neighbors automatically. Thus, we design a similarity-aware neighbor selector. It selects similar neighbors under each relation using *top-p* sampling with an adaptive filtering threshold. We also devise a reinforcement learning (RL) algorithm to find optimal thresholds during the GNN training process.

3.3.1 Top-p Sampling. We employ *top-p* sampling to filter camouflaged neighbors under each relation. The filtering threshold for relation r at the l-th layer is $p_r^{(l)} \in [0, 1]$. The closed interval means we could discard or keep all neighbors of a node under a relation. Specifically, during the training phase, for a node v in current batch under relation r, we first compute a set of similarity scores $\{S^{(l)}(v, v')\}$ using Eq. (3) at the l-th layer where $(v, v') \in \mathcal{E}_r^{(l)}$. $\mathcal{E}_r^{(l)}$ is a set of edges under relation r at the l-th layer. Then we rank its neighbors based on $\{S^{(l)}(v, v')\}$ in descending order and take the first $p_r^{(l)} \cdot |\{S^{(l)}(v, v')\}|$ neighbors as the selected neighbors at the l-th layer. All other nodes are discarded at the current batch and will not attend the aggregation process. The *top-p* sampling process is applied to the center node at every layer for each relation.

3.3.2 Finding the Optimal Thresholds with RL. Previous works [6, 25] set the filtering threshold as a hyperparameter and tune it with validation to find the optimal value. However, their models are built upon homogeneous benchmark graphs, and without noise induced by camouflaged fraudsters. However, owing to the multi-relation graph of fraud problems as well as the relation camouflage problem, we need an automatic approach to find the optimal threshold $p_r^{(l)}$ for each relation. Since $p_r^{(l)}$ is a probability and has no gradient, we cannot use back-propagation from the classification loss to update it. Meanwhile, given a $p_r^{(l)}$, it is infeasible to estimate the quality of selected neighbors solely based on the similarity scores under the current batch/epoch. To overcome the above challenges, we propose to employ a reinforcement learning (RL) framework to find optimal thresholds.

Concretely, we formulate the RL process as a Bernoulli Multi-armed Bandit (BMAB) $\mathcal{B}(A, f, T)$ between the neighbor selector and the GNN with the similarity measure. A is the action space, f is the reward function, and T is the terminal condition [36]. Given an initial $p_r^{(l)}$, the neighbor selector choose to increase or decrease $p_r^{(l)}$ as actions and the reward is dependent on the average distance differences between two consecutive epochs. Next, we introduce the details of each BMAB component:

- **Action.** The action represents how RL updates the $p_r^{(l)}$ based on the reward. Since $p_r^{(l)} \in [0, 1]$, we define the action $a_r^{(l)}$ as plus or minus a fixed small value $\tau \in [0, 1]$ from $p_r^{(l)}$.
- **Reward.** The optimal $p_r^{(l)}$ is expected to find the most similar (i.e., minimum distances in Eq. (2)) neighbors of a center node under relation r at the l-th layer. We cannot sense the state of GNN due to its black-box nature; thus, we design a binary stochastic reward solely based on the average distance differences between two consecutive epochs. The average neighbor distances for relation r at the l-th layer for epoch e is:

$$G(\mathcal{D}_r^{(l)})^{(e)} = \frac{\sum_{v \in \mathcal{V}_{train}} \mathcal{D}_r^{(l)}(v, v')^{(e)}}{|\mathcal{V}_{train}|}. \quad (5)$$

Then, we can define the reward for epoch e as:

$$f(p_r^{(l)}, a_r^{(l)})^{(e)} = \begin{cases} +1, G(\mathcal{D}_r^{(l)})^{(e-1)} - G(\mathcal{D}_r^{(l)})^{(e)} \geq 0, \\ -1, G(\mathcal{D}_r^{(l)})^{(e-1)} - G(\mathcal{D}_r^{(l)})^{(e)} < 0. \end{cases} \quad (6)$$

The reward is positive when the average distance of newly selected neighbors at epoch e is less than that of the previous epoch, and vice versa. It is not easy to estimate the cumulative reward. Thus, we use the immediate reward to update the action greedily without exploration. Concretely, we increase $p_r^{(l)}$ with a positive reward and decrease it vice versa.

- **Terminal.** We define the terminal condition for RL as:

$$\Big|\sum_{e-10}^{e} f(p_r^{(l)}, a_r^{(l)})^{(e)}\Big| \le 2,\ \ where\ e \ge 10. \tag{7}$$

It means that the RL converges in the recent ten epochs and indicates an optimal threshold $p_r^{(l)}$ is discovered. After the RL module terminates, the filtering thresholds are fixed as the optimal one until the convergence of GNN.

Discussion. Different node classes may have different amounts of similar neighbors under the same relation. For instance, as Table 2 shows, under the *R-S-R* relation of the Yelp dataset, for positive nodes, only 5% of their neighbors have the same label. This is due to the class-imbalance nature of fraud problems and the relation camouflage of fraudsters. According to the cost-sensitive learning research [30], misclassifying a fraudster has a much higher cost to defenders than misclassifying a benign entity. Meanwhile, a large number of benign entities already fuel sufficient information for the classifier. Therefore, to accelerate the training process, we compute the filtering thresholds by only considering positive center nodes (i.e., fraudsters) and apply them for all node classes. The complete RL process is shown in Lines 15-19 of Algorithm 1. The experiment results in Section 4.4 verify the RL effectiveness.

3.4 Relation-aware Neighbor Aggregator

After filtering neighbors under each relation, the next step is to aggregate the neighbor information from different relations. Previous methods adopt attention mechanism [23, 37, 48] or devise weighting parameters [24] to learn the relation weights during aggregating information from different relations. However, supposing we have selected the most similar neighbors under each relation, the attention coefficients or weighting parameters should be similar among different relations. Thus, to save the computational cost while retaining the relation importance information, we directly apply the optimal filtering threshold $p_r^{(l)}$ learned by the RL process as the inter-relation aggregation weights. Formally, under relation r at the l-th layer, after applying the *top-p* sampling, for node v, we define the **intra-relation** neighbor aggregation as follows:

$$\mathbf{h}_{v,r}^{(l)} = \text{ReLU}\left(\text{AGG}_r^{(l)}\left(\left\{\mathbf{h}_{v'}^{(l-1)} : (v,v') \in \mathcal{E}_r^{(l)}\right\}\right)\right), \tag{8}$$

where a mean aggregator is used for all $\text{AGG}_r^{(l)}$. Then, we define the **inter-relation** aggregation as follows:

$$\mathbf{h}_{v}^{(l)} = \text{ReLU}\left(\text{AGG}_{all}^{(l)}\left(\mathbf{h}_v^{(l-1)} \oplus \{p_r^{(l)} \cdot \mathbf{h}_{v,r}^{(l)}\}|_{r=1}^{R}\right)\right), \tag{9}$$

where $\mathbf{h}_v^{(l-1)}$ is the center node embedding at the previous layer, $\mathbf{h}_{v,r}^{(l)}$ is the intra-relation neighbor embedding at the l-th layer and $p_r^{(l)}$ is filtering threshold of relation r which is directly used as its inter-relation aggregation weight. $\oplus$ denotes the embedding summation operation. AGG_{all}^{l} can be any type of aggregator, and we test them in Section 4.3.

3.5 Proposed CARE-GNN

Optimization. For each node v, its final embedding is the output of the GNN at the last layer $\mathbf{z}_v = \mathbf{h}_v^{(L)}$. We can define the loss of GNN as a cross-entropy loss function:

$$\mathcal{L}_{\text{GNN}} = \sum_{v \in \mathcal{V}} -\log\left(y_v \cdot \sigma(MLP(\mathbf{z}_v))\right). \tag{10}$$

Together with the loss function of the similarity measure in Eq. (4), we define the loss of CARE-GNN as:

$$\mathcal{L}_{\text{CARE}} = \mathcal{L}_{\text{GNN}} + \lambda_1 \mathcal{L}_{\text{Simi}}^{(1)} + \lambda_2 ||\Theta||_2, \tag{11}$$

where $||\Theta||_2$ is the $L2$-norm of all model parameters, λ_1 and λ_2 are weighting parameters. Since the neighbor filtering process at the first layer is critical to both GNN and similarity measures in the following layers, we only use the similarity measure loss at the first layer to update the parameterized similarity measure in Eq. (3).

Algorithm Description. Algorithm 1 shows the training process of the proposed CARE-GNN. Given a multi-relational fraud graph, we employ the mini-batch training technique [11] as the result of its large scale. In the beginning, we randomly initialize the parameters of the similarity measure module and GNN module. We initialize all filtering thresholds as 0.5 (Line 2). For each batch of nodes, we first compute the neighbor similarities using Eq. (3) (Line 7) and

Algorithm 1: CARE-GNN: Camouflage Resistant GNN.

```
Input : An undirected multi-relation graph with node
        features and labels: $\mathcal{G} = \{\mathcal{V}, \mathbf{X}, \{\mathcal{E}_r\}|_{r=1}^{R}, Y\}$;
        Number of layers, batches, epochs: $L, B, E$;
        Parameterized similarity measures: $\{S^{(l)}(\cdot,\cdot)\}|_{l=1}^{L}$;
        Filtering thresholds: $P = \{p_1^{(l)}, \dots, p_R^{(l)}\}|_{l=1}^{L}$;
        Intra-R aggregators: $\{\text{AGG}_r^{(l)}\}|_{r=1}^{R}, \forall l \in \{1,\dots,L\}$;
        Inter-R aggregators: $\{\text{AGG}_{all}^{(l)}\}, \forall l \in \{1,\dots,L\}$.
Output: Vector representations $\mathbf{z}_v, \forall v \in \mathcal{V}_{train}$.
1  // Initialization
2  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}; p_r^{(0)} = 0.5, \mathcal{E}_r^{(0)} = \mathcal{E}, \forall r \in \{1,\dots,R\}$;
3  for $e = 1, \cdots, E$ do                          // Train CARE-GNN
4      for $b = 1, \cdots, B$ do
5          for $l = 1, \cdots, L$ do
6              for $r = 1, \cdots, R$ do
7                  $S^{(l)}(v,v') \leftarrow$ Eq. (3) , $\forall (v,v') \in \mathcal{E}_r^{(l-1)}$;
8                  $\mathcal{E}_r^{(l)} \leftarrow$ top-p sampling (Section 3.3.1);
9                  $\mathbf{h}_{v,r}^{(l)} \leftarrow$ Eq. (8) $\forall v \in \mathcal{V}_b$;      // Intra-R AGG
10             $\mathbf{h}_v^{(l)} \leftarrow$ Eq. (9) $\forall v \in \mathcal{V}_b$;          // Inter-R AGG
11             $\mathcal{L}_{\text{Simi}}^{(1)} \leftarrow$ Eq. (4);                // Simi loss
12         $\mathbf{z}_v \leftarrow \mathbf{h}_v^{(L)}, \forall v \in \mathcal{V}_b$;   // Batch node embeddings
13         $\mathcal{L}_{\text{GNN}} \leftarrow$ Eq. (10);                       // GNN loss
14         $\mathcal{L}_{\text{CARE}} \leftarrow$ Eq. (11);                  // CARE-GNN loss
15     for $l = 1, \cdots, L$ do                         // RL Module
16         for $r = 1, \cdots, R$ do
17             if Eq. (7) is False then
18                 $f(p_r^{(l)}, a_r^{(l)})^{(e)} \leftarrow$ Eqs.(5) and (6);
19                 $p_r^{(l)} \leftarrow p_r^{(l)} + f(p_r^{(l)}, a_r^{(l)})^{(e)} \cdot \tau$;
```

then filter the neighbors using *top-p* sampling (Line 8). Then, we can compute the intra-relation embeddings (Line 9), inter-relation embeddings (Line 10), loss functions (Lines 11-14) for the current batch, respectively. As for the RL process, we assign random actions for the first epoch since it has no reference. From the second epoch, we update $p_r^{(l)}$ according to Lines 15-19.

4 EXPERIMENTS

In the experiment section, we mainly present:

- how we construct multi-relation graphs upon different fraud data (Section 4.1.2);
- camouflage evidences in real-world fraud data (Section 4.2);
- the performance comparison over baselines and CARE-GNN variants (Section 4.3);
- the learning process and explanation of the RL algorithm (Section 4.4);
- the sensitivity study of hyper-parameters and their effects on model designing (Section 4.5).

4.1 Experimental Setup

Table 2: Dataset and graph statistics.

	#Nodes (Fraud%)	Relation	#Edges	Avg. Feature Similarity	Avg. Label Similarity
Yelp	45,954 (14.5%)	*R-U-R*	49,315	0.83	0.90
		R-T-R	573,616	0.79	0.05
		R-S-R	3,402,743	0.77	0.05
		ALL	3,846,979	0.77	0.07
Amazon	11,944 (9.5%)	*U-P-U*	175,608	0.61	0.19
		U-S-U	3,566,479	0.64	0.04
		U-V-U	1,036,737	0.71	0.03
		ALL	4,398,392	0.65	0.05

4.1.1 Dataset. We use the Yelp review dataset [29] and Amazon review dataset [26] to study the fraudster camouflage and GNN-based fraud detection problem. The Yelp dataset includes hotel and restaurant reviews filtered (spam) and recommended (legitimate) by Yelp. The Amazon dataset includes product reviews under the Musical Instruments category. Similar to [47], we label users with more than 80% helpful votes as benign entities and users with less than 20% helpful votes as fraudulent entities. Though previous works have proposed other fraud datasets like Epinions [18] and Bitcoin [40], they only contain graph structures and compacted features, with which we cannot build meaningful multi-relation graphs. In this paper, we conduct a spam review detection (fraudulent user detection resp.) task on the Yelp dataset (Amazon dataset resp.), which is a binary classification task. We take 32 handcrafted features from [29] (25 handcrafted features from [47] resp.) as the raw node features for Yelp (Amazon resp.) dataset. Table 2 shows the dataset statistics.

4.1.2 Graph Construction. **Yelp:** based on previous studies [27, 29] which show that opinion fraudsters have connections in user, product, review text, and time, we take reviews as nodes in the graph and design three relations: 1) *R-U-R*: it connects reviews posted by the same user; 2) *R-S-R*: it connects reviews under the same product with the same star rating (1-5 stars); 3) *R-T-R*: it connects two reviews under the same product posted in the same month. **Amazon:** similarly, we take users as nodes in the graph and design three relations: 1) *U-P-U*: it connects users reviewing at least one same product; 2) *U-S-V*: it connects users having at least one same star rating within one week; 3) *U-V-U*: it connects users with top 5% mutual review text similarities (measured by TF-IDF) among all users. The number of edges belonging to each relation is shown in Table 2.

4.1.3 Baselines. To verify the ability of CARE-GNN in alleviating the negative influence induced by camouflaged fraudsters, we compare it with various GNN baselines under the semi-supervised learning setting. We select GCN [17], GAT [34], RGCN [31], and GraphSAGE [12] to represent general GNN models. We choose GeniePath [23], Player2Vec [48], SemiGNN [37], and GraphConsis [25] as four state-of-the-art GNN-based fraud detectors. Their detailed introduction can be found in Section 5. We also implement several variants of CARE-GNN: CARE-*Att*, CARE-*Weight*, and CARE-*Mean*, and they differ from each other in Attention [34], Weight [24], and Mean [12] inter-relation aggregator respectively.

Among those baselines, GCN, GAT, GraphSAGE, and GeniePath are run on homogeneous graphs (i.e., Relation *ALL* in Table 2) where all relations are merged together. Other models are run on multi-relation graphs where they handle information from different relations in their approaches.

4.1.4 Experimental Setting. From Table 2, we can see that the percentage of fraudsters are small in both datasets. Meanwhile, real-world graphs usually have great scales. To improve the training efficiency and avoid overfitting, we employ mini-batch training [11] and under-sampling [22] techniques to train CARE-GNN and other baselines. Specifically, under each mini-batch, we randomly sample the same number of negative instances as the number of positive instances. We also study the sample ratio sensitivity in Section 4.5.

We use unified node embedding size (64), batch size (1024 for Yelp, 256 for Amazon), number of layers(1), learning rate (0.01), optimizer (Adam), and L2 regularization weight ($\lambda_2 = 0.001$) for all models. For CARE-GNN and its variants, we set the RL action step size (τ) as 0.02 and the similarity loss weight (λ_1) as 2. In Section 4.5, we present the sensitivity study for the number of layers, embedding size, and λ_1.

4.1.5 Implementation. For the GCN, GAT, RGCN, GraphSAGE, GeniePath, we use the source code provided by authors. For Player2Vec, SemiGNN, and GraphConsis, we use the open-source implementations[2]. We implement CARE-GNN with Pytorch. All models are running on Python 3.7.3, 2 NVIDIA GTX 1080 Ti GPUs, 64GB RAM, 3.50GHz Intel Core i5 Linux desktop.

4.1.6 Evaluation Metric. Since the Yelp dataset has imbalanced classes, and we focus more on fraudsters (positive instances), like previous work [29], we utilize ROC-AUC (AUC) and Recall to evaluate the overall performance of all classifiers. AUC is computed based on the relative ranking of prediction probabilities of all instances, which could eliminate the influence of imbalanced classes.

[2]https://github.com/safe-graph/DGFraud

Table 3: Fraud detection performance (%) on two datasets under different percentage of training data.

	Metric	Train%	GCN	GAT	RGCN	Graph-SAGE	Genie-Path	Player-2Vec	Semi-GNN	Graph-Consis	CARE-*Att*	CARE-*Weight*	CARE-*Mean*	CARE-GNN
Yelp	AUC	5%	54.98	56.23	50.21	53.82	56.33	51.03	53.73	61.58	66.08	71.10	69.83	**71.26**
		10%	50.94	55.45	55.12	54.20	56.29	50.15	51.68	62.07	70.21	71.02	71.85	**73.31**
		20%	53.15	57.69	55.05	56.12	57.32	51.56	51.55	62.31	73.26	74.32	73.32	**74.45**
		40%	52.47	56.24	53.38	54.00	55.91	53.65	51.58	62.07	74.98	74.42	74.77	**75.70**
	Recall	5%	53.12	54.68	50.38	54.25	52.33	50.00	52.28	62.60	63.52	66.64	**68.09**	67.53
		10%	51.10	52.34	51.75	52.23	54.35	50.00	52.57	62.08	67.38	68.35	**68.92**	67.77
		20%	53.87	53.20	50.92	52.69	54.84	50.00	52.16	62.35	68.34	69.07	**69.48**	68.60
		40%	50.81	54.52	50.43	52.86	50.94	50.00	50.59	62.08	71.13	70.22	69.25	**71.92**
Amazon	AUC	5%	74.44	73.89	75.12	70.71	71.56	76.86	70.25	85.46	89.49	89.36	89.35	**89.54**
		10%	75.25	74.55	74.13	73.97	72.23	75.73	76.21	85.29	**89.58**	89.37	89.43	89.44
		20%	75.13	72.10	75.58	73.97	71.89	74.55	73.98	85.50	89.58	**89.68**	89.34	89.45
		40%	74.34	75.16	74.68	75.27	72.65	56.94	70.35	85.50	89.70	89.69	89.52	**89.73**
	Recall	5%	65.54	63.22	64.23	69.09	65.56	50.00	63.29	85.49	88.22	88.31	88.02	**88.34**
		10%	67.81	65.84	67.22	69.36	66.63	50.00	63.32	85.38	87.87	**88.36**	88.12	88.29
		20%	66.15	67.13	65.08	70.30	65.08	50.00	61.28	85.59	88.40	**88.60**	88.00	88.27
		40%	67.45	65.51	67.68	70.16	65.41	50.00	62.89	85.53	88.41	88.45	88.22	**88.48**

4.2 Camouflage Evidence

We analyze fraudster camouflage using two metrics introduced in [25]. For the feature camouflage, we compute the feature similarity of neighboring nodes based on their feature vectors' Euclidean distance, ranging from 0 to 1. The average feature similarity is normalized w.r.t. the total number of edges, which is presented in Table 2. We observe that the averaged similarity scores under all relations are high. High feature similarity implies that fraudsters camouflage their features in a similar way to benign nodes. Moreover, the minor feature similarity difference across different relations proves that the unsupervised similarity measure cannot effectively discriminate fraudsters and benign entities. For instance, the label similarity difference between *R-U-R* and *R-T-R* is 0.85, but the feature similarity difference is only 0.04.

For the relation camouflage, we study it by calculating the label similarity based on whether two connected nodes have the same label. The label similarity is normalized w.r.t. the total number of edges. The average label similarity for each relation is shown in Table 2. High label similarity score implies that the fraudsters fail to camouflage, and low score implies that fraudsters camouflage successfully. We observe that only *R-U-R* relation has a high label similarity score, while the other relations have label similarity scores less than 20%. It suggests that we need to select different amounts of neighbors for different relations to facilitate the GNN aggregation process. Meanwhile, we should distinguish relations in order to prevent fraudsters from camouflaging.

4.3 Overall Evaluation

Table 3 shows the performance of proposed CARE-GNN and various GNN baselines under the fraud detection task on two datasets. We report the best testing results after thirty epochs. We observe that CARE-GNN outperforms other baselines under most of the training proportions and metrics.

Single-relation vs. Multi-relation. Among all GNN baselines in Table 3, GCN, GAT, GraphSAGE, and GeniePath run on single-relation (i.e., homogeneous) graph where all relations are merged together (*ALL* in Table 2). Other baselines are built upon multi-relation graphs. The performances of single-relation GNNs are better than Player2Vec and SemiGNN, which indicates previously designed fraud detection methods are not suitable for multi-relation graphs. Among the multi-relation GNNs, GraphConsis outperforms all other multi-relation GNNs. The reason is that GraphConsis samples the neighbors based on the node features before aggregating them. Better than GraphConsis, CARE-GNN and its variants adopt parameterized similarity measure and adaptive sampling thresholds, which could better identify and filter camouflaged fraudsters. It demonstrates that neighbor filtering is critical to GNNs when the graph contains many noises (i.e., dissimilar/camouflaged neighbors). Also, CARE-GNN has higher scores than all single-relation GNNs, suggesting that a noisy graph undermines the performance of multi-relation GNNs. A possible reason is the higher complexity of multi-relation GNNs comparing to single-relation ones.

Training Percentage. From Table 3, there is little performance gain for GNNs when increasing the training percentages. It demonstrates the advantage of semi-supervised learning, where a small amount of supervised signals is enough to train a good model. Meanwhile, with informative handcraft features as inputs for two datasets, GNNs are much easier to learn high-quality embeddings.

CARE-GNN Variants. The last four columns of Table 3 show the performance of CARE-GNN and its variants with different inter-relation aggregators. It is observed that those four models have similar performances under most training percentages and metrics. It verifies our assumption in Section 3.4 that the attention coefficients and relation weights will become unnecessary when we select similar neighbors under all relations. Moreover, for the Yelp dataset, the CARE-*Att* has worse performances under a smaller training percentage (e.g., 5%). While for CARE-GNN, since it does not need to train extra attention weights, it attains the best performance against other variants. The first column of Figure 3 presents more evidence that the relation weights finally become equal for all relations under both datasets. The better performance of CARE-GNN comparing to CARE-*Mean* shows that keeping the filtering

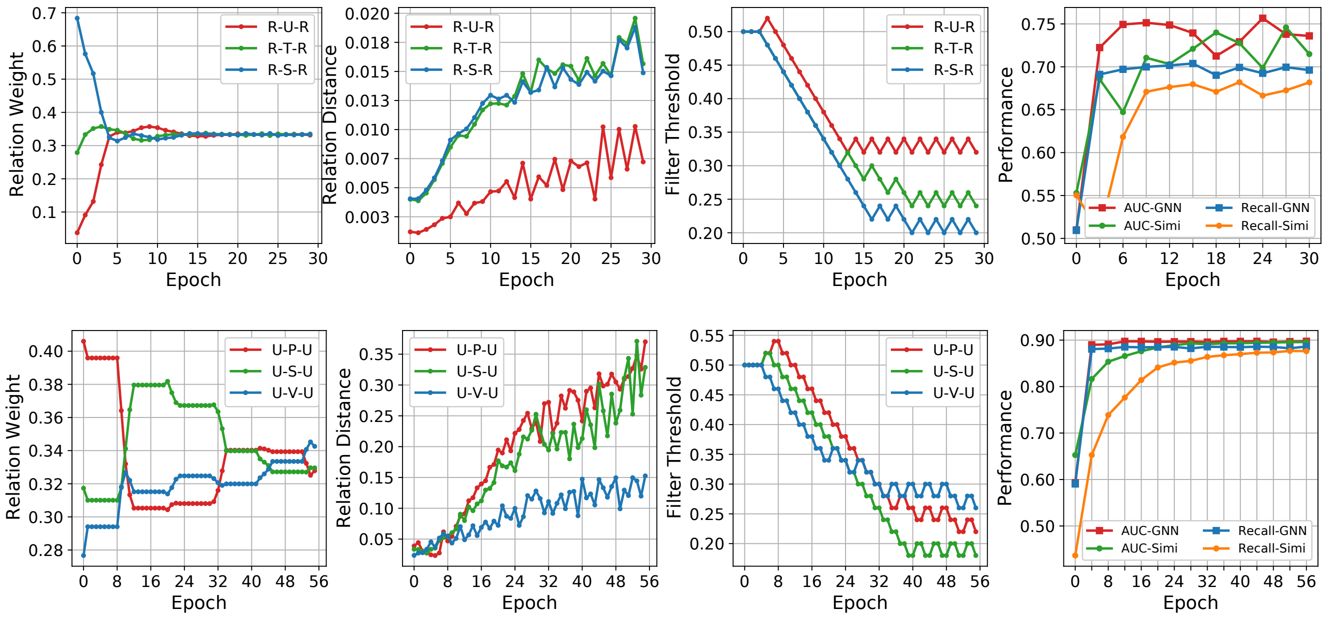

Figure 3: The training process and testing performance of CARE-*Weight* on Yelp (upper) and Amazon (lower) dataset.

threshold as inter-relation aggregation weights could enhance the GNN performance and reduce model complexity.

GNN vs. Similarity Measure (Figure 3 Column 4). Figure 3 Column 4 shows the testing performances solely based on the outputs of the GNN module and similarity measure module during training. For the Yelp dataset, GNN has better AUC and Recall than the similarity measure, which suggests that leveraging the structural information benefits the model to classify fraud and benign entities. For Amazon, the performance of GNN and the similarity measure are comparable with each other. It is because the input features provide enough information to discriminate fraudsters.

4.4 RL Process Analysis

In this paper, we jointly train the similarity measure and GNN together and employ RL to find the neighbor filtering thresholds adaptively. To present the RL process from different perspectives, in Figure 3, we plot the updating process of three parameters without terminating the RL process during training CARE-*Weight*. Since CARE-*Weight* learns the aggregation weight for each relation, plotting its training process instead of CARE-GNN could help understand the effects of our proposed GNN enhancement modules. During training, we also test the model every three epochs for Yelp (four epochs for Amazon) and plot the testing performance for both GNN and similarity measure at the last column of Figure 3.

Relation Weights (Figure 3 Column 1). We observe that the randomly initialized relation aggregation weights gradually converge to the same value as the neighbor selector updates its filtering thresholds and selects more similar neighbors under each relation. When neighbors under each relation provide similar information, their aggregation weights will be similar as well.

Relation Distance (Figure 3 Column 2). As the training epoch increases, it is clearly that the differences between neighbor distances under each relation (computed by Eq. (5)) become larger and comparable to each other. The reason is that the GNN projects the node embeddings to a broader range of space and makes them more distinguishable. As the model filters more noisy neighbors, the average distance across different relations become closer.

Neighbor Filtering Threshold (Figure 3 Column 3). We take 0.02 as the action step size; all thresholds are updated and converge to different values. When the filtering threshold oscillates for several rounds, it reaches the terminal condition in Eq. (7). For different datasets, the proposed RL algorithm could adaptively find the optimal filtering thresholds.

To demonstrate the advantage of the optimal neighbor filtering thresholds found by RL, in Figure 4, we plot the testing performances of three different neighbor selection criteria under two datasets. `Adaptive` filters neighbors using converged thresholds found by RL (as shown in Figure 3 Column 3); `Fixed-Half` keeps the *top* 50% similar neighbors under each relation and `Fixed-All` keeps all neighbors without filtering. It is illustrated that CARE-GNN with adaptive filtering thresholds is optimized faster than the other two neighbor selectors. Meanwhile, it has a better and smoother performance during training. It verifies the effectiveness of the proposed RL algorithm, which is able to find informative neighbors under each relation.

4.5 Hyper-parameter Sensitivity

Figure 5 shows the testing performance of CARE-GNN regarding four hyper-parameters on the Yelp dataset. From Figure 5(a), we observe that increasing the number of layers barely improves the performance of CARE-GNN. For the three-layer model, the CARE-GNN suffers the overfitting problem (Recall = 0.5). Therefore, the

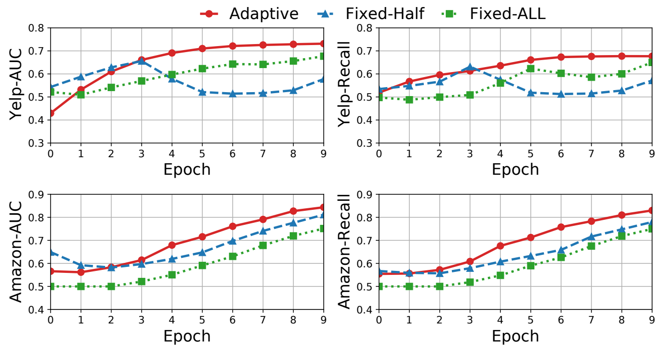

Figure 4: The testing AUC and Recall for CARE-GNN with different neighbor filtering methods during training.

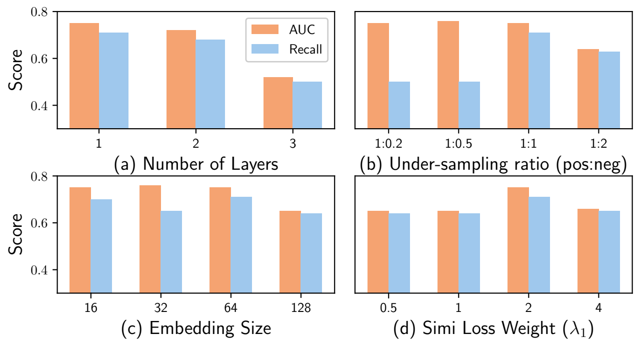

Figure 5: Parameter Sensitivity. For each parameter configuration, only the best results among 30 epochs are recorded.

one-layer model is not only able to save the computational cost but also achieve better classification results. Figure 5(b) presents the CARE-GNN performance under different under-sampling ratios as introduced in Section 4.1.4. Note that CARE-GNN is tested on an imbalanced test set. Moreover, CARE-GNN is overfitted when negative instances are less than positive ones (under 1:0.2 and 1:0.5, Recalls are equal to 0.5). An equal under-sampling ratio guarantees a good and fair performance of CARE-GNN. Figure 5(c) shows the influence of different embedding sizes. Embedding sizes with 16, 32, and 64 have comparable performance. Figure 5(d) illustrates the effects of different weighting values for the similarity loss (λ_1 in Eq. (11)). When the weight of similarity loss is doubled compared to which of GNN loss, CARE-GNN reaches the best performance. Therefore, the similarity measure is crucial for GNN training.

4.6 Discussion

Since the multi-relation graphs used in the experiments are very dense (average node degree > 150), one-layer CARE-GNN (which aggregates one-hop neighbors) has already utilized abundant information and thus can achieve excellent performance. CARE-GNN with more layers is suitable for sparse graphs. We improve the computational efficiency using multiple approaches: the light-weight similarity measure, the classic and fast RL framework, positive-node based neighbor selector, no attention mechanism, and mini-batch training with under-sampling. For CARE-GNN, each epoch only takes 17 seconds on Yelp (3 seconds on Amazon), and it has a great performance gain comparing to other baselines.

5 RELATED WORK

GNN and Its Enhancement. As the most popular deep learning framework on graph data, GNNs have two major types [42]: 1) Spectral-based GNNs (GCN [17], AGCN [20]): they turn a graph into a Laplacian matrix and make convolutional operations in the spectral domain. 2) Spatial-based GNNs (GAT [34], GraphSAGE [12]): they propagate the information based on the spatial relation (i.e., the adjacent nodes). Since spatial-based GNNs are more flexible, many GNN variants belong to this type. The proposed CARE-GNN is a spatial-based GNN as well.

To enhance the GNN performance on graphs with noisy nodes. One approach is the *graph structure learning* (GSL) [6, 9, 20]. Those works learn new graph structures from original graphs, which could better render the latent connections between nodes. Comparing to our work, those papers only investigate the single-relation benchmark datasets without camouflaged fraudsters. Our model filters dissimilar neighbors instead of learning new structures.

Another approach is the *metric learning* [4, 13]. Those works devise new metrics to measure the similarity between connect nodes and aggregate neighbors according to the metrics. Among those works, [4] proposes a neural network to predict the labels of neighboring nodes. [13] devises two metrics to measure the average neighborhood similarity and label similarity in a graph. However, those methods either have weak similarity metrics or fixed neighbor filtering thresholds, which need to be calibrated empirically. CARE-GNN proposed by us is more flexible which could learn the similarity metric based on domain knowledge. The relation filtering thresholds of CARE-GNN are optimized during training GNN which retains the end-to-end learning fashion.

GNN sampling methods [5, 46, 51] also filter the neighbors. While these works only consider selecting representative nodes to accelerate GNN training. For our work, taking account of domain knowledge and relational information, our goal is to filter dissimilar neighbors before aggregation, which could alleviate the negative effect of camouflaged fraudsters.

GNN-based Fraud Detection. Many GNN-based fraud detectors transfer the heterogeneous data into homogeneous data before applying GNNs. Fdgars [39] and GraphConsis [25] construct a single homo-graph based on multiple relations and employ GNNs to aggregate neighborhood information. GeniePath [23] learns convolutional layers and neighbor weights using LSTM and the attention mechanism [34]. GEM [24], SemiGNN [37], ASA [41], and Player2Vec [48] all construct multiple homo-graphs based on node relations in corresponding datasets. After aggregating neighborhood information with GNNs on each homo-graph, SemiGNN and Player2Vec adopt attention mechanism to aggregate node embeddings across multiple homo-graphs; while GEM learns weighting parameters for different homo-graphs, and ASA directly sums information from each homo-graph. Player2Vec leverages GCN & GAT to encode the intra- & inter-relation neighbor information. GAS [19] learns unique aggregators for different node types and updates the embeddings of each node types iteratively.

In this paper, CARE-GNN constructs multiple homo-graphs with only one node type like GEM and ASA. Among the above works, only two works [25, 41] have noticed the camouflage behaviors of fraudsters. While [41] only crafts new but inflexible features, and [25] suffers from unsupervised similarity measures and fixed filtering thresholds. CARE-GNN remedies those shortcomings by filtering neighbors based on label-aware similarity measures with adaptive filtering thresholds.

6 CONCLUSION

This paper investigates the camouflage behavior of fraudsters and their negative influence on GNN-based fraud detectors. To enhance the GNN-based fraud detectors against the feature camouflage and relation camouflage of fraudsters, we propose a label-aware similarity measure and a similarity-aware neighbor selector using reinforcement learning. Along with two neural modules, we further propose a relation-aware aggregator to maximize the computational utility. Experiment results on real-world fraud datasets present evidence of fraudster camouflage and demonstrate the effectiveness and efficiency of proposed enhancement modules, especially the reinforcement learning module.

ACKNOWLEDGMENTS

This work is supported by NSF under grants III-1526499, III-1763325, III-1909323, and CNS-1930941. For any correspondence, please refer to Hao Peng.

REFERENCES

[1] L. Akoglu, H. Tong, and D. Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* (2015).

[2] A. Breuer, R. Eilat, and U. Weinsberg. 2020. Friend or Faux: Graph-Based Early Detection of Fake Accounts on Social Networks. In *WWW*.

[3] D. Chen, Y. Lin, Wei Li, Peng Li, J. Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI*.

[4] H. Chen, L. Wang, S. Wang, D. Luo, W. Huang, and Z. Li. 2019. Label Aware Graph Convolutional Network–Not All Edges Deserve Your Attention. *arXiv preprint arXiv:1907.04707* (2019).

[5] J. Chen, T. Ma, and C. Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*.

[6] Y. Chen, L. Wu, and M. J. Zaki. 2020. Deep Iterative and Adaptive Learning for Graph Neural Networks. *AAAI Workshops* (2020).

[7] S. Dhawan, S.C.R. Gangireddy, S. Kumar, and T. Chakraborty. 2019. Spotting Collusive Behaviour of Online Fraud Groups in Customer Reviews. In *IJCAI*.

[8] Y. Dou, G. Ma, P. S. Yu, and S. Xie. 2020. Robust Spammer Detection by Nash Reinforcement Learning. In *KDD*.

[9] L. Franceschi, M. Niepert, M. Pontil, and X. He. 2019. Learning discrete structures for graph neural networks. In *ICML*.

[10] S. Ge, G. Ma, S. Xie, and P. S. Yu. 2018. Securing behavior-based opinion spam detection. In *IEEE Big Data*.

[11] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[12] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

[13] Y. Hou, J. Zhang, J. Cheng, K. Ma, R. T. B. Ma, H. Chen, and M. Yang. 2020. Measuring and Improving the Use of Graph Information in Graph Neural Networks. In *ICLR*.

[14] M. Jiang, P. Cui, and C. Faloutsos. 2016. Suspicious behavior detection: Current trends and future directions. *IEEE Intelligent Systems* (2016).

[15] P. Kaghazgaran, M. Alfifi, and J. Caverlee. 2019. Wide-Ranging Review Manipulation Attacks: Model, Empirical Study, and Countermeasures. In *CIKM*.

[16] P. Kaghazgaran, J. Caverlee, and A. Squicciarini. 2018. Combating crowdsourced review manipulators: A neighborhood-based approach. In *WSDM*.

[17] T.N. Kipf and M. Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[18] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and VS Subrahmanian. 2018. Rev2: Fraudulent user prediction in rating platforms. In *WSDM*.

[19] A. Li, Z. Qin, R. Liu, Y. Yang, and D. Li. 2019. Spam Review Detection with Graph Convolutional Networks. In *CIKM*.

[20] R. Li, S. Wang, F. Zhu, and J. Huang. 2018. Adaptive graph convolutional neural networks. In *AAAI*.

[21] X. Li, S. Liu, Z. Li, X. Han, C. Shi, B. Hooi, H. Huang, and X. Cheng. 2020. FlowScope: Spotting Money Laundering Based on Graphs. In *AAAI*.

[22] X. Liu, J. Wu, and Z. Zhou. 2008. Exploratory undersampling for class-imbalance learning. *IEEE TSMC* (2008).

[23] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi. 2019. Geniepath: Graph neural networks with adaptive receptive paths. In *AAAI*.

[24] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song. 2018. Heterogeneous Graph Neural Networks for Malicious Account Detection. In *CIKM*.

[25] Z. Liu, Y. Dou, P. S. Yu, Y. Deng, and H. Peng. 2020. Alleviating the Inconsistency Problem of Applying Graph Neural Network to Fraud Detection. *SIGIR*.

[26] J. McAuley and J. Leskovec. 2013. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *WWW*.

[27] A. Mukherjee, V. Venkataraman, B. Liu, and N. S. Glance. 2013. What Yelp Fake Review Filter Might Be Doing?. In *ICWSM*.

[28] H. Nilforoshan and N. Shah. 2019. SilceNDice: Mining Suspicious Multi-attribute Entity Groups with Multi-view Graphs. In *DSAA*.

[29] S. Rayana and L. Akoglu. 2015. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In *KDD*.

[30] Y. Sahin, S. Bulkan, and E. Duman. 2013. A cost-sensitive decision tree approach for fraud detection. *Expert Systems with Applications* (2013).

[31] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.

[32] L. Sun, B. Cao, J. Wang, W. Srisa-an, P. Yu, A. D. Leow, and S. Checkoway. 2020. KOLLECTOR: Detecting Fraudulent Activities on Mobile Devices Using Deep Learning. *IEEE TMC* (2020).

[33] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, and B. Li. 2018. Adversarial Attack and Defense on Graph Data: A Survey. *arXiv preprint arXiv:1812.10528* (2018).

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2017. Graph attention networks. In *ICLR*.

[35] V. Verma, M. Qu, A. Lamb, Y. Bengio, J. Kannala, and J. Tang. 2019. GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning. *arXiv preprint arXiv:1909.11715* (2019).

[36] J. Vermorel and M. Mohri. 2005. Multi-armed bandit algorithms and empirical evaluation. In *ECML*.

[37] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi. 2019. A Semi-supervised Graph Attentive Network for Fraud Detection. In *ICDM*.

[38] H. Wang, C. Zhou, J. Wu, W. Dang, X. Zhu, and J. Wang. 2018. Deep structure learning for fraud detection. In *ICDM*.

[39] J. Wang, R. Wen, C. Wu, Y. Huang, and J. Xiong. 2019. FdGars: Fraudster Detection via Graph Convolutional Networks in Online App Review System. In *WWW Workshops*.

[40] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *KDD Workshops* (2019).

[41] R. Wen, J. Wang, C. Wu, and J. Xiong. 2020. ASA: Adversary Situation Awareness via Heterogeneous Graph Convolutional Networks. In *WWW Workshops*.

[42] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. 2020. A comprehensive survey on graph neural networks. *IEEE TNNLS* (2020).

[43] C. Yang, H. Wang, L. Sun, and B. Li. 2020. Secure Network Release with Link Privacy. *arXiv preprint arXiv:2005.00455* (2020).

[44] X. Yang, Y. Lyu, T. Tian, Y. Liu, Y. Liu, and X. Zhang. 2020. Rumor Detection on Social Media with Graph Structured Adversarial Learning. In *IJCAI*.

[45] S F Yilmaz and S S Kozat. 2020. Unsupervised Anomaly Detection via Deep Metric Learning with End-to-End Optimization. *arXiv preprint arXiv:2005.05865* (2020).

[46] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. 2020. Graphsaint: Graph sampling based inductive learning method. *ICLR* (2020).

[47] S. Zhang, H. Yin, T. Chen, Q. V. N. Hung, Z. Huang, and L. Cui. 2020. GCN-Based User Representation Learning for Unifying Robust Recommendation and Fraudster Detection. In *SIGIR*.

[48] Y. Zhang, Y. Fan, Y. Ye, L. Zhao, and C. Shi. 2019. Key Player Identification in Underground Forums over Attributed Heterogeneous Information Network Embedding Framework. In *CIKM*.

[49] H. Zheng, M. Xue, H. Lu, S. Hao, H. Zhu, X. Liang, and K. Ross. 2018. Smoke screener or straight shooter: Detecting elite sybil attacks in user-review social networks. *NDSS* (2018).

[50] Q. Zhong, Y. Liu, X. Ao, B. Hu, J. Feng, J. Tang, and Q. He. 2020. Financial Defaulter Detection on Online Credit Payment via Multi-View Attributed Heterogeneous Information Network. In *WWW*.

[51] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *NeurIPS*.