Fight Fire with Fire: Towards Robust Graph Neural Networks on Dynamic Graphs via Actively Defense

Haoyang LI
HKUST
hlicg@connect.ust.hk

Shimin DI*
HKUST
sdiaa@connect.ust.hk

Calvin Hong Yi LI
The Cigna Group
calvin.li@evernorth.com

Lei CHEN
HKUST(GZ)&HKUST
leichen@cse.ust.hk

Xiaofang ZHOU
HKUST
zxf@cse.ust.hk

ABSTRACT

Graph neural networks (GNNs) have achieved great success on various graph tasks. However, recent studies have revealed that GNNs are vulnerable to injective attacks. Due to the openness of platforms, attackers can inject malicious nodes with carefully designed edges and node features, making GNNs misclassify the labels of target nodes. To resist such adversarial attacks, recent researchers propose GNN defenders. They assume that the attack patterns have been known, e.g., attackers tend to add edges between dissimilar nodes. Then, they remove edges between dissimilar nodes from attacked graphs, aiming to alleviate the negative impact of adversarial attacks. Nevertheless, on dynamic graphs, attackers can change their attack strategies at different times, making existing passive GNN defenders that are passively designed for specific attack patterns fail to resist attacks. In this paper, we propose a novel active GNN defender for dynamic graphs, namely ADGNN, which actively injects guardian nodes to protect target nodes from effective attacks. Specifically, we first formulate an active defense objective to design guardian node behaviors. This objective targets to disrupt the prediction of attackers and protect easily attacked nodes, thereby preventing attackers from generating effective attacks. Then, we propose a gradient-based algorithm with two acceleration techniques to optimize this objective. Extensive experiments on four real-world graph datasets demonstrate the effectiveness of our proposed defender and its capacity to enhance existing GNN defenders.

PVLDB Reference Format:
Haoyang LI, Shimin DI, Calvin Hong Yi LI, Lei CHEN, and Xiaofang ZHOU. Fight Fire with Fire: Towards Robust Graph Neural Networks on Dynamic Graphs via Actively Defense. PVLDB, 17(8): 2050 - 2063, 2024.
doi:10.14778/3659437.3659457

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Refrainlhy/ADGNN.

1 INTRODUCTION

With the increasing prevalence of graph-structured data in various domains, such as social networks and e-commerce platforms, ensuring the security and integrity of graph data has become a paramount concern for the database community. Adversarial attacks, aimed at compromising the integrity and availability of graph data, pose significant challenges to the database community. Due to the openness of current web platforms, attackers can register multiple users, manipulate node features, and create deceptive connections, thereby undermining the integrity of the graph data. Consequently, traditional defense mechanisms employed in database systems, such as access control [2, 4], encryption [64, 71], and differential privacy [10–12], fall short in protecting graph data within open-world applications from determined attackers. The challenge posed by adversarial attacks on graph data is particularly significant for graph neural networks (GNNs) [7, 8, 47, 49, 50, 61, 73, 91], which are specifically designed for graph-structured data and have achieved great success on various graph tasks (e.g., node classification [82, 89], recommendation [48, 86], knowledge graph [9], community search [20, 29], and graph isomorphism [14, 80]). GNNs propose to use a message-passing manner to learn low-dimensional representations by aggregating the information from neighbors.

Despite the great success, recent studies have shown that GNNs are vulnerable to *injective attacks* [19, 43, 95, 96], i.e., attackers can inject nodes into the graph to degrade the performance of GNNs (e.g., decreasing node classification accuracy). For example, as shown in Figure 1 (a), given a target node v_t that attackers are interested in, attackers can affect the GNN at platforms to misclassify the label of v_t by injecting nodes. More specifically, due to the openness of web platforms, attackers can register several users (red nodes v_1 and v_2) on the platforms, where we call these users registered by attackers as *malicious users*. Then, attackers can crawl users and their edges from open web platforms and train a surrogate GNN model to mimic the platform GNNs [31, 42,

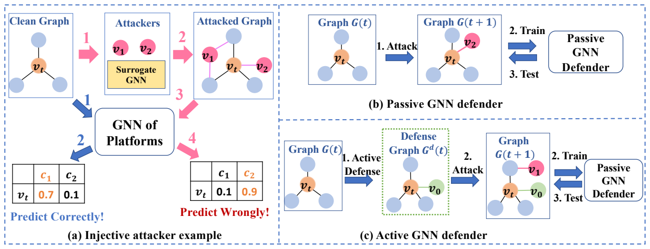


Figure 1: (a) Injective attack example. The GNN of platforms predicts the node v_t as class c_1 correctly based on the clean graph. After the attacker injects malicious nodes (v_1 and v_2) to affect the target node v_t, GNN predicts the label of v_t wrongly. (b) Passive GNN defender. Red nodes v_1 is injected by attackers. (c) Active GNN defender. Platform injects green node v_0 as the guardian node before malicious node v_1.

*Corresponding Author

Proceedings of the VLDB Endowment, Vol. 17, No. 8 ISSN 2150-8097.
doi:10.14778/3659437.3659457

63, 95]. Next, they use the trained surrogate GNN to design the edges and node features of malicious nodes v_1 and v_2 to affect the node representations of v_t. As a result, the platform's GNNs may misclassify the labels of target node v_t. For instance, a spammer can register a group of accounts with carefully crafted profiles and interconnected relationships. By utilizing these accounts to subscribe to its own account, the representation of this account can be affected by these newly registered accounts. Consequently, the GNN at platforms can misclassify this spammer account as a normal account.

To resist adversarial attackers, recent researchers [31, 42, 63, 79] propose to recover the graph quality by removing the adversarial edges from the attacked graph. As shown in Figure 1 (b), existing strategies restore the graph integrity and alleviate the negative impact of attackers on a graph after this graph has been attacked, i.e., they are *passive defense approaches*. Specifically, they assume that they know the attack patterns of existing attackers. For example, they believe that existing attackers tend to add edges between dissimilar nodes or increase the rank of the adjacency matrix [15, 31, 42, 44, 79]. Then, they propose to restore the clean graph from the attacked graph by removing edges between dissimilar nodes or decreasing the adjacency matrix rank. However, such passive defense strategies may fail to resist attacks on dynamic graphs, which is more practical in real-world graph scenarios. On the dynamic graphs, attackers can use different attack strategies to design the node behaviors at different times. As a result, the platform may not be able to identify the exact attack pattern used by attackers at each timestamp [31, 55, 63], and thus these passive defense strategies may have the risk of potential failures in restoring the graph quality. Consequently, GNNs trained on the processed graph data may still misclassify node labels.

To enhance the capability of GNNs to resist attacks, we propose a novel active GNN defender to actively defend attackers on dynamic graphs, which targets to actively degrade the effectiveness of GNN attackers. Such a way can help platforms avoid effective attack behaviors that will affect GNNs to misclassify the node labels of target nodes. As shown in Figure 1 (c), different from existing passive GNN defenders, the platform can actively inject nodes (e.g., v_0) with carefully designed node features and edges into graphs $G(t)$ to avoid effective attacks during time t and $t+1$. Here, we call these nodes injected by platforms actively to avoid effective attacks as *guardian nodes*. As mentioned before, attackers commonly train a surrogate GNN on the graph and use the trained GNN to guide the design of malicious node behaviors to attack target nodes. Thus, these guardian nodes injected by platforms can be used to degrade the accuracy of surrogate GNNs and protect the more vulnerable target nodes, thereby preventing the malicious nodes from changing the labels of target nodes. First, these guardian nodes can be utilized to change the node representations of training nodes. As a result, the surrogate GNN of attackers trained on the protected graph may not predict the real label distribution of target nodes. Without accurate predictions, attackers cannot generate suitable malicious node behaviors to attack target nodes. Second, the target nodes have different vulnerabilities to attacks. Thus, we can place the guardian node around easily attacked target nodes to disrupt the malicious node information, thereby preventing the attackers from changing the labels of target nodes. With these two approaches, we can enable the platform GNNs to classify nodes correctly at time $t+1$. To achieve this framework, we need to address the following technique challenges.

- So far, no metrics are to measure the impact of each training node on the predictions of target nodes, and the vulnerability of target nodes to attacks. Without these metrics, it is challenging to design effective behaviors for guardian nodes that can effectively prevent attackers from generating successful attacks.
- Furthermore, no objective has been proposed to guide the generation of guardian node behaviors. Since the platform can only inject guardian nodes with a limited budget (i.e., limited node number), the objective of active defense is crucial in designing guardian node behaviors to prevent potential effective attacks.
- Last but not least, the search space for potential behaviors of guardian nodes is exponentially large, making it infeasible to find the optimal solution through exhaustive search.

To address the above technique challenges, we propose an active GNN defender to actively resist the attacks on dynamic graphs, namely ADGNN. First, we theoretically analyze the importance of each training node on the label predictions of target nodes, as well as the vulnerability of target nodes. Secondly, based on the analysis, we propose an active defense objective that targets to change the label distribution of important training nodes and to protect the easily attacked target nodes. By optimizing this objective, the generated guardian node behaviors are capable of impeding attackers from generating effective attacks, thus enabling the platform GNN to classify nodes correctly. Thirdly, we observe that the search spaces of potential edges between guardian nodes and all other nodes, as well as candidate features for guardian nodes, are exponential. This indicates that identifying optimal protective behaviors through exhaustive search is impractical due to exponential combinations. Therefore, we propose a scalable gradient-based algorithm with two acceleration techniques to optimize the active defense objective efficiently. Specifically, we first identify the top-k influential and diverse training nodes to replace all training nodes. Then, we only design the guardian node behaviors by disrupting the labels of these top-k training nodes instead of all training nodes. By focusing our efforts on modifying the labels of just these top-k nodes, we diminish the computational burden on the defense objective. Secondly, rather than calculating the gradient across all possible candidate edges, we propose to sample the most significant edge candidates between the guardian nodes and all nodes. This sampling approach decreases the gradient computation time on the candidate edges. Based on the top-k training nodes and candidate edge sampling, we can efficiently design the node behaviors for dynamic graphs without overwhelming the computational resources (i.e., GPU memory). Our contributions are summarized as follows:

- We propose an effective and efficient active GNN defender for dynamic graphs, namely ADGNN. We inject guardian nodes in advance to degrade attackers and protect easily attacked nodes. To the best of our knowledge, this is the first work to propose active defense against GNN attackers.
- We theoretically analyze the impact of each training node on the prediction of target nodes, as well as the vulnerability of target nodes. Based on the analysis, we formulate an active defense

objective that can both impede the training process of attackers and protect the easily attacked target nodes.

- We propose an effective and efficient gradient-based greedy algorithm to optimize the active defense objective. Additionally, we introduce two techniques, i.e., top-k influential training node selection and crucial edge sampling, to accelerate it.
- Experiments on four real-world graph datasets demonstrate the superior performance and high scalability of our active GNN defender ADGNN. Besides, ADGNN can help existing passive GNN defenders to achieve better performance against the attackers.

2 PRELIMINARY AND RELATED WORK

In various real-world applications, e.g., social networks, the node feature and graph topology of a dynamic graph usually evolve with different graph events. Formally, we denote a dynamic graph at time t as $G(t) = (V(t), \mathbf{A}(t), \mathbf{X}(t))$, where $V(t), \mathbf{A}(t) \in \{0,1\}^{|V(t)|\times|V(t)|}$, and $\mathbf{X}(t) \in \mathbb{R}^{|V(t)|\times d_x}$ denote nodes, adjacency matrix, and node features at time t, respectively. The training nodes $V^{tr}(t) \subseteq V(t)$ with labels $\mathbf{Y}(t) \in \{0,1\}^{|V^{tr}(t)|\times|\mathcal{Y}|}$ denote the labeled training nodes at time t. Also, $N_v(t) = \{u : \mathbf{A}(t)[u][v] = 1\}$ denotes the neighbors of v at time t. In general, GNNs for the node classification task [70] are to learn a classifier $f(t) : V(t) \to \mathcal{Y}$ to map nodes to label space $\mathcal{Y}$. We summarize the important notations in Table 1.

2.1 Dynamic Graph Neural Networks

2.1.1 Node Representation Learning. In general, DGNNs [21, 36, 38, 45, 46, 57, 67, 72, 84, 92] consist of two steps as follows.

Message-passing Aggregation. Given the dynamic graph $G(t) = (V(t), \mathbf{A}(t), \mathbf{X}(t))$, a DGNN $\mathcal{M}_\theta$ parameterized by $\theta \in \Theta$ learns the node representations $\mathbf{h}_v(t) \in \mathbb{R}^{d_h}$ for each node v by aggregating the latest state information from their neighbors $N_v(t)$. The l-th hidden representation of each node v can be learned by aggregating the information from their neighbors $N_v(t)$ as:

$$\mathbf{h}_v^{(l)}(t) = \sigma\Big(\sum_{u\in N_v(t)} \mathbf{A}_n(t)[v][u] \cdot \mathbf{h}_u^{(l-1)}(t)\, \mathbf{W}^{(l-1)}\Big), \tag{1}$$

where $\mathbf{A}_n(t)$ is the normalized adjacency matrix, such as $\mathbf{A}_n(t) = (\mathbf{D}(t))^{-1}\mathbf{A}(t)$ [25, 27], where $\mathbf{D}(t)$ is the degree matrix. σ is a non-linear function (e.g., `Sigmoid` or `ReLU`) and $\mathbf{W}^{(l-1)} \in \theta$ is the transform parameters of the *(l-1)*-th layer. In particular, $\mathbf{h}_u^{(0)}(t) = \mathbf{s}_u(t)$ is the state vector of each node $u \in V(t)$.

Node State Updating. DGNNs maintain a state vector $\mathbf{s}_v(t) \in \mathbb{R}^{d_s}$ for each node v, reflecting the status of v at time t. Specifically, several works [37, 56, 59, 84], such as DySAT [59] and RoLAND [84], directly take the node feature $\mathbf{x}_v(t)$ as the state vector $\mathbf{s}_v(t)$. On the other hand, several works [33, 46, 53, 57, 67, 72], such as TGCN [57] and DyREP [67], maintain state vectors by update functions, such as **MEAN** [72], i.e., they update $\mathbf{s}_v(t) = (\mathbf{s}_v(t-1) + \mathbf{s}_u(t-1))/2$ under a new edge $e(v,u)$ between node v and node u at time t.

2.1.2 Node Classification Task. In node classification task, each node v can be categorized into a label $y_v \in \mathcal{Y}$. The target of GNN $\mathcal{M}_\theta$ at each time t is to predict the label score $\mathbf{Z}(t) \in [0,1]^{|V(t)|\times|\mathcal{Y}|}$ for nodes. Specifically, given a graph $G(t)$, training nodes $V^{tr}(t)$ with labels $\mathbf{Y}(t)$, the optimized GNN parameter $\theta^*(t)$ can be obtained by minimizing the following loss:

$$\min_{\theta\in\Theta} L_{gnn}(G(t), V^{tr}(t), \mathcal{M}_\theta) = \frac{\sum_{u\in V^{tr}(t)} l_{nc}(u, G(t), \mathcal{M}_\theta)}{|V^{tr}(t)|} \tag{2}$$

where the loss $l_{nc}(u, G(t), \mathcal{M}_\theta) = -\log \mathbf{Z}(t)[u][y_u]$ is the cross-entropy loss and y_u is the ground label of node u.

Table 1: Important notations.

Notation	Description
t, t^-, T	Time index, the latest time before t, final time
v, u, v_i, v_j	The generic node index
$G(t)$	Graph at time t, $G(t) = (V(t), \mathbf{A}(t), \mathbf{X}(t), \mathbf{Y}(t))$
$N_v(t), N_v^l(t)$	v's 1-hop and l-hop neighbors
$G^p(t), G^a(t)$	Protected graph and attacked graph at t
$V^{tr}(t)$	The training nodes with labels $\mathbf{Y}(t)$ at t
$V^{tar}(t)$	The target nodes at time t
$V^p(t)$	Guardian nodes for protection at t
$V^m(t)$	Malicious nodes controlled by attackers at t
$y_v(t)$	The ground label of training node v
$y_u^*(t)$	The predicted label of unlabeled node u
$M_{\theta^*(t)}$	The optimized GNN model at time t
$\mathbf{H}(t) \in \mathbb{R}^{|V(t)|\times d_z}$	Node representation matrix of all nodes at time t
$\mathbf{h}_v(t) \in \mathbb{R}^{d_z}$	Node representation vector of node v at time t
$\mathbf{s}_v(t) \in \mathbb{R}^{d_s}$	State vector of node v at time t
$\mathbf{Z}(t) \in \mathbb{R}^{|V(t)|\times|\mathcal{Y}|}$	Predicted label score of nodes at time t
k_a, k_p	Budget (i.e., number) of malicious and guardian nodes
k_{tr}	The number of selected training nodes
s	Sampling size for candidate edges
$\hat{\mathcal{I}}(u, V^{tar}(t))$	The normalized influence score of training node u
$\hat{\mathcal{E}}(v, t)$	The normalized easy score of target node v
$H_\theta(t)$	Hessian matrix at time t
$AO(\cdot)$	Active defense objective
$RS(\cdot)$	Representative score function

2.2 GNN Attackers on Dynamic Graphs

GNN attackers aim to degrade the performance of the target GNNs, i.e., decreasing the node classification accuracy. Due to the openness of real-world applications, GNN attackers can inject a set of malicious nodes into the graph $G(t)$, and design their features and edges. Then, attackers can affect the node representations of target nodes. Consequently, GNNs may misclassify the labels of target nodes at the next time $t+1$. For clarification, we call the nodes injected by attackers as *malicious nodes*. Here we formally formulate the GNNs attack problem on dynamic graphs at each time t.

Definition 2.1 (Target of GNN attackers.). Given the malicious nodes budget k_a, a GNN model $\mathcal{M}_\theta$, graph $G(t) = (V(t), \mathbf{A}(t), \mathbf{X}(t))$, training nodes $V^{tr}(t)$ with labels $\mathbf{Y}(t)$, and target nodes $V^{tar}(t)$, GNN attackers can inject malicious nodes $V^m(t)$ with size constraint $|V^m(t)| \le k_a$ into graph $G(t)$. The target of GNN attackers is to design node features and connected edges of $V^m(t)$ to generate the attacked graph $G^a(t) = (V^a(t), \mathbf{A}^a(t), \mathbf{X}^a(t))$, which can minimize the attack loss $L_{atk}(\cdot)$ as follows.

$$\min_{G^a(t)} L_{atk}(G^a(t), \mathcal{M}_{\theta^*(t)}), \tag{3}$$
$$s.t. \begin{cases} \theta^*(t) = \arg\min_{\theta\in\Theta} L_{gnn}(G^a(t), V^{tr}(t), \mathcal{M}_\theta) \\ |V^m(t)| \leq k_a \end{cases},$$

where $L_{gnn}(\cdot)$ is GNN training loss in Equation (2). $L_{atk}(\cdot)$ denotes the attack loss defined by attackers. One formulation of $L_{atk}(\cdot)$ is $L_{atk}(G^a(t), \mathcal{M}_{\theta^*(t)}) = -L_{gnn}(G^a(t), V^{tr}(t), \mathcal{M}_{\theta^*(t)})$ [95, 96], since GNN M_θ with a high training error on attacked graph $G^a(t)$ is more likely to generalize poorly on target nodes $V^{tar}(t)$.

In general, existing GNN attackers can be categorized into three types, i.e., white-box, gray-box, and black-box attackers. Specifically, white-box attackers [22, 79] assume full access to the graph (i.e., graph structure, node features), labeled training nodes, and the target GNN parameters, to design the node features and edges of injected malicious nodes. Different from white-box attackers, gray-box attackers [6, 95, 96] do not need the parameters of target GNNs, while they only require the graph structure, node features, and labeled training nodes. They train a surrogate GNN which simulates the target GNNs, and then generate attacks based on the surrogate model. However, white-box and gray-box attackers are impractical since it is scarcely realistic to obtain the parameters of target GNNs or a large number of node labels in most real-world applications. Recently, black-box attackers [5, 16, 42, 51, 52], are proposed to only access partial graph structure, node features, and limited node predictions from the target GNNs, to design malicious node behaviors. For example, due to the openness of website platforms, attackers can collect the user profiles (i.e., node features) and their connections through subscriptions or watching history (i.e., edges) from social media platforms (e.g., IMDB [28], rottentomatoes [58], Goodreads [24], and YouTube [85]) and e-commerce platforms (e.g., Amazon [1] and Yelp [83]). Also, platforms may assign tags to users, and attackers can crawl these tags as the user labels predicted by platform GNNs [39]. Thus, attackers can use these predictions to infer the behaviors of platform GNNs.

Following [22, 79], we investigate the robustness of GNNs on dynamic graphs under the white-box attackers. This is because white-box attackers assume access to all information about GNNs and graphs from platforms. In other words, white-box attackers are assumed to know the exact behaviors of platform GNNs, therefore most effectively compromising the performance of the platform's GNN. Therefore, from the defender perspective, it is better to directly propose GNN defenders that resist white-box attackers. If the proposed defender can resist white-box attackers effectively, we can expect that it can resist gray-box and black-box attackers effectively as well. Also, due to information leaks, attackers can obtain user data from various platforms. For example, even if a Facebook user sets its profile and subscriptions to private, attackers have the opportunity to collect its comments or moments from platforms like Twitter to construct this user's profile and subscriptions. Therefore, even though black-box attackers are more realistic, we cannot propose a GNN defender that only resists black-box attackers.

2.3 GNN Defenders on Dynamic Graphs

As introduced in Section 2.2, GNN attackers inject k_a malicious nodes into the graph $G(t)$ between the time t and $t+1$. Correspondingly, GNN defenders are proposed, which target to enable the platform GNN to classify the labels of target nodes correctly under the attacked graph. In general, the core procedure of existing GNN defenders is to purify the structure of attacked graphs, which can alleviate the negative impact of injected nodes on the information aggregation process. Thus, GNNs can capture the underlying patterns of each node on the purified graph and predict node labels correctly. Without loss of generality, we define GNN defenders on dynamic graphs at each time $t+1$ as follows.

Definition 2.2 (Target of GNN defenders). Given a GNN model $\mathcal{M}_\theta$, the graph $G(t+1) = (V(t+1), \mathbf{A}(t+1), \mathbf{X}(t+1))$ attacked by attackers between time t and $t+1$, and target nodes $V^{tar}(t)$, the target of GNN defenders is to find a purified graph $\tilde{G}^*(t+1) = (V(t+1), \tilde{\mathbf{A}}^*(t+1), \tilde{\mathbf{X}}^*(t+1))$, which minimizes $L_{gnn}(\cdot)$:

$$\theta^*(t+1) = \arg\min_{\theta\in\Theta} \mathcal{L}_{gnn}(\tilde{G}^*(t+1), V^{tr}(t+1), \mathcal{M}_\theta) \tag{4}$$
$$s.t.\ \tilde{G}^*(t+1) = \arg\min_{\tilde{G}(t+1)} \mathcal{L}_{pfy}(\mathcal{M}_{\theta^*(t+1)}, \tilde{G}(t+1), V^{tar}(t))$$

where $\mathcal{L}_{pfy}(\cdot)$ measures the loss of the purification algorithm, which targets to generate the purified graph structure $\tilde{\mathbf{A}}^*(t+1)$ and node features $\tilde{\mathbf{X}}^*(t+1)$ to make the platform GNN get a lower loss on training nodes $V^{tr}(t+1)$. In such a way, they believe that the platform GNN can predict the target nodes $V^{tar}(t)$ correctly.

Generally, existing GNN defenders can be classified into three categories. First, preprocessing-based defenders [77, 81] eliminate edges between dissimilar nodes from the attacked graph. Second, attention-based defenders [70, 90, 93] use the attention mechanism [69] to measure the relative weight of edges, and expect that they can assign lower weight for the adversarial edges. Third, graph learning-based defenders [30, 31, 88] reconstruct graph structure $\tilde{\mathbf{A}}(t+1)$ and node features $\tilde{\mathbf{X}}(t+1)$ by minimizing the GNN training loss on $\tilde{G}(t+1)$ and decreasing the rank of adjacent matrix. However, these existing GNN defenses are passive, as they are designed for specific attack patterns and cannot adapt to new attack strategies [55]. On the dynamic graphs, the attackers can employ various strategies to design the node behaviors at different times. As a result, the platform may not be able to identify the exact attack pattern used by attackers at each timestamp, leading to potential failures in protecting the GNN from misclassifying node labels.

In this paper, we propose an active GNN defender, which injects guardian nodes at time t in advance to prevent effective attacks during time t and $t+1$. Such a way can ensure GNNs to classify the target nodes correctly at time $t+1$.

2.4 Robustness in Machine Learning

Adversarial robustness in machine learning [60, 62] is concerned with the resilience of models against malicious inputs, known as adversarial examples, which are subtly modified to induce errors in the model's output. Specifically, in the computer vision field, researchers have observed that even minor perturbations to an image can drastically affect the performance of machine learning models [35, 65]. To counteract such vulnerabilities, a variety of defensive strategies have been proposed, including adversarial training [26, 66, 75], contrastive learning [17, 23, 40], and attack detection [13]. Inspired by computer vision, researchers have shown that graph-based models, such as GNNs, can also be compromised by slight alterations to the graph structure [95, 96]. Due to the intricate and non-uniform nature of graph data, researchers [30, 31, 70, 77, 81, 88, 90, 93] have advocated for the purification of graph structures before learning node representations. As we introduced in Section 2.3, such methods are categorized as passive defenses for GNNs. In this paper, the proposed ADGNN model addresses the unique challenges posed by dynamic graphs with an active defense strategy.

3 FRAMEWORK OVERVIEW

We introduce the basic procedure of active GNN defender, which consists of three stages and is summarized in Algorithm 1.

Stage 1: GNN Training and Evaluation. As shown in Algorithm 1 lines 2-3, given a graph $G(t)$ with training nodes $V^{tr}(t)$ and labels $\mathbf{Y}(t)$, platforms will train a GNN model M_θ by minimizing the loss in Equation (2), where $\theta \in \Theta$. Then, the platforms can use the optimized GNN $M_{\theta^*(t)}$ to predict the labels of the target nodes and facilitate the downstream tasks. In particular, the GNN of platforms can be raw GNNs (e.g., GraphSAGE [25] and RoLAND [84]) and existing passive GNN defenders (e.g., ProGNN [31] and GNAT [42]).

Stage 2: Active Defense Graph Generation. As shown in Algorithm 1 line 4, after the platform optimizes the GNN model on graph $G(t)$, our active GNN defender (See Section 4) can actively inject k_p nodes to protect the target nodes $V^{tar}(t)$ from being successfully attacked during time t and $t+1$. Specifically, as shown in Figure 2, in Step 1, we first select top-k training nodes from all training nodes (Section 4.3.1). Then, in Step 2, we compute the sampling probability of candidate edges between all nodes and guardian nodes and then sample important edges (Section 4.3.2). In Step 3, we compute the active objective in Equation (8) and optimize sampled edge weights and features of guardian nodes. We will repeat Step 2 to sample candidate edges and optimize their weights and node features in Step 3. Finally, in Step 4, we obtain the active defense graph $G^d(t)$ by generating edges of guardian nodes and node features based on edge weights and feature weights, respectively. We introduce the details of active defense graph generation in Section 4.

Stage 3: Adversarial Attacks and Natural Evolving. As shown in Algorithm 1 line 5-7, during time t and $t+1$, new graph events $\alpha(t, t+1)$ arrive. Specifically, $\alpha(t, t+1)$ includes various events, such as edge addition/deletion (e.g., subscription), node addition/deletion (e.g., user registration), and node feature modifications (e.g., user profile modifications). Besides, attackers can inject k_a malicious nodes into platforms to attack the target nodes $V^{tar}(t)$, which will affect the platforms to misclassify the labels of the target nodes $V^{tar}(t)$ at time $t+1$. In particular, we assume the attackers are the most effective white-box attackers to evaluate the active defender, i.e., the attackers can access all information from platforms, such as the platform GNN M_θ, graph $G(t)$ with training nodes $V^{tr}(t)$ and labels $\mathbf{Y}(t)$, target nodes $V^{tar}(t)$, and new events $\alpha(t, t+1)$.

Algorithm 1: Active GNN defender framework

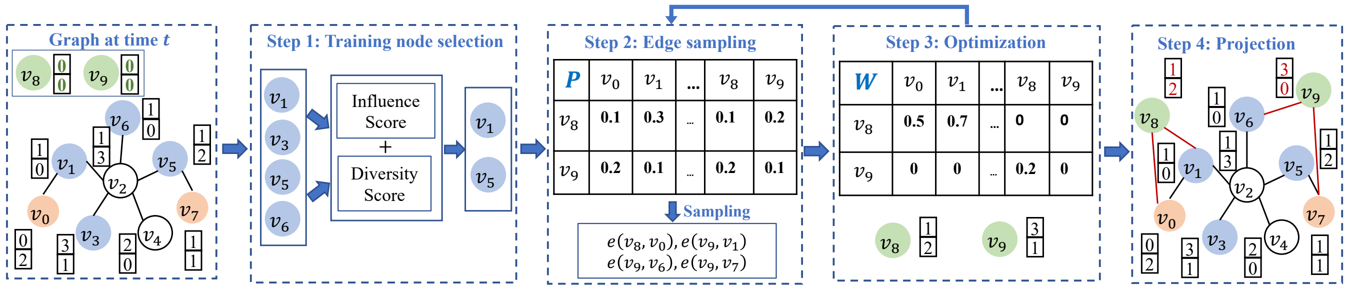
```
Input: The initial graph $G(1)$ with new graph events $\{\alpha(t,t+1)\}_{t=1}^{T-1}$, the GNN model $\mathcal{M}_\theta$, the target nodes $\{V^{tar}(t)\}$, the protection budget $k_p$
Output: The optimized GNN and active defense graph $\{\mathcal{M}_{\theta^*(t)}, G^d(t)\}_{t=1}^{T}$ and $\overline{Acc}$
1 for $t = 1$ to $T$ do
2     $\theta^*(t) = \arg\min_{\theta\in\Theta} L_{gnn}(G(t), V^{tr}(t), \mathcal{M}_\theta)$
3     $Acc(t) = \textsc{Evaluate}(\mathcal{M}_{\theta^*(t)}, V^{tar}(t))$
4     $G^d(t) = \textsc{ActiveDefender}(G(t), \mathcal{M}_{\theta^*}(t), V^{tar}(t), k_p)$
5     $G^e(t) = \textsc{NaturalEvolving}(G^d(t), \alpha(t, t+1))$
6     $G^a(t) = \textsc{Attacker}(G^e(t), \mathcal{M}_{\theta^*}(t), V^{tar}(t), k_a)$
7     $G(t+1) = G^a(t)$
8 $\overline{Acc} = \frac{1}{T-1}\sum_{t=2}^{T} Acc(t)$
9 Return $\{\mathcal{M}_{\theta^*(t)}, G^d(t)\}_{t=1}^{T}$ and $\overline{Acc}$
```

4 ACTIVE DEFENSE GRAPH GENERATION

We introduce the details of our proposed active defense graph generation in Stage 2 of Section 3. Specifically, we first analyze how to degrade the effectiveness of attackers and the vulnerability of each target node. Then, we formulate the active defense objective that can actively degrade attackers and protect target nodes. Due to the exponential search space for optimizing this objective, we propose a gradient-based greedy algorithm with the support of two acceleration algorithms to solve it, i.e., top-k training node selection and crucial edge sampling. As illustrated in Figure 2, we provide an example to demonstrate the generation of an active defense graph with our proposed techniques.

4.1 Theoretical Analysis

As illustrated in Equation (3) in Section 2.2, existing attackers design the malicious node behaviors by solving a bi-level optimization problem. Specifically, attackers first train a GNN on the training nodes to predict the labels of target nodes at the low-level optimization. Then, attackers use this pretrained GNN to guide the malicious node behavior generation at the high-level optimization. This process reveals that attackers' success relies on two key factors: the quality of the pretrained GNN at the low-level and the behavior design of malicious nodes to attack target nodes at the high-level optimization. Therefore, the actively guardian nodes injected by platforms can create interference at both levels of the attackers' strategy, thereby thwarting their ability to generate successful attacks that impact the labels of target nodes. First, these guardian nodes can be utilized to change the node representations of training nodes. As a result, the surrogate GNN of attackers trained on the protected graph may not predict the real label distribution of target nodes. Without accurate predictions, attackers cannot generate suitable malicious node behaviors to attack target nodes. Second, target nodes exhibit varying vulnerabilities to attacks. A target node whose representation is easily influenced is inherently more vulnerable and more likely to be incorrectly classified by the platform when under attack. Thus, we can place the guardian node around easily attacked target nodes to disrupt the malicious node



P	v_0	v_1	...	v_8	v_9
v_8	0.1	0.3	...	0.1	0.2
v_9	0.2	0.1	...	0.2	0.1

W	v_0	v_1	...	v_8	v_9
v_8	0.5	0.7	...	0	0
v_9	0	0	...	0.2	0

Figure 2: A running example of active defense graph generation at each time t. Two orange nodes (v_0 and v_7) denote target nodes for protection. Four black nodes (v_1, v_3, v_5, and v_6) denote training nodes. Two green nodes (v_8 and v_9) denote guardian nodes injected by platforms. Specifically, at each time t, we first select top-k (i.e., $k = 2$) training nodes from all training nodes (Section 4.3.1). Then, in Step 2, we compute the sampling probability of candidate edges between all nodes and the guardian nodes and then sample four edges (Section 4.3.2). In Step 3, we compute the active objective in Equation (8) and optimize the weight of sampled edges and features of guardian nodes. We will repeat Step 2 to sample candidate edges and optimize the weight of sampled edges and node features in Step 3. Finally, in Step 4, we obtain the active defense graph $G^d(t)$ by generating the connected edges of guardian nodes (red edges) and node features based on the edge weights and feature weights, respectively.

information, thereby preventing the attackers from changing the labels of target nodes.

4.1.1 *Attackers Misleading.*

We first theoretically analyze the impact of each training node on the predicted score of target nodes. Intuitively, if we can first identify the important training nodes that can more affect the predicted label score of target nodes, we can actively inject guardian nodes to perturb these important training nodes, i.e., changing their node labels. In such a way, the attackers trained on the protected graph may not predict the score of target nodes accurately. Consequently, attackers cannot generate effective malicious node behaviors to attack target nodes.

As introduced in Section 2.1, at each time t, the GNN model $\mathcal{M}_\theta$ obtains the optimized parameters $\theta^*(t)$ by minimizing the loss function on each training nodes $u \in V^{tr}(t)$ with weight $\frac{1}{|V^{tr}(t)|}$. Intuitively, if we change a bit weight of a training node u from $\frac{1}{|V^{tr}(t)|}$ to $\frac{1}{|V^{tr}(t)|} + \zeta$ where $\zeta \to 0$, we can get the new optimized GNN parameters $\theta^*_{\zeta,u}$ by minimizing the following loss:

$$\theta^*_{\zeta,u}(t) = \arg\min_{\theta\in\Theta} \frac{1}{|V^{tr}(t)|} \sum_{v\in V^{tr}(t)} l_{nc}(v,\theta) + \zeta l_{nc}(u,\theta) \quad (5)$$

We denote the label score of nodes predicted by GNN model $\mathcal{M}_{\theta^*_{\zeta,u}(t)}$ as $\mathbf{Z}_{\zeta,u}(t)$. Then, the predicted score difference between $\mathbf{Z}_{\zeta,u}(t)$ and $\mathbf{Z}(t)$ on target nodes $V^{tar}(t)$ is $\sum_{v\in V^{tar}(t)} \|\mathbf{Z}_{\zeta,u}(t)[v] - \mathbf{Z}(t)[v]\|$. Intuitively, if $\sum_{v\in V^{tar}(t)} \|\mathbf{Z}_{\zeta,u}(t)[v] - \mathbf{Z}(t)[v]\|$ is larger, the training node u is more influential on predicting the label distribution for target nodes. Therefore, we can measure the influence of each training node u on target nodes $V^{tar}(t)$ as $\mathcal{I}(u, V^{tar}(t)) = \sum_{v\in V^{tar}(t)} \|\mathbf{Z}_{\zeta,u}(t)[v] - \mathbf{Z}(t)[v]\|$. One straightforward way to compute $\mathcal{I}(u, V^{tar}(t))$ for each $u \in V^{tr}(t)$ is to retrain the GNN model on training nodes $V^{tr}(t)$ by slightly changing the weight of node u. However, it is too time-consuming. Therefore, inspired by the influence function [34], we measure the importance score of each training node $u \in V^{tr}(t)$ without retraining as Theorem 4.1.

THEOREM 4.1. *Given graph $G(t)$, GNN model M_θ parameterized by $\theta \in \Theta$, the GNN training loss function $l_{nc}(\cdot)$ in Equation* (2)*, training nodes $V^{tr}(t) \subseteq V(t)$, and the target nodes $V^{tar}(t) \subseteq V(t)$, the importance $\mathcal{I}(u, V^{tar}(t))$ of each training node $u \in V^{tr}(t)$ regarding the target nodes $V^{tar}(t)$ can be measured as :*

$$\mathcal{I}(u, V^{tar}(t)) = \sum_{v\in V^{tar}(t)} \|\nabla_\theta \mathbf{Z}(t)[v]^\top H_\theta^{-1} \nabla_\theta l_{nc}(u, \mathcal{M}_{\theta^*(t)})\|, \quad (6)$$

where $H_\theta = \frac{1}{|V^{tr}(t)|} \sum_{u_1\in V^{tr}(t)} \nabla^2_\theta l_{nc}(u_1, \mathcal{M}_{\theta^*}(t))$ is the Hessian matrix. $\theta^*(t)$ is the optimized GNN parameters on training nodes $V^{tr}(t)$ based on Equation (2)*.*

PROOF SKETCH. We first compute $\left.\frac{\partial \mathbf{Z}(t)[v]}{\partial\zeta}\right|_{\zeta=0}$ based on influence function [34] and then compute $\mathcal{I}(u, V^{tar}(t))$. Due to space limits, we put full proof in technique report [41] Appendix A.1. □

Particularly, we denote the normalized influence score of each training node u as $\hat{\mathcal{I}}(u, V^{tar}(t)) = \frac{\mathcal{I}(u,V^{tar}(t))}{\sum_{u_1\in V^{tr}(t)} \mathcal{I}(u_1,V^{tar}(t))}$. If a training node u has a larger influence score $\hat{\mathcal{I}}(u, V^{tar}(t))$ towards the prediction of target nodes, we can design guardian node behaviors to decrease the score on its ground truth label y_v, i.e., $\mathbf{Z}(t)[v][y_v]$. In such a way, the attackers trained on these perturbed training nodes cannot accurately predict the label score of target nodes, thereby preventing them from generating effective attacks.

4.1.2 *Target Node Protection.*

Secondly, we theoretically analyze the vulnerability of each target node to attacks. Intuitively, if the predicted label of target nodes is more easily changed after a set of attacks, this target node is more vulnerable to attacks. Therefore, if we can identify these easily attacked target nodes, we can place guardian nodes around them to prevent the information propagation from potential malicious nodes, thereby enabling them to be invariant to attackers. Specifically, based on Theorem 4.2, we identify two factors that mainly determine the vulnerability of each node, i.e., the neighbor number and the predicted score difference between the correct label and the wrong label.

Theorem 4.2. *Given a L-layer GNN model $M_{\theta^*(t)}$ in Section 2.1.1 optimized on graph $G(t)$, the prediction score of a target node v is denoted as $\mathbf{Z}(t)[v] \in [0,1]^{|\mathcal{Y}|}$. Let y_v denote the ground truth of node v, and let y'_v denote the wrong label with the largest score in $\mathbf{Z}(t)[v]$, i.e., $y'_v = \arg\max_{y \in \mathcal{Y} \setminus y_v} \mathbf{Z}(t)[v][y]$. Considering that attackers inject k malicious nodes $\mathcal{V}^m(t)$ to generate the attacked graph $G^a(t)$, and the predicted score for v on $G^a(t)$ is denoted as $\mathbf{Z}^a[v]$. Then, the predicted score difference $\mathbf{Z}^a(t)[v][y_v]$ and $\mathbf{Z}^a(t)[v][y''_v]$ under k malicious nodes can be bound as follows:*

$$\mathbf{Z}^a(t)[v][y_v] - \mathbf{Z}^a(t)[v][y''_v] \geq \frac{(N_v(t)|+1)(\mathbf{Z}(t)[v][y_v] - \mathbf{Z}(t)[v][y'_v])}{(|N_v(t)|+1+k)(k+1)^{L-1}} + c^{min}_{v,k}$$

where $y''_v = \arg\max_{y \in \mathcal{Y} \setminus y_v} \mathbf{Z}^a(t)[v][y]$ is the wrong label with the largest score in $\mathbf{Z}^a(t)[v]$. The constant $c^{min}_{v,k} = \min_{y \in \mathcal{Y} \setminus y_v} -|\mathbf{W}^\top[y]|$, where $\mathbf{W} = \prod_{\mathbf{W}^l \in \theta^*(t)} \mathbf{W}^l$.

Proof Sketch. We first compute worse-case value of $\mathbf{Z}^a[v][y_v] - \mathbf{Z}^a[v][y''_v]$ after injecting k malicious nodes. Then, we relax GNN by removing non-linear functions and get a lower bound. Due to space limits, we put full proof in technique report [41] Appendix A.2. □

If the predicted score on ground label y_v is larger than the predicted score on any other label $y \in \mathcal{Y} \setminus y_v$ after attacks, the node v will be classified correctly. Therefore, smaller $\mathbf{Z}^a(t)[v][y_v] - \mathbf{Z}^a(t)[v][y''_v]$ indicates that it is more likely to predict the node label of v incorrectly. According to Theorem 4.2, we can observe that $\mathbf{Z}^a(t)[v][y_v] - \mathbf{Z}^a(t)[v][y''_v]$ is bounded by two factors (1) the neighbor number $|\mathcal{N}_v(t)|$ of node v at time t and (2) the original predicted score difference $\mathbf{Z}(t)[v][y_v] - \mathbf{Z}(t)[v][y'_v]$ between the correct label y_v and the most incorrect label y'_v before attacks. In other words, a node v with more neighbors $N_v(t)$ and a larger difference score $\mathbf{Z}(t)[v][y_v] - \mathbf{Z}(t)[v][y'_v]$ is more difficult to be attacked during time t and $t+1$. Based on this analysis, we formally define the easy score of each target node $v \in V^{tar}(t)$ as follows:

$$\mathcal{E}(v,t) = \frac{-\log(c(\mathbf{Z}(t)[v][y^*_v] - \mathbf{Z}(t)[v][y'_v]) + b)}{\log(|N_v(t)|+1)} \tag{7}$$

where $y^*_v = \arg\max_{y \in \mathcal{Y}} \mathbf{Z}(t)[v][y]$ is the predicted label for node v and $y'_v = \arg\max_{y \in \mathcal{Y} \setminus y^*_v} \mathbf{Z}(t)[v][y]$ the label with largest predicted score except label y^*_v. Also, c is a hyperparameter to scale the score difference between y_v and y'_v and b is a constant. A larger $\mathcal{E}(v,t)$ indicates that the v is more easily to be attacked during time t and $t+1$. Particularly, we denote the normalized easy score of each node v as $\hat{\mathcal{E}}(v,t) = \frac{\mathcal{E}(v,t)}{\sum_{u \in V^{tar}(t)} \mathcal{E}(u,t)}$. If a target node v has a high easy score $\mathcal{E}(v,t)$, we can design guardian node behaviors to increase the score on its predicted label and its neighbor's number. Such a way increases the difficulty of attacking it.

4.2 Active Defense Objective

We first propose the active defense objective and propose a gradient-based greedy algorithm.

Definition 4.3 (Active Defense Objective). Given a graph $G(t) = (V(t), \mathbf{A}(t), \mathbf{X}(t))$ at time t, the GNN model $\mathcal{M}_\theta$, the training nodes $V^{tr}(t)$ with labels $\mathbf{Y}(t)$, the platform actively inject k_p guardian nodes $V^p(t)$ into graph $G(t)$ to generate active defense graph $G^d(t) = (V^d(t), \mathbf{A}^d(t), \mathbf{X}^d(t))$. Considering that each guardian node can connect at most τ edges with other nodes, the connected edges and node features of the guardian nodes are designed by maximizing the following active defense objective:

$$\begin{aligned} \max_{G^d(t)} \ & (1-\lambda) \sum_{u \in V^{tr}(t)} \hat{I}(u, V^{tar}(t)) \cdot \sigma(-\triangle \mathbf{z}'_u(t)[y_u]) \\ & + \lambda \sum_{v \in V^{tar}(t)} \hat{\mathcal{E}}(v,t) \cdot (\sigma(\triangle \mathbf{z}'_v(t)[y^*_v]) + \log(\triangle |N'_v(t)| + 1)) \\ & s.t. |V^p| \leq k_p; \|\mathbf{A}^d[u]\|_0 \leq \tau, \forall u \in V^p(t) \end{aligned} \tag{8}$$

where $\hat{I}(u, V^{tar}(t))$ and $\hat{\mathcal{E}}(v,t)$ is the normalized influence of each training node u in Section 4.1.1 and the easy score of each target node v in Section 4.1.2, respectively. y_u is the ground label of each training node $u \in V^{tr}(t)$ and y^*_v is the predicted label of each unlabeled target node $v \in V^{tar}(t)$ by GNN $\mathcal{M}_{\theta^*(t)}$. λ is a trade-off hyperparameter between misleading attackers and protecting target nodes. Also, $\triangle \mathbf{z}'_v(t)[y_v] = \mathbf{z}^d_v(t)[y_v] - \mathbf{z}_v(t)[y_v]$ is the difference between scores predicted on graph $G(t)$ and protected graph $G^d(t)$. $\triangle |N'_v(t)| = |N^d_v(t)| - |N_v(t)|$ is v's neighbor number difference. $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the `Sigmoid` function.

The first part of Equation (8) targets to reduce the score of the ground truth of more influential training nodes, thereby degrading the accuracy of attackers' predictions. The second part of Equation (8) targets to increase the predicted score and neighbor number of easier target nodes. Such two ways actively protect the target node against successful attacks by potential attackers. However, it is non-trivial to optimize the active defense objective due to the exponential search space for guardian node behaviors. Specifically, given the graph $G(t)$ and the guardian node number k_p, the number of guardian node behavior combinations is $O(2^{k_p|V(t)|+k_p d_x})$. Thus, it is infeasible to obtain the best behaviors by an exhaustive search. Then, we propose a greedy-based greedy algorithm in Algorithm 2 to solve the problem in Definition 4.3.

4.2.1 Gradient-based Greedy Algorithm. We illustrate the procedure of our gradient-based algorithm in Algorithm 2. Particularly, we propose two acceleration techniques, i.e., top-k training node selection (line 4) and candidate edge sampling (line 6), to improve the efficiency of Algorithm 2. These two acceleration techniques will be introduced in detail in Section 4.3. As shown in Algorithm 2, we first inject k_p guardian nodes $V^p(t)$ into the graph $G(t)$ (line 1). Then, we initialize the adjacency matrix $\mathbf{A}^d(t)$ and node features $\mathbf{X}^d(t)$ of the active defense graph based on zero padding (line 2-3). Specifically, given the adjacency matrix $\mathbf{A}(t) \in \{0,1\}^{|V(t)| \times |V(t)|}$, the $\mathbf{A}^d$ can be generated as $\tilde{\mathbf{A}}^d(t) = \begin{bmatrix} \mathbf{A}(t) & \tilde{\mathbf{A}}^p_2(t) \\ \tilde{\mathbf{A}}^p_1(t) & \tilde{\mathbf{A}}^p_3(t) \end{bmatrix}$, where $\tilde{\mathbf{A}}^p_1(t) \in \mathbb{R}^{|V^p(t)| \times |V(t)|}$ and $\tilde{\mathbf{A}}^p_2(t) \in \mathbb{R}^{|V(t)| \times |V^p(t)|}$ denotes the directed edges between $V^p(t)$ and $V(t)$, and $\tilde{\mathbf{A}}^p_3(t) \in \mathbb{R}^{|V^p(t)| \times |V^p(t)|}$ denotes the edges among nodes in $V^p(t)$. $\tilde{\mathbf{A}}^p_1(t), \tilde{\mathbf{A}}^p_2(t)$, and $\mathbf{A}^p_3(t)$ are initialized as 0, which will be tuned into continuous value based on gradients. Similarly, given node features $\mathbf{X}(t) \in \mathbb{R}^{|V(t)| \times d_x}$, $\tilde{\mathbf{X}}^d(t) = \begin{bmatrix} \mathbf{X}(t) \\ \tilde{\mathbf{X}}^p(t) \end{bmatrix}$, where $\tilde{\mathbf{X}}^p(t) \in \mathbb{R}^{|V^p(t)| \times d_x}$ are initialized as

Algorithm 2: Active Defense Graph Generation

```
Input: Graph $G(t) = (V(t), \mathbf{A}(t), \mathbf{X}(t))$ at time $t$, training nodes
       $V^{tr}(t)$ with labels $\mathbf{Y}(t)$, GNN model $\mathcal{M}_{\theta^*(t)}$, target nodes
       $V^{tar}(t)$, and protectecd node budget $k_p$
Output: The active defende graph under $k_p$ budget, i.e.,
       $G^d(t) = (V^d(t), \mathbf{A}^d(t), \mathbf{X}^d(t))$
1  $V^d(t) = V(t) \cup V^p(t)$
2  $\tilde{\mathbf{A}}^d(t) =$ Edge_Zero_Padding$(\mathbf{A}(t), V^p(t))$
3  $\tilde{\mathbf{X}}^d(t) =$ Feature_Zero_Padding$(\mathbf{X}(t), V^p(t))$
4  $V_s^{tr}(t) =$ NodeSelection$(V^{tr}(t), k_{tr})$ // Algorithm 3
5  for $i = 1$ to $n_e$ do
       // $idx$ is the flattened index of $\tilde{\mathbf{A}}^p(t)$ in $\mathbf{A}^d(t)$.
          The details are in Algorithm 4
6      $\tilde{\mathbf{A}}_s^p(t), idx =$ EdgeSample$(\tilde{\mathbf{A}}^d(t) \setminus \mathbf{A}(t), \tilde{\mathbf{X}}^d, s)$
7      Compute $AO(G^d, M_{\theta^*(t)}, V_s^{tr}(t), V^{tar}(t)) \leftarrow$ Equation (8)
8      $\tilde{\mathbf{A}}^d(t)[idx] = \tilde{\mathbf{A}}^d(t)[idx] + \alpha \cdot \nabla_{\tilde{\mathbf{A}}_s^p(t)} AO(\cdot)$
9      $\tilde{\mathbf{X}}^d(t)[V^p(t)] = \tilde{\mathbf{X}}^d(t)[V^p(t)] + \alpha \cdot \nabla_{\tilde{\mathbf{X}}^d(t)[V^p(t)]} AO(\cdot)$
10 $\mathbf{A}^d(t) =$ Project$(\tilde{\mathbf{A}}^d(t))$, $\mathbf{X}^d(t) =$ Project$(\tilde{\mathbf{X}}^d(t))$
11 Return $G^d(t) = (V^d(t), \mathbf{A}^d(t), \mathbf{X}^d(t))$
```

0 and will be tuned into continuous values based on gradients. In line 4, we propose to select a set of influential nodes $V_s^{tr}(t) \subseteq V^{tr}(t)$ to replace the whole $V^{tr}(t)$ to further accelerate the gradient computation (See details in Section 4.3.1).

In each epoch i, we first sample s crucial candidate edges $\tilde{\mathbf{A}}_s^p(t)$ from $\tilde{\mathbf{A}}^d(t) \backslash \mathbf{A}(t)$, i.e., $\{\tilde{\mathbf{A}}_1^p(t), \tilde{\mathbf{A}}_2^p(t), \tilde{\mathbf{A}}_3^p(t)\}$ in line 6 (See Section 9). Then, we compute the active objective $AO(\cdot)$ in Equation (8) based on the selected training nodes $V_s^{tr}(t)$, active graph $G^d(t)$, GNN model $M_{\theta^*(t)}$, and target nodes $V^{tar}(t)$ (line 7). Based on this, we use the stochastic gradient ascent approach [3] to compute the gradient of $\tilde{\mathbf{A}}_s^p(t)$ and $\tilde{\mathbf{X}}^p(t)$ that gives a direction to maximize the active defense objective. Based on gradients, we update the edges and features of these guardian nodes with the learning rate α (line 8-9). After finishing the optimization of the guardian node behaviors, we project the designed edges of each guardian node $v \in V^p(t)$ by setting the top-τ entry in $\tilde{\mathbf{A}}^d(t)[v]$ as 1 and others as 0. Also, we project the node features $\tilde{\mathbf{X}}^d(t)$ of nodes into the normal range of node features $\mathbf{X}^d(t) \in [-B, B]^{|V^d(t)| \times d_x}$, where B is the maximum absolute value of node features (line 10).

We analyze the time complexity of Algorithm 2 without two acceleration algorithms. Specifically, Algorithm 2 will take all training nodes $V^{tr}(t)$ to compute the active defense objective and optimize all candidate edges at each epoch. The analysis is as follows:

Time Complexity of Algorithm 2 without Accelerations. The time is $O(n_e D^L d_x^2 (|V^{tr}(t)| + |V^{tar}(t)|)(|V^p(t)||V(t)| + |V^p(t)|d_x))$. Please refer to Appendix A.5 in our technique report [41].

4.3 Acceleration Techniques

Generally, node feature dimension of real-world graphs is small. For example, AmazonProducts [87] and Yelp [87] shown in Table 2 only have 100 dimension. Thus, the time complexity of Algorithm 2 mainly depends on the number of training nodes $V^{tr}(t)$ and the number of candidate edges $|V^p(t)||V(t)|$. Therefore, we propose to select the top-k representative nodes $V_s^{tr}(t) \subseteq V^{tr}(t)$ to replace all training nodes $V^{tr}(t)$ and sample s crucial candidate edges

Algorithm 3: Greedy Node Selection Algorithm

```
Input: Training nodes $V^{tr}(t)$, GNN model $\mathcal{M}_{\theta^*(t)}$, target nodes
       $V^{tar}(t)$, and the budget $k_{tr}$
Output: Selected nodes $V_s^{tr}(t)$
1 $V_s^{tr}(t) \leftarrow \emptyset$
2 while $|V_s^{tr}(t)| < k$ do
3     $V_{sam}^{tr}(t) =$ Random_Sample$(V^{tr}(t) \setminus V_s^{tr}(t), n_s)$
4     for $v \in V_{sam}^{tr}(t)$ do
5         $RS(V_s^{tr}(t) \cup \{v\}) \leftarrow$ Equation (9)
6         $\triangle RS(v|V_s^{tr}(t)) = RS(V_s^{tr}(t) \cup \{v\}) - RS(V_s^{tr}(t))$
7     $v^* = \arg\max_{v \in V_{sam}^{tr}(t)} \triangle RS(v|V_s^{tr}(t))$
8     $V_s^{tr}(t) \leftarrow V_s^{tr}(t) \cup \{v^*\}$
9 Return selected training nodes $V_s^{tr}(t)$
```

to replace $|V^p(t)||V(t)|$ edges. Such two ways can accelerate the objective and gradient computation of Algorithm 2.

4.3.1 Top-k Training Node Selection. Intuitively, one simpler way is to select k_{tr} training nodes with the largest influence score $\mathcal{I}(u, V^{tar}(t))$. However, such a way may select a set of similar training nodes, which cannot fully represent the whole training nodes. This is because the influence score in Equation (6) is computed independently for each individual training node. As analyzed in the technical report Appendix A.1, even if we can extend Equation (6) to measure the importance of multiple nodes, it is impractical due to heavily time-consuming. Therefore, we propose to select the top-k influential and diverse training nodes. In such a way, we can effectively mislead attackers by perturbing these selected training nodes, achieving results similar to perturbing all training nodes. We define the top-k training node selection problem as follows.

Definition 4.4 (Top-k Training Node Selection Problem). Given the training nodes $V^{tr}(t)$, GNN model $\mathcal{M}_{\theta^*(t)}$, and target nodes $V^{tar}(t)$, the target is to select the k_{tr} nodes $V_s^{tr}(t)$ from $V^{tar}(t)$ by maximizing the following representative score $RS(V_s^{tr}(t))$:

$$\max_{V_s^{tr}(t) \subseteq V^{tr}(t)} \sum_{v \in V_s^{tr}} \mathcal{I}(v, V^{tar}(t)) + \sum_{v \in V_s^{tr}(t)} \mathcal{D}(v, V_s^{tr}(t)) \qquad (9)$$

where $\mathcal{D}(v, V_s^{tr}(t))$ is the diversity value of node v regarding $V_s^{tr}(t)$. Specifically, $\mathcal{D}(v, V_s^{tr}(t)) = c_{max} - \min_{u \in V_s^{tr}(t) \backslash v} ||\mathbf{h}_v(t) - \mathbf{h}_u(t)||$ is the maximum distance minus the minimum distance between the node representation of v and nodes in $V_s^{tr}(t)$, where $c_{max} = \max_{u_1, u_2 \in V^{tr}(t)} ||\mathbf{h}_{u_1}(t) - \mathbf{h}_{u_2}(t)||$ is the maximum distance among training nodes and $\mathbf{h}_v(t)$ is the node representation of node v.

Theorem 4.5. *The training node selection problem is NP-hard.*

Proof Sketch. We prove that our top-k training node selection problem is NP-hard based on the k-clique problem [68]. Due to space limits, we put full proof in technique report [41] Appendix A.3 □

Sampling-based Greedy Algorithm. Theorem 4.5 indicates that top-k training node selection problem is NP-hard. Therefore it is unlikely to obtain the optimal solution of this problem in polynomial time unless P=NP. In this subsection, we propose an efficient sampling-based greedy algorithm to address this problem. Specifically, we first define the marginal representative score gain brought by each node v if we select it into $V_s^{tr}(t)$. Formally, given selected

```
Algorithm 4: Candidate Edge Sampling
  Input: All candidate edges $\tilde{\mathbf{A}}^d(t) \setminus \mathbf{A}(t)$, node features
         $\tilde{\mathbf{X}}^d(t)$ and samling size $s$
  Output: Sampled $\tilde{\mathbf{A}}^p_s(t)$ with their flattened index list $idx$
1 Compute $w_{v,u}(t)$ for each pair of $v \in V^p(t)$ and $u \in V^d(t)$
2 $\tilde{\mathbf{A}}^p_s(t), idx = \text{MultiNomial}(\tilde{\mathbf{A}}^d(t) \setminus \mathbf{A}(t), w, s)$
3 Return Sampled $\tilde{\mathbf{A}}^p_s(t)$ with their flattened index list $idx$
```

nodes $V^{tr}_s(t)$, the marginal representative score gain of a node v can be defined as $\triangle RS(v|V^{tr}_s(t)) = RS(V^{tr}_s(t) \cup \{v\}) - RS(V^{tr}_s(t))$.

As shown in Algorithm 3, we will first initialize node set $V^{tr}_s(t)$ as an empty set (line 1). Then, we sample n_s nodes $V^{tr}_{sam}(t)$ from the un-selected training nodes $V^{tr}(t) \setminus V^{tr}_s(t)$. Finally, we will select the representative node with the maximum marginal representative gain $\triangle RS(v|V^{tr}_s(t))$ into $V^{tr}_s(t)$ from the sampled nodes $V^{tr}_{sam}(t)$ (line 2-8). We will repeat the selection process to obtain the selected training nodes $V^{tr}_s(t)$ until exceeding the node budget k_{tr}.

Theorem 4.6. *Given* $n_s = \frac{|V^{tr}(t)|}{k_{tr}} \log \frac{1}{\epsilon}$, *Algorithm 3 can achieve an approximation ratio of* $1 - 1/e - \epsilon$.

Proof Sketch. Let $V^{tr}_{s,i}(t)$ and $V^{tr}_{s,opt}(t)$ denote the solution of Algorithm 3 at the i-th iteration and the optimal solution of Equation (9), respectively. First, expectation satisfies $\mathbb{E}(\triangle RS(v|V^{tr}_{s,i}(t))) \geq \frac{1-\epsilon}{k} \sum_{u \in V^{tr}_{s,opt}(t) \setminus V^{tr}_{s,i}(t)} \triangle RS(u|V^{tr}_{s,i}(t))$. Then, $RS(V^{tr}_s(t)) \geq (1 - 1/e - \epsilon) \cdot RS(V^{tr}_{s,opt}(t))$. Due to space limits, we put the full proof in our technique report [41] Appendix A.4. □

Time Complexity of Algorithm 3. In line 3, it takes O($|V^{tr}(t)|$) to sample $V^{tr}_{sam}(t)$. in line 4-8, at each epoch, it takes $O(|V^{tr}_{sam}(t)| + |V^{tr}_{sam}(t)||V^{tr}_s(t)|d_x)$ to select one representative node. Thus, the total time complexity is $O(k_{tr}|V^{tr}(t)| + k_{tr}n_s + n_s k^2_{tr}/2)$.

4.3.2 Crucial Candidate Edge Sampling. We propose to sample a set of crucial candidate edges instead of all edges to compute active defense objective and gradients, thereby accelerating the guardian node behaviors optimization. As shown in Algorithm 4 line 1, we first compute a weight for each edge candidate. Specifically, Inspired by current researchers [31, 42], the label of a node v tends to be changed if we add more edges between it and dissimilar nodes. Thereby, we can add edges between training nodes and guardian nodes that are dissimilar to training nodes, thereby changing the labels of training nodes and misleading attackers. Similarly, we can add edges between target nodes with guardian nodes that are similar to target nodes, thereby protecting them. Additionally, as shown in Theorem 4.2, the node with a larger degree is more robust with attacks. Based on these two observations, we define the weight of each candidate edge between each pair of $v \in V^p(t)$ and $u \in V^d(t)$ as follows:

$$w_{v,u}(t) = \begin{cases} \frac{1 - cosine(\mathbf{x}^d_v(t), \mathbf{x}^d_u(t))}{2\log(|N_u(t)+1|)}, & u \in V^{tr}_s(t) \\ \frac{1 + cosine(\mathbf{x}^d_v(t), \mathbf{x}^d_u(t))}{2\log(|N_u(t)|+1)}, & u \in V^{tar}(t) \\ \frac{\sum_{u_1 \in V^{tr}_s(t) \cup V^{tar}(t)} w_{v,u_1}(t)}{|V^{tr}_s(t)| + |V^{tar}(t)|} \cdot, & u \in V(t) \setminus V^{tar}(t) \setminus V^{tr}_s(t) \end{cases}$$

Table 2: Dataset statistics summary, including the node number (#Node), feature dimension (#Feat), edge number (#Edge), timestamp number (#TS), and node class number (#Class).

	#Node	#Feat	#Edge	#Degree	#TS	#Class
Cora	2,708	1,433	10,556	3.90	10	7
DBLP	28,085	100	324,902	11.57	10	5
Yelp	716,847	300	13,954,819	19.47	10	100
Products	1,569,960	200	264,339,468	168.37	10	107

Then, based on weights, we sample s edges from all candidate edges based on multinomial sampling (line 2). We analyze the time complexity of Algorithm 4 as follows.

Time Complexity of Algorithm 4. It takes $O(|V^p(t)||V(t)|d_x)$ time to compute edge weights. Then, it takes $O(|V^p(t)||V(t)|)$ time to sample s edges. Thus, the time is $O(|V^p(t)||V(t)|d_x)$ in total.

4.3.3 Acceleration Analysis. We analyze the time complexity of Algorithm 2 with two acceleration techniques in Section 4.3.1 and Section 9. Specifically, Algorithm 3 will select top k_{tr} training nodes to compute the active defense objective and gradients for node behaviors. At each epoch i of Algorithm 2 (line 5-9), Algorithm 2 will optimize s crucial edges sampled by Algorithm 4 instead of optimizing $|V^p(t)||V(t)|$ candidate edges. Thus, by considering the training node selection time and crucial edge sampling time, the time complexity of Algorithm 2 will be decreased from $O(n_e D^L d^2_x(|V^{tr}(t)| + |V^{tar}(t)|)(|V^p(t)||V(t)| + |V^p(t)|d_x))$ to $O(t_{sel} + n_e D^L d^2_x(k_{tr} + |V^{tar}(t)|)(s + |V^p(t)|d_x) + t_{sam})$, where training selection time $t_{sel} = k_{tr}|V^{tr}(t)| + k_{tr}n_s + n_s k^2_{tr}/2$ and edge sample time $t_{sam} = |V^p(t)||V(t)|d_x$.

5 EXPERIMENTS

In this section, we compare our active GNN defender with state-of-the-art baselines on four real-world graph datasets.

5.1 Experiment Setting

Codes are implemented by PyTorch and are run on a CentOS 7 machine with a 20-core Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz, 8 NVIDIA GeForce RTX 2080 Ti GPUs (11G), and 256G of RAM.

5.1.1 Datasets. We introduce four widely-used graph datasets and the data statistics are summarized on Table 2. Cora [32] and DBLP [18, 78] are citation graphs. Yelp [87] are user social networks from a business review website. AmazonProducts (Products) [87] are item purchasing graph. For each dataset, we randomly choose 50% of nodes as training nodes, choose 10% of nodes as validation nodes. There exist 40% of remaining nodes. Then, at each time t, we randomly select 10% of nodes from the remaining 40% of nodes as the target nodes to evaluate our proposed active GNN defender. The details of datasets are in our technical report [41] Appendix B.1.

5.1.2 GNN Attackers. To evaluate our active defender, we utilize state-of-the-art attackers to attack the dynamic graphs, including Metattack [96], PRBCD [22], and TDGIA [94]. We employ sampling technique in PRBCD for Metattack and TDGIA for scalability. The details are in technical report [41] Appendix B.2.

5.1.3 GNN Defender Baselines. We choose six GNNs as baselines. Specifically, GCN [32] and RoLAND [84] are two representative

Table 3: Node classification performance (Accuracy±Std). GNN-J. denotes GNN-Jaccard. OOM indicates out of GPU memory.

	Cora			**DBLP**			**Yelp**			**Products**		
	Mettack	**TDGIA**	**RBGCD**	**Mettack**	**TDGIA**	**RBGCD**	**Mettack**	**TDGIA**	**RBGCD**	**Mettack**	**TDGIA**	**RBGCD**
GCN	76.21±1.27	75.81±0.98	76.26±1.72	52.72±1.82	51.59±2.01	50.29±1.67	51.24±1.36	52.16±1.28	50.43±1.62	23.12±0.28	23.36±0.33	23.02±0.39
RoLAND	76.76±1.34	76.12±1.18	77.24±0.83	52.53±1.92	51.27±1.65	51.14±1.73	51.28±1.92	51.82±1.71	51.78±1.83	23.16±0.51	23.23±0.39	23.13±0.58
GNNSVD	78.26±1.79	77.92±1.32	77.10±1.65	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
ProGNN	82.37±1.37	82.96±1.87	81.96±1.85	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
GNN-J.	79.28±1.49	80.23±1.31	78.98±1.52	56.23±1.76	54.23±2.32	55.43±2.10	52.86±1.72	53.03±1.27	51.24±1.36	23.53±0.66	23.41±0.38	23.39±0.65
GNAT	**83.82±1.61**	83.21±1.71	82.83±1.53	57.10±2.13	56.03±1.96	56.52±1.87	52.12±1.59	53.29±1.85	52.17±1.62	23.81±0.47	23.73±0.63	23.65±0.51
ADGNN	83.71±1.39	**83.92±1.50**	**83.27±1.64**	**60.27±1.79**	**59.28±2.03**	**58.13±1.91**	**52.87±1.86**	**53.41±1.92**	**53.05±1.75**	**24.22±0.62**	**23.92±0.56**	**23.81±0.76**

raw GNNs that do not consider attacks. Also, GNN-Jaccard [79], GNN-SVD [15], ProGNN [31], and GNAT [42] are passive GNN defenders and take attacked graphs to generate purified graphs. In particular, we employ state-of-the-art dynamic RoLAND [84] as the GNN encoder of these GNN defenders. Also, to compute the influence score efficiently for large graphs (i.e., DBLP, Yelp, and Products), we use 2-layer MLP to replace the GNN in Equation (6), which directly takes node features to predict labels The details of baselines are in our technique report [41] Appendix B.3.

5.1.4 Evaluation Procedure. For clarification, we elaborately introduce the evaluation procedure for passive GNN defenders and active GNN defenders on dynamic graphs.

- **Passive GNN Defenders and Raw GNNs.** Given graph $G(t)$ at each time t, we first use GNN attackers in Section 5.1.2 to inject k_a malicious nodes to obtain the attacked graph $G^a(t)$. Then, with normal evolving graph events (i.e., edge deletion/addition) during t and $t+1$, we can get $G(t+1)$. We will evaluate the passive GNN defender and raw GNNs on $G(t+1)$ by predicting the labels of target nodes $V^{tar}(t)$.
- **Our Active GNN Defender**: As shown in Algorithm 1 (lines 2-7), given graph $G(t)$ at each time t, our active defender first injects k_p guardian nodes in $G(t)$. Then, we obtain protected graph $G^p(t)$ to avoid potential effective attacks on target nodes $V^{tar}(t)$. Then, we use GNN attackers to inject k_a malicious nodes in graph $G^p(t)$. With normal evolving graph events during t and $t+1$, we can get $G(t+1)$, and train a GNN on $G(t+1)$ to predict the labels of target nodes $V^{tar}(t)$. Particularly, when training the GNN, we remove all injected guardian nodes to avoid the negative impacts of guardian nodes on training nodes.

In particular, given target nodes $V^{tar}(t)$ at time t, we will evaluate whether $V^{tar}(t)$ are successfully attacked at time $t+1$. Thus, given a graph with T time snapshots, we use the mean accuracy tested on target nodes from the second time to the T time for evaluation, which can be defined as: $\overline{Acc} = \frac{1}{T-1}\sum_{t=2}^{T}\frac{\sum_{v\in V^{tar}(t-1)}\mathbb{I}(y_v^*==y_v)}{|V^{tar}(t-1)|}$. Besides, we report the average accuracy of three different runs.

5.1.5 Hyperparameter Setting. For attackers in Section 5.1.2 and GNN defender baselines in Section 5.1.3, we use their default hyperparameter settings. In experiments except for Section B.4, attackers and our GNN active defenders only inject nodes at the first time once and design the behaviors of these nodes at each time. We set the malicious node budget of attackers $k_a = r_a \cdot |V(T)|$ where $|V(T)|$ is the node number of the whole platform, and set $r_a = 0.01$ as the default. Also, we set the node protection budget as $k_p = r_p \cdot |V(T)|$ and set $r_p = 0.01$ as the default. In Algorithm 3, we set the number of selected training nodes as $k_{tr} = r_{tr} \cdot |V^{tr}(t)|$ and tune $r_{tr} \in \{0.1, 0.2, 0.3, \cdots, 1\}$ based on validation data. We tune λ in Equation (8) by $\lambda \in \{0.1, 0.2, 0.3, \cdots, 1\}$ based on validation data. For both attackers and our active defender, we set the maximum edges of each injected node that can connect as the average degree of nodes of each data in Table 2. In such a way, the behaviors of injected nodes are more likely to normal nodes and can avoid injected node detection.

5.2 Main Results

5.2.1 Effectiveness Evaluation. We report the performance of GNN defenders against attackers on four datasets in Table 3. A higher node classification accuracy indicates a more robust GNN defender. As shown in Table 3, under the same attacker, the raw GNNs, including GCN and RoLAND, achieve unsatisfying performance since they are directly trained on the attacked graph and cannot predict the labels of target nodes correctly. Second, the passive GNN defenders do not outperform our active GNN defenders. It is because they purify the attacked graph for platforms, and such a passive defense manner cannot prevent attackers from generating effective attacks. Also, these passive GNN defenders are proposed to resist specific attack patterns and cannot resist different attackers.

Besides, ProGNN and GNN-SVD cannot scale well on DBLP, Yelp, and Products datasets, i.e., they meet the out of GPU memory (OOM) issue. This is because ProGNN and GNN-SVD directly optimize the whole adjacent matrix by decreasing the rank of the adjacent matrix on GPU, significantly exceeding the GPU memory. Our proposed active GNN defender ADGNN achieves the best and most comparable performance under different attackers and graph datasets, which demonstrates the effectiveness of ADGNN. It is mainly because before each time attackers inject malicious nodes, ADGNN actively injects nodes to protect the graph from two aspects. First, these injected guardian nodes can change the labels of important training nodes to degrade the effectiveness of attackers. Second, the injected guardian node can protect nodes of easily attacked target nodes.

5.2.2 Deployment with Passive Defenders. As introduced in Section 1, our active GNN defender and existing passive GNN defender alleviate the negative impact of attacks on $V^{tar}(t)$ in different phases. Here, we more comprehensively explore how our active GNN defenders help existing passive defenders resist attacks. Specifically, at each time t, we first use our ADGNN to inject protected graphs to avoid effective attacks. Then, after attackers attack the

Table 4: Node classification performance (Accuracy±Std) on four graph datasets with Attacker RBGCD.

	Cora	DBLP	Yelp	Products
GNN-SVD	77.10±1.65	OOM	OOM	OOM
GNN-SVD++	**83.47±1.89**	OOM	OOM	OOM
ProGNN	81.96±1.85	OOM	OOM	OOM
ProGNN++	**83.92±1.71**	OOM	OOM	OOM
GNN-Jaccard	78.98±1.52	55.43±2.10	51.24±1.36	23.39±0.65
GNN-Jaccard++	**83.56±2.07**	**58.87±2.11**	**53.17±1.52**	**24.02±0.50**
GNAT	82.83±1.53	56.52±1.87	52.17±1.62	23.65±0.51
GNAT++	**84.12±2.13**	**59.24±1.92**	**53.31±1.49**	**23.90±0.62**

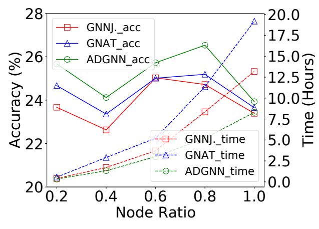
protected graph during t and $t+1$, we use the passive GNN defenders to purify the attacked graphs for platform GNNs. In particular, we use [model++] to denote that this [model] is deployed with ADGNN. Since the trend of GNN defenders under different attackers is similar in Table 3, we use RBGCD attacker for evaluation.

As shown in Table 4, after deploying ADGNN, all passive GNN defenders improve their performance. It demonstrates that ADGNN can actively degrade the effectiveness of attackers. Besides, the performance of ADGNN with one passive GNN defender, such as ADGNN+GNAT (i.e., GNAT++), is better than the performance of a single ADGNN in Table 3. It demonstrates that after degrading the effectiveness of attackers by ADGNN, existing passive defenders can further help purify the graph to improve the performance on target nodes together. These observations demonstrate the high practicability of ADGNN: (1) It provides a new active manner to resist attackers on real-world dynamic graphs. (2) It can seamlessly integrate with existing passive GNN defenders to boost their performance in defending against attackers.

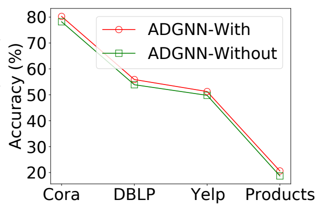
5.2.3 Scalability Evaluation. To comprehensively explore the scalability of ADGNN and baselines, we use $\beta \in \{20\%, 40\%, 60\%, 80\%, 100\%\}$ nodes of Products dataset to construct subgraphs for evaluation. Specifically, given all nodes V of Products, we use the first $\beta|V|$ nodes and only keep temporal events that happen in these $\beta|V|$ nodes. Since the trend of GNN defenders under different attackers is similar in Table 3, we use RBGCD attacker for evaluation. We compare ADGNN with existing passive GNN defenders. ProGNN and GNNSVD are not scalable, i.e., they meet the out of GPU memory problem. As shown in Figure 3 (a), the time of GNN-Jaccard and GNAT increases as the node number increases. It is because the time complexity of their purifying the attacked graph step is $O(|V(t)|^2 d_x)$, i.e., they need to compute the similarity between each pair of nodes. Instead, as analyzed in Section 4.3.3, the time complexity of ADGNN is dominated by $O(|V^{tar}(t)||V^p(t)|)$, which is significantly less than GNAT and GNN-Jaccard. Consequently, ADGNN requires less computational time than both GNAT and GNN-Jaccard. Also, the node prediction accuracy of ADGNN outperforms GNAT and GNN-Jaccard at different node ratios, indicating that ADGNN degrades attackers to generate effective attacks.

5.3 Ablation Study

5.3.1 Effect of Previously Guardian Nodes. We evaluate the effect of previously injected guardian nodes before time t on the protection of target nodes at each time t. Specifically, we set that the attacker



(a) Scalability evaluation.



(b) Effect of previous guardian nodes.

Figure 3: Evaluation on scalability and guardian nodes.

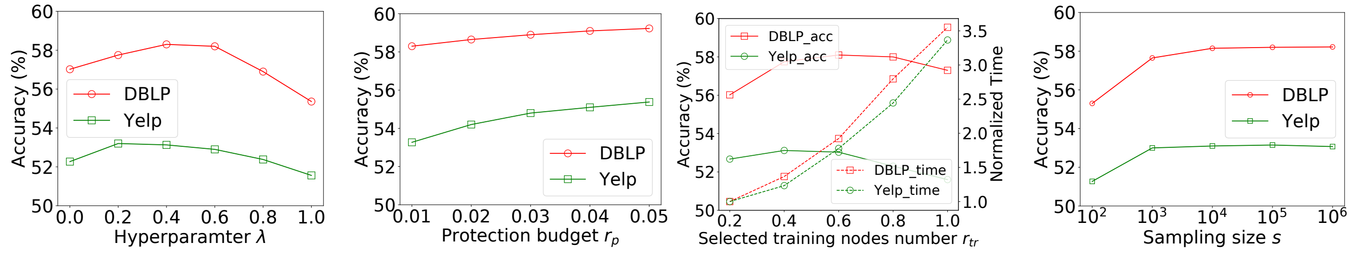
Table 5: Node classification performance (Accuracy±Std) on two graphs under the attacker RBGCD. A. is the abbreviation of ADGNN. Time(s) is the mean time of injecting guardian nodes on all snapshots.

Variants	DBLP		Yelp	
	Accuracy	Time (s)	Accuracy	Time (s)
A.\S	OOM	-	OOM	-
A.\TopK	57.61±2.36	2082	51.29±1.82	26298
A.\S\TopK	OOM	-	OOM	-
ADGNN	**58.13±1.91**	872	**53.05±1.75**	7629

and our proposed ADGNN inject $0.01 \cdot |V(t)|$ new nodes at each time t. We evaluate two variants, ADGNN-With and ADGNN-Without. When injecting new guardian nodes at each time t, ADGNN-With keeps the guardian nodes injected before time t, and ADGNN-Without removes all guardian nodes injected before time t. As indicated by the results in Figure 3 (b), ADGNN-With consistently outperforms ADGNN-Without across all four datasets in terms of node classification accuracy. This superiority suggests that the guardian nodes injected before time t can help platforms protect nodes at time t. The reason is that the guardian nodes before t, even when the target nodes differ across time steps, can still interfere with the natural information flow within the graph at time t. This interference hampers an attacker's ability to learn accurate and intrinsic representations of the target nodes, thereby reducing the efficacy of their attacks.

5.3.2 Framework. We investigate the effect of different parts of ADGNN. Specifically, we propose the gradient-based algorithm in Algorithm 2 and propose to select top-k training nodes (Algorithm 3) and sample candidate edges (Algorithm 4) to accelerate Algorithm 2. We denote Algorithm 2 without the support of Algorithm 3 as **ADGNN\TopK**, without Algorithm 4 as **ADGNN\S**, and without both Algorithm 3 and Algorithm 4 as **ADGNN\S\TopK**. Particularly, due to the large size of dynamic graphs, ADGNN\TopK and ADGNN\S\TopK repeatedly first sample training nodes and then update the sampled edge weighs. In particular, since results are similar under different attackers and datasets, we report the accuracy and time on both DBLP and Yelp under the attacker RBGCD.

As shown in Table 5, ADGNN\S and ADGNN\S\TopK are not scalable and meet the out of GPU memory (OOM) issue. It is because they directly compute the gradients for all candidate edges and node features of injected guardian nodes, exceeding the capacity of GPU memory. ADGNN\TopK takes more time, since they need to compute active defense objective in Equation (8) and gradients of



(a) Hyperparameter λ. **(b) The budget of guardian nodes.** **(c) Selected training nodes number.** **(d) Sampling size.**

Figure 4: Parameter sensitivity evaluations.

guardian node behaviors based on all training nodes. Also, ADGNN outperforms ADGNN\TopK. The reason is that Algorithm 3 selects the most influential training nodes regarding the target nodes. Then, under limited node budget, it is more effective to design guardian node behaviors to affect the node labels of these top-k nodes. In other words, these limited protection nodes are incapable of affecting the node labels of all training nodes, thereby decreasing the performance of ADGNN\TopK.

5.4 Parameter Sensitivity Analysis

Since results are similar under different attackers and datasets, we report results on DBLP and Yelp datasets under the attacker RBGCD.

5.4.1 The hyperparameter λ. We investigate the effects of the trade-off parameter λ in Equation (8) between affecting influential training nodes to mislead attackers and protecting easily attacked target nodes. A higher λ more emphasizes protecting the target nodes. Specifically, we vary $\lambda \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. As shown in Figure 4 (a), as λ increases, the performance of ADGNN on both datasets first increases and then decreases. It indicates that only considering affect training nodes ($\lambda = 0$) and protecting target nodes ($\lambda = 1$) cannot achieve the best performance. Instead, by incorporating both misleading attackers and protecting target nodes, ADGNN can achieve the best performance.

5.4.2 Guardian node budget k_p. As introduced in Section 5.1.5, we control the number of guardian nodes by the parameter r_p, i.e., $k_p = r_p|V(t)|$. Here, we vary $r_p \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$. As shown in Figure 4 (b), as the budget of protection nodes increases, the node classification accuracy on target nodes increases as well. It indicates that with a more protection node budget, our proposed ADGNN can degrade the effectiveness of attacks and avoid potentially effective attacks. With more guardian nodes, we gain the capacity to perturb a larger subset of training nodes, leading to more affecting their labels. Then, this will disrupt the attackers' ability to accurately measure the impact of their malicious node behaviors on the target nodes. Consequently, attackers are prevented from generating effective attacks.

5.4.3 The number of selected training nodes k_{tr}. We investigate the effects of the number of the selected training nodes in Algorithm 3, i.e., $k_{tr} = r_{tr}|V^{tr}(t)|$. Here, we vary $r_{tr} \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. For clarification, we normalize the time under different r_{tr} by dividing it by the time under $r_{tr} = 0.2$. As shown in Figure 4 (c), as k_{tr} increases, node classification accuracy first increases and decreases. It indicates that partial training nodes can represent all training nodes. Also, a larger number of training nodes decreases the node classification accuracy. It is because limited protection nodes cannot fully affect a large number of training nodes to mislead the attackers and thereby cannot avoid effective adversarial attacks. Also, as k_{tr} increases, the consumed time increases, since ADGNN take k_{tr} training nodes to compute the objective and the gradients.

5.4.4 Sampling size s. We investigate the effects of the sampling size s in In Algorithm 4 by varying $s \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$. As shown in Figure 4 (d), as the sampling size increases, the performance on both datasets first increases and then keeps stable. It is because the exploration of a smaller search space is limited, and as a consequence, the generated protection node behaviors remain suboptimal. As sample size s continues to increase, ADGNN is capable of sampling a more extensive set of critical candidate edges for optimization. As a result, ADGNN can generate effective protection nodes and achieve satisfying performance.

6 CONCLUSION

In this paper, we propose a novel active GNN defender ADGNN designed for dynamic graphs, which aims at protecting nodes from effective adversarial attacks. ADGNN proactively injects guardian nodes into graphs to disrupt the predictions of attackers and protect vulnerable nodes, ultimately preventing attackers from generating effective attacks. Extensive experiments on four real-world datasets validate the effectiveness of our proposed defender and demonstrate its seamless integration with existing passive GNN defenders.

ACKNOWLEDGMENTS

This work is supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Guangdong Province Science and Technology Plan Project 2023A0505030011, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants.

REFERENCES

[1] Amazon. 2024. amazon.com.au. https://www.amazon.com.au/. [Accessed 10-02-2024].

[2] Elisa Bertino, Gabriel Ghinita, Ashish Kamra, et al. 2011. Access control for databases: Concepts and systems. Foundations and Trends® in Databases 3, 1–2 (2011), 1–148.

[3] Aleksandr Beznosikov, Eduard Gorbunov, Hugo Berard, and Nicolas Loizou. 2023. Stochastic gradient descent-ascent: Unified theory and new efficient methods. In International Conference on Artificial Intelligence and Statistics. PMLR, 172–235.

[4] Ji-Won Byun and Ninghui Li. 2008. Purpose based access control for privacy protection in relational database systems. The VLDB Journal 17 (2008), 603–619.

[5] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Wenwu Zhu, and Junzhou Huang. 2020. A restricted black-box adversarial framework towards attacking graph embedding models. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34. 3389–3396.

[6] Jinyin Chen, Jian Zhang, Zhi Chen, Min Du, and Qi Xuan. 2021. Time-aware gradient attack on dynamic network link prediction. IEEE Transactions on Knowledge and Data Engineering (2021).

[7] Yue Cui, Kai Zheng, Dingshan Cui, Jiandong Xie, Liwei Deng, Feiteng Huang, and Xiaofang Zhou. 2021. METRO: a generic graph neural network framework for multivariate time series forecasting. Proceedings of the VLDB Endowment 15, 2 (2021), 224–236.

[8] Gunduz Vehbi Demirci, Aparajita Haldar, and Hakan Ferhatosmanoglu. 2022. Scalable Graph Convolutional Network Training on Distributed-Memory Systems. Proc. VLDB Endow. 16, 4 (2022), 711–724. https://www.vldb.org/pvldb/vol16/p711-demirci.pdf

[9] Shimin Di and Lei Chen. 2023. Message Function Search for Knowledge Graph Embedding. In Proceedings of the ACM Web Conference 2023. 2633–2644.

[10] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2t: Instance-optimal truncation for differentially private query evaluation with foreign keys. In Proceedings of the 2022 International Conference on Management of Data. 759–772.

[11] Wei Dong and Ke Yi. 2021. Residual sensitivity for differentially private multi-way joins. In Proceedings of the 2021 International Conference on Management of Data. 432–444.

[12] Wei Dong and Ke Yi. 2022. A Nearly Instance-optimal Differentially Private Mechanism for Conjunctive Queries. In Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. 213–225.

[13] Nathan Drenkow, Numair Sani, Ilya Shpitser, and Mathias Unberath. 2021. A systematic review of robustness in deep learning for computer vision: Mind the gap? arXiv preprint arXiv:2112.00639 (2021).

[14] Chi Thang Duong, Trung Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, and Karl Aberer. 2021. Efficient streaming subgraph isomorphism with graph neural networks. Proceedings of the VLDB Endowment 14, 5 (2021), 730–742.

[15] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. 2020. All you need is low (rank) defending against adversarial attacks on graphs. In Proceedings of the 13th International Conference on Web Search and Data Mining. 169–177.

[16] Houxiang Fan, Binghui Wang, Pan Zhou, Ang Li, Zichuan Xu, Cai Fu, Hai Li, and Yiran Chen. 2021. Reinforcement learning-based black-box evasion attacks to link prediction in dynamic graphs. In 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys). IEEE, 933–940.

[17] Lijie Fan, Sijia Liu, Pin-Yu Chen, Gaoyuan Zhang, and Chuang Gan. 2021. When does contrastive learning preserve adversarial robustness from pretraining to finetuning? Advances in neural information processing systems 34 (2021), 21480–21492.

[18] Yucai Fan, Yuhang Yao, and Carlee Joe-Wong. 2021. Gcn-se: Attention as explainability for node classification in dynamic graphs. In 2021 IEEE International Conference on Data Mining (ICDM). IEEE, 1060–1065.

[19] Fuli Feng, Xiangnan He, Jie Tang, and Tat-Seng Chua. 2019. Graph adversarial training: Dynamically regularizing based on graph structure. IEEE Transactions on Knowledge and Data Engineering (2019).

[20] Jun Gao, Jiazun Chen, Zhao Li, and Ji Zhang. 2021. ICS-GNN: lightweight interactive community search via graph neural network. Proceedings of the VLDB Endowment 14, 6 (2021), 1006–1018.

[21] Shihong Gao, Yiming Li, Yanyan Shen, Yingxia Shao, and Lei Chen. 2024. ETC: Efficient Training of Temporal Graph Neural Networks over Large-scale Dynamic Graphs. Proc. VLDB Endow. 17, 5 (2024), 1060–1072. https://www.vldb.org/pvldb/vol17/p1060-gao.pdf

[22] Simon Geisler, Tobias Schmidt, Hakan Şirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. 2021. Robustness of graph neural networks at scale. Advances in Neural Information Processing Systems 34 (2021), 7637–7649.

[23] Aritra Ghosh and Andrew Lan. 2021. Contrastive learning improves model robustness under label noise. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2703–2708.

[24] Goodreads. 2024. Goodreads — goodreads.com. https://www.goodreads.com/. [Accessed 10-02-2024].

[25] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In Advances in neural information processing systems. 1024–1034.

[26] Tianjin Huang, Vlado Menkovski, Yulong Pei, and Mykola Pechenizkiy. 2020. Bridging the performance gap between fgsm and pgd adversarial training. arXiv preprint arXiv:2011.05157 (2020).

[27] Wenbing Huang et al. 2018. Adaptive sampling towards fast graph representation learning. In Advances in neural information processing systems. 4558–4567.

[28] IMDb. 2024. IMDb: Ratings, Reviews, and Where to Watch the Best Movies & TV Shows — imdb.com. https://www.imdb.com/. [Accessed 10-02-2024].

[29] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang. 2022. Query driven-graph neural networks for community search: from non-attributed, attributed, to interactive attributed. Proceedings of the VLDB Endowment 15, 6 (2022), 1243–1255.

[30] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. 2021. Node similarity preserving graph convolutional networks. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining. 148–156.

[31] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 66–74.

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

[33] Boris Knyazev, Carolyn Augusta, and Graham W Taylor. 2019. Learning Temporal Attention in Dynamic Graphs with Bilinear Interactions. arXiv preprint arXiv:1909.10367 (2019).

[34] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In International conference on machine learning. PMLR, 1885–1894.

[35] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. arXiv preprint arXiv:1611.01236 (2016).

[36] Janet Layne, Justin Carpenter, Edoardo Serra, and Francesco Gullo. 2023. Temporal SIR-GN: Efficient and Effective Structural Representation Learning for Temporal Graphs. Proceedings of the VLDB Endowment 16, 9 (2023), 2075–2089.

[37] Haoyang Li and Lei Chen. 2021. Cache-based GNN System for Dynamic Graphs. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 937–946.

[38] Haoyang Li and Lei Chen. 2023. EARLY: Efficient and Reliable Graph Neural Network for Dynamic Graphs. Proceedings of the ACM on Management of Data 1, 2 (2023), 1–28.

[39] Haoyang LI, Shimin Di, and Lei Chen. 2022. Revisiting Injective Attacks on Recommender Systems. Advances in Neural Information Processing Systems 35 (2022), 29989–30002.

[40] Haoyang Li, Shimin Di, Lei Chen, and Xiaofang Zhou. 2024. E2GCL: Efficient and Expressive Contrastive Learning on Graph Neural Networks. In 2024 IEEE 40th International Conference on Data Engineering (ICDE). IEEE.

[41] Haoyang Li, Shimin Di, Calvin Hong Yi Li, Lei Chen, and Xiaofang Zhou. 2024. Fight Fire with Fire: Towards Robust Graph Neural Networks on Dynamic Graphs via Actively Defense-Techniqcal Report. Online (2024). https://drive.google.com/drive/folders/1jGIm0G21kP9BJRFNSs6lbwiQp8cZr2LI?usp=sharing

[42] Haoyang Li, Shimin Di, Zijian Li, Lei Chen, and Jiannong Cao. 2022. Black-box Adversarial Attack and Defense on Graph Neural Networks. In 2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, 1017–1030.

[43] Jintang Li, Tao Xie, Chen Liang, Fenfang Xie, Xiangnan He, and Zibin Zheng. 2021. Adversarial attack on large scale graph. IEEE Transactions on Knowledge and Data Engineering (2021).

[44] Kuan Li, Yang Liu, Xiang Ao, and Qing He. 2022. Revisiting graph adversarial attack and defense from a data distribution perspective. In The Eleventh International Conference on Learning Representations.

[45] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Orca: Scalable Temporal Graph Neural Network Training with Theoretical Guarantees. Proceedings of the ACM on Management of Data 1, 1 (2023), 1–27.

[46] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. Proceedings of the VLDB Endowment 16, 6 (2023), 1332–1345.

[47] Zhiyuan Li, Xun Jian, Yue Wang, Yingxia Shao, and Lei Chen. 2024. DAHA: Accelerating GNN Training with Data and Hardware Aware Execution Planning. Proc. VLDB Endow. 17, 6 (2024), 1364–1376.

[48] Zhao Li, Xin Shen, Yuhang Jiao, Xuming Pan, Pengcheng Zou, Xianling Meng, Chengwei Yao, and Jiajun Bu. 2020. Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In 2020 IEEE 36th International

Conference on Data Engineering (ICDE). IEEE, 1677–1688.

[49] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. 2022. SCARA: scalable graph neural networks with feature-oriented optimization. Proceedings of the VLDB Endowment 15, 11 (2022), 3240–3248.

[50] Husong Liu, Shengliang Lu, Xinyu Chen, and Bingsheng He. 2020. G3: when graph neural networks meet parallel graph processing systems on GPUs. Proceedings of the VLDB Endowment 13, 12 (2020), 2813–2816.

[51] Jiaqi Ma, Junwei Deng, and Qiaozhu Mei. 2021. Near-Black-Box Adversarial Attacks on Graph Neural Networks as An Influence Maximization Problem. https://openreview.net/forum?id=sbyjwhxxT8K

[52] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. 2020. Black-box adversarial attacks on graph neural networks with limited node access. arXiv preprint arXiv:2006.05057 (2020).

[53] Yao Ma, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. 2018. Streaming graph neural networks. arXiv preprint arXiv:1810.10627 (2018).

[54] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013).

[55] Felix Mujkanovic, Simon Geisler, Stephan Günnemann, and Aleksandar Bojchevski. 2022. Are Defenses for Graph Neural Networks Robust? Advances in Neural Information Processing Systems 35 (NeurIPS 2022) (2022).

[56] Aldo Pareja et al. 2020. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In Proceedings of the AAAI conference on artificial intelligence, Vol. 34. 5363–5370.

[57] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637 (2020).

[58] RottenTomatoes. 2024. Rotten Tomatoes. https://www.rottentomatoes.com/. [Accessed 10-02-2024].

[59] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In Proceedings of the 13th International Conference on Web Search and Data Mining. 519–527.

[60] Vikash Sehwag, Arjun Nitin Bhagoji, Liwei Song, Chawin Sitawarin, Daniel Cullina, Mung Chiang, and Prateek Mittal. 2019. Analyzing the robustness of open-world machine learning. In Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security. 105–116.

[61] Zezhi Shao, Zhao Zhang, Wei Wei, Fei Wang, Yongjun Xu, Xin Cao, and Christian S Jensen. 2022. Decoupled dynamic spatial-temporal graph neural network for traffic forecasting. Proceedings of the VLDB Endowment 15, 11 (2022), 2733–2746.

[62] Samuel Henrique Silva and Peyman Najafirad. 2020. Opportunities and challenges in deep learning adversarial robustness: A survey. arXiv preprint arXiv:2007.00753 (2020).

[63] Lichao Sun, Yingtong Dou, Carl Yang, Kai Zhang, Ji Wang, S Yu Philip, Lifang He, and Bo Li. 2022. Adversarial attack and defense on graph data: A survey. IEEE Transactions on Knowledge and Data Engineering (2022).

[64] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. 2021. Building enclave-native storage engines for practical encrypted databases. Proceedings of the VLDB Endowment 14, 6 (2021), 1019–1032.

[65] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013).

[66] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1705.07204 (2017).

[67] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2018. Dyrep: Learning representations over dynamic graphs. (2018).

[68] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In Proceedings of the 24th international conference on world wide web. 1122–1132.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in neural information processing systems. 5998–6008.

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

[71] Sheng Wang, Yiran Li, Huorong Li, Feifei Li, Chengjin Tian, Le Su, Yanshan Zhang, Yubing Ma, Lie Yan, Yuanyuan Sun, et al. 2022. Operon: An encrypted database for ownership-preserving data management. Proceedings of the VLDB Endowment 15, 12 (2022), 3332–3345.

[72] Xuhong Wang et al. 2021. APAN: Asynchronous propagation attention network for real-time temporal graph embedding. In Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data. 2628–2638.

[73] Zhili Wang, Shimin Di, and Lei Chen. 2023. A Message Passing Neural Network Space for Better Capturing Data-dependent Receptive Fields. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2489–2501.

[74] Wikipedia. 2023. Taylor's theorem — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Taylor's%20theorem&oldid=1174353983. [Online; accessed 22-September-2023].

[75] Eric Wong, Leslie Rice, and J Zico Kolter. 2020. Fast is better than free: Revisiting adversarial training. arXiv preprint arXiv:2001.03994 (2020).

[76] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In International conference on machine learning. PMLR, 6861–6871.

[77] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. 2019. Adversarial examples on graph data: Deep insights into attack and defense. arXiv preprint arXiv:1903.01610 (2019).

[78] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Yameng Gu, Xiao Liu, Jingchao Ni, Bo Zong, Haifeng Chen, and Xiang Zhang. 2019. Adaptive neural network for node classification in dynamic networks. In 2019 IEEE International Conference on Data Mining (ICDM). IEEE, 1402–1407.

[79] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology attack and defense for graph neural networks: An optimization perspective. arXiv preprint arXiv:1906.04214 (2019).

[80] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018).

[81] Xiaojun Xu, Yue Yu, Bo Li, Le Song, Chengfeng Liu, and Carl Gunter. 2018. Characterizing malicious edges targeting on graph neural networks. (2018).

[82] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S Bhowmick. 2020. Scaling attributed network embedding to massive graphs. Proceedings of the VLDB Endowment 14, 1 (2020), 37–49.

[83] Yelp. 2024. Yelp — yelp.com. https://www.yelp.com/. [Accessed 10-02-2024].

[84] Jiaxuan You, Tianyu Du, and Jure Leskovec. 2022. ROLAND: Graph Learning Framework for Dynamic Graphs. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2358–2366.

[85] YouTube. 2024. YouTube — youtube.com. https://www.youtube.com/. [Accessed 10-02-2024].

[86] Wenhui Yu, Xiao Lin, Jinfei Liu, Junfeng Ge, Wenwu Ou, and Zheng Qin. 2021. Self-propagation Graph Neural Network for Recommendation. IEEE Transactions on Knowledge and Data Engineering (2021).

[87] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. arXiv preprint arXiv:1907.04931 (2019).

[88] Ao Zhang and Jinwen Ma. 2020. Defensevgae: Defending against adversarial attacks on graph data via a variational graph autoencoder. arXiv preprint arXiv:2006.08900 (2020).

[89] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. 2021. GRAIN: improving data efficiency of gra ph neural networks via diversified in fluence maximization. Proceedings of the VLDB Endowment 14, 11 (2021), 2473–2482.

[90] Xiang Zhang and Marinka Zitnik. 2020. Gnnguard: Defending graph neural networks against adversarial attacks. arXiv preprint arXiv:2006.08149 (2020).

[91] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: efficient graph neural network training at large scale. Proceedings of the VLDB Endowment 15, 6 (2022), 1228–1242.

[92] Yanping Zheng, Zhewei Wei, and Jiajun Liu. 2023. Decoupled Graph Neural Networks for Large Dynamic Graphs. Proc. VLDB Endow. 16, 9 (2023), 2239–2247. https://www.vldb.org/pvldb/vol16/p2239-zheng.pdf

[93] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 1399–1407.

[94] Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. 2021. Tdgia: Effective injection attacks on graph neural networks. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2461–2471.

[95] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2847–2856.

[96] Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. arXiv preprint arXiv:1902.08412 (2019).