Revisiting Adversarial Attacks on Graph Neural Networks for Graph Classification

Xin Wang, *Member, IEEE*, Heng Chang, Beini Xie, Tian Bian, Shiji Zhou, Daixin Wang, Zhiqiang Zhang, and Wenwu Zhu, *Fellow, IEEE*

Abstract—Graph neural networks (GNNs) have achieved tremendous success in the task of graph classification and its diverse downstream real-world applications. Despite the huge success in learning graph representations, current GNN models have demonstrated their vulnerability to potentially existent adversarial examples on graph-structured data. Existing approaches are either limited to structure attacks or restricted to local information, urging for the design of a more general attack framework on graph classification, which faces significant challenges due to the complexity of generating *local-node-level* adversarial examples using the *global-graph-level* information. To address this "global-to-local" attack challenge, we present a novel and general framework *CAMA* to generate adversarial examples via manipulating graph structure and node features. Specifically, we make use of Graph Class Activation Mapping and its variant to produce node-level importance corresponding to the graph classification task. Then through a heuristic design of algorithms, we can perform both feature and structure attacks under unnoticeable perturbation budgets with the help of both node-level and subgraph-level importance. Experiments towards attacking four state-of-the-art graph classification models on six real-world benchmarks verify the flexibility and effectiveness of our framework.

Index Terms—Adversarial Attack, Deep Graph Learning, Graph Neural Networks, Graph Classification.

1 Introduction

Graph structured data is ubiquitous for capturing relations and interactions at the level of node classification [1], edge prediction [2], and graph classification [3]. Among them, graph classification plays a vital role in a wide range of domains [4].

For instance, in social network analysis, the fake news detection problem can be regarded as a binary graph classification task over Twitter's news propagation networks [5]. As a powerful tool with the expressive capability of deep learning on graph data, the family of Graph Neural Networks (GNNs) has gained tremendous popularity over the past few years in graph classification and its downstream real-world applications [6]–[11].

Despite the powerful ability of GNNs in learning graph representations, their vulnerability to potentially existent adversarial examples on graph-structured data has been revealed recently [12]. Therefore, the lack of robustness within GNNs may be exploited by fraudsters or spammers, potentially provoking dissent on their applications in security-critical domains. For example, deliberately modifying personal identity information without authorization will result in credit card fraud [13]. Similar to the utilization of graph-structured data, adversarial attacks on graphs can also be broadly categorized into node level and graph level, in terms of the type of different tasks. On the one hand, studies towards node-level adversarial attacks are quite comprehensive from various perspectives [14]–[18]. On the other hand, in contrast to the remarkable and relatively mature frameworks for adversarial attacks on node-level tasks, systematic research regarding a general attacking framework for adversarial attacks on graph classification tasks is largely unexplored regardless of the vast importance.

Compared with perturbations for node-level classification, migrating these adversarial examples to graph-level

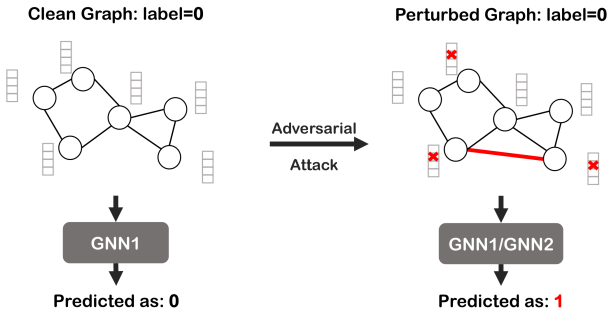


Fig. 1: **Adversarial attack on graph classification**. Given a cleaned graph, we can manipulate node features and edges to generate a poisoned graph to fool the victim GNN.

This work was supported by the National Key Research and Development Program of China No. 2020AAA0107800, National Natural Science Foundation of China (No. 62222209, 62250008, 62102222), Beijing National Research Center for Information Science and Technology under Grant No. BNR2023RC01003, BNR2023TD03006, and Beijing Key Lab of Networked Multimedia. (Corresponding authors: Xin Wang and Wenwu Zhu.)

- *Xin Wang and Wenwu Zhu are with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China (e-mail: xin_wang@tsinghua.edu.cn; wwzhu@tsinghua.edu.cn).*
- *Heng Chang, Beini Xie and Shiji Zhou are with the Tsinghua-Berkeley Shenzhen Institute, Tsinghua University, Shenzhen 518055, China (e-mail: changh17@mails.tsinghua.edu.cn; xbn20@mails.tsinghua.edu.cn; zsj17@mails.tsinghua.edu.cn).*
- *Tian Bian is with the System Engineering and System Management Department, Chinese University of Hong Kong, Hong Kong 999077, China (e-mail: tianbian@link.cuhk.edu.hk).*
- *Daixin Wang and Zhiqiang Zhang are with the Ant Group, Hangzhou 310063, China (e-mail: daixin.wdx@antgroup.com; lingyao.zzq@antgroup.com).*

- *Digital Object Identifier 10.1109/TKDE.2023.3313059*

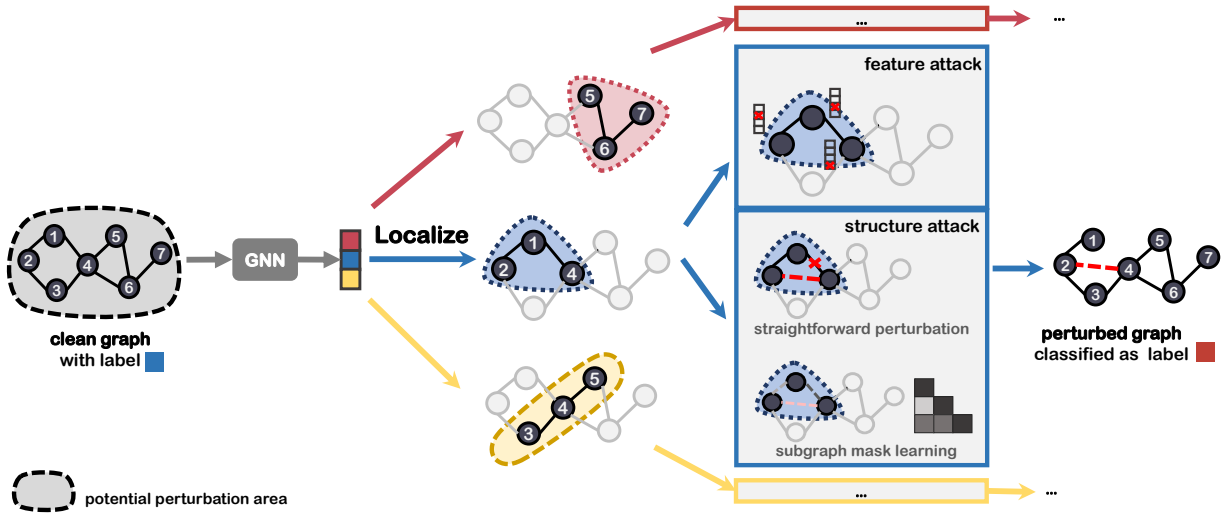


Fig. 2: **An example of *CAMA* for a three-classes graph classification task.** In the CAMA framework, we first **localize** potential perturbations to top-ranked important nodes. Then, we generate corresponding perturbed graphs until obtaining a successful feature/structure attack.

tasks is a non-trivial problem, since they have different goals for optimization from the local to global scale. We denote this problem as the "global-to-local attack challenge" which is illustrated in detail in Section 4.1. The desired attack framework for graph classification should be general to conduct both feature and structure attacks. Moreover, the effective attack on one graph classification model is expected to be able to be successfully transferred to other graph classifiers (an illustration is shown in Figure 1). Most importantly, a successful attacker is expected to perform unnoticeable attacks and effectively localize important nodes/edges to perturb via the global-level classification information. In a nutshell, the research on graph classification adversarial attacks still faces **three main challenges**:

- Given that graph classification tasks depend on efficiently learning global graph representation from local node embeddings via pooling functions, it is complex to exploit information of global-graph-level classification to generate local-node-level adversarial examples;
- Most existing approaches can only attack graph structure. However, node features might contain more fruitful information. For example, personal identity information and loan history apparently matter more in credit models and fraud detection. Therefore in a more realistic and practical condition, we need a general attack framework that is able to manipulate both node features and graph structure;
- Current attack methods for graph classification using gradient information only consider the training of target models and fail to reflect the information from the global graph structure, which might easily result in the generated adversarial edges being trapped around a single node or concentrating on high-degree nodes as we observe from experiments.

To tackle these challenges, we propose a novel hierarchical framework, namely ***CAMA***, to bridge the gap between local-node-level and global-graph-level information. We migrate the idea from **C**lass **A**ctivation **M**apping (CAM) [19] to conduct powerful adversarial **A**ttack towards graph classification tasks. This unified solution sheds light on the problem of quantifying the contribution of local node information to global representation in attacking graph classification tasks. An example of *CAMA* is shown in Figure 2. To summarize, our work makes the following **main contributions**,

- **Framework:** We propose the novel *CAMA* framework for adversarial attacks on graph classification. Our attack approach fills the gap in generating local perturbation examples from global graph classification as well as performs attacks unnoticeably. Given the simplicity and effectiveness, *CAMA* can serve as a strong benchmark for future works in this branch.
- **Algorithm:** We heuristically design novel algorithms to select target nodes in a graph by graph class activation mapping and its variant, then generate adversarial examples in the level of both structure and feature.
- **Experiment:** We show that our method is able to deteriorate graph classification performance by a significant margin on various benchmarks via targeting multiple state-of-the-art GNNs. Further, except for white-box attacks, we also test the transferability of our attack method under the black-box setting for evasion attacks.

2 Related Work

GNNs on graph classification. GNNs have proliferated in recent years for tasks like node classification, link prediction, graph classification, and graph generation. GNNs often stack multiple graph convolutions followed by a readout operation to aggregate nodes' information to a graph-level representation when dealing with graph classification tasks.

Various graph convolution layers and graph pooling operations are proposed to learn both nodes and graph representation better [20], [21]. One of the most popular GNNs is Graph Convolutional Networks (GCN) [1] which is inspired by the first-order approximation of Chebyshev

polynomials in ChebNet. It updates the node representation by taking an average representation of their one-hop neighbors. GCN has excellent results in the semi-supervised node classification tasks. Graph Isomorphism Network (GIN) [22] uses sum aggregation and multi-layer perceptrons instead of one single activation function. It has excellent discriminative power equal to that of the WL test. Facing the finite nature of recurrent GNNs, Implicit Graph Neural Network (IGNN) [23] is able to capture long-range dependencies and performs well in both graph classification and node classification on heterogeneous networks. Its framework ensures well-posedness based on Perron-Frobenius's theory.

Except for novel graph convolution operations, diverse pooling strategies affect graph tasks differently. Direct pooling methods like simple node pooling (node-wise mean-pooling, sum-pooling, and max-pooling) directly generate graph-level representation based on node representations [24]. In contrast, hierarchical graph pooling exploits the hierarchical graph structure. DiffPool [25] proposes a differentiable hierarchical clustering algorithm to learn representations of the new coarsened graph by training a soft cluster assign matrix in each layer. Based on the graph Fourier transform, EigenPooling [26] jointly uses node features and local structure. The graph pooling layer (gPool) [27] conducts down-sampling on graph data by selecting top-k nodes from calculated projection value. Inversely, the graph unpooling layer (gUnpool) does up-sampling to restore graphs to their original structure. Inspired by U-Net in computer vision, graph U-Nets (g-U-Nets) [27] is proposed using gPool and gUnpooling operations. g-U-Nets can encode and decode high-level features for network embedding.

In this paper, we use GCN, GIN, and IGNN as representatives of general graph classification neural networks and use g-U-Nets to represent hierarchical graph classification models.

Adversarial attacks on graph classification. GNNs have shown their vulnerability under adversarial attacks [28]. Most recent works aim to attack models on node classification tasks [16], [29]–[31]. Despite their fruitful progress, these methods can only perform attacks on node-level tasks.

For graph-level tasks, based on reinforcement learning, *RL-S2V* [15] flips edges by selecting two endpoints under black-box attack. *ReWatt* [32] proposes to perform unnoticeable attacks via rewiring operation and utilizes a similar reinforcement learning strategy as *RL-S2V*. *Grabnel* [33] exploits Bayesian optimization to conduct adversarial attacks targeting graph classification models. Under the white-box setting, *GradArgmax* [15] exploits gradients over the adjacency matrix of classification loss and flips edges with the largest absolute gradient. *Projective ranking* [34] generates adversarial examples by ranking potential edge perturbation masks through encoding node features and projecting selected edge masks.

Nevertheless, the above methods cannot perturb node features. Further, [35] proposes an attacking strategy on hierarchical graph pooling neural networks. However, they overlook the importance of direct pooling, like simple node pooling. Thus, this approach loses its strength when the graph classification model is unknown. A novel generic attack framework *GraphAttacker* is recently proposed by [36], which could attack multiple tasks. But the time complexity serves as its main concern due to the process of training the GAN-based model.

Considering all of these, adversarial attacks on graph classification are not been fully explored by previous studies. To mitigate this gap, our proposed general framework could flexibly perform structure attacks and feature attacks. Aside from the white-box attack, we also analyze the transferability of our method under black-box attacks.

CAM on graphs. Class Activation Mapping (CAM) localizes image-level classification into pixel-level image areas by using global average pooling (GAP) in convolutional neural networks in computer vision when it was firstly proposed [37]. CAM has a strong discriminative localization ability in the explanation of image classification. For example, it can localize the toothbrush region in a picture classified as brushing teeth. Compared with the blossom of grand application in computer vision, the utilization of CAM on graph-structured data (Graph CAM) is quite rare with only being applied to the explainability in GNNs [19], [38]. Given a graph classification task, Graph CAM can localize the most influential nodes for classification, which then helps us better understand GNNs. Grad Class Activation Mapping on graphs (Graph Grad CAM) [19] extends CAM on graphs by loosening architecture restrictions and using gradients of hidden layers as projection weights. In this work, we first integrate the localization ability of Graph CAM with the awareness of adversarial attacks on the graph classification tasks. We will undoubtedly increase the scope of research on Graph CAM.

3 PRELIMINARIES

3.1 Notations

Given a set of graphs $\mathcal{G} = \{G_i\}_{i=1}^{N}$, where $|\mathcal{G}| = N$, we consider graph classification on $\mathcal{G}$. Each graph $G_i = (\boldsymbol{A}_i, \boldsymbol{X}_i)$ has n_i nodes, where $\boldsymbol{A}_i \in \{0,1\}^{n_i \times n_i}$ is the adjacency matrix and $\boldsymbol{X}_i \in \mathbb{R}^{n_i \times D}$ is the node feature matrix with dimension D. Each G_i is assigned with a label $c_i \in \mathcal{C} = \{1, 2, ..., C\}$, where C is the total number of classes.

3.2 Graph Classification

Graph classification aims to predict the labels of unlabeled graphs. With paired graphs and labels $\{G_i, c_i\}_{i=\{1,\cdots,N\}}$, its goal is to learn a mapping function $f : \mathcal{G} \rightarrow \mathcal{C}$. We simplify graph classification model architecture and consider only one fully connected layer. Given a graph $G_i = (\boldsymbol{A}_i, \boldsymbol{X}_i)$ with n_i nodes, a standard procedure for graph classification with direct pooling can be formulated as:

$$\boldsymbol{h}_i^{(0)} = \boldsymbol{X}_i, \;\; \boldsymbol{h}_i^{(l)} = f_{conv}(\boldsymbol{h}_i^{(l-1)}; \boldsymbol{\Theta}^l), l = 1, 2, ..., L \quad (1)$$

$$\boldsymbol{h}_i = \text{pooling}(\boldsymbol{h}_i^{(L)}), \;\; \boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{b}, \quad (2)$$

where $\boldsymbol{h}_i^{(l)} \in \mathbb{R}^{n_i \times D_l}$ denotes the hidden node embedding in the l-th graph convolution f_{conv}, and $\boldsymbol{\Theta}^l$ is the corresponding parameter matrix. $\boldsymbol{h}_i \in \mathbb{R}^{D_L}$ is the graph embedding of G_i after pooling of final node embedding $\boldsymbol{h}_i^{(L)} \in \mathbb{R}^{n_i \times D_L}$. $\boldsymbol{W} \in \mathbb{R}^{C \times D_L}$ and $\boldsymbol{b} \in \mathbb{R}^C$ are parameters in the output fully connected layer, and L is the number of graph convolution.

The objective function for graph classification can be further formulated as:

$$\min_{\Theta} \mathcal{L}_{\Theta}(\mathcal{G}) = \sum_{i=1}^{N} l(f_{\Theta}(G_i), c_i),$$

where $l(\cdot, \cdot)$ is a loss function such as the cross-entropy.

3.3 Adversarial Attacks towards GNNs

The problem of adversarial attacks on graph classification is to misclassify graph labels, which is formulated as follows:

Problem 1. *Given paired data of graphs and their labels $\{G_i, c_i\}_{i=1}^{N}$, the goal of an attacker is to minimize the attack objective function $\mathcal{L}_{atk}$:*

$$\operatorname{argmin}_{\mathcal{G}'} \mathcal{L}_{atk}(\mathcal{G}') = \sum_{i=1}^{N} l_{atk}(f_{\Theta}(G_i'), c_i),$$

where l_{atk} is the attack loss function, and G_i' denotes the perturbed version of G_i.

We could define $l_{atk} = -l$ where l is set as the cross-entropy loss for graph classification. We can also define l_{atk} as the other attack loss like the CW-loss [29].

In the real world, the attacker usually only unnoticeably attacks within perturbation budget Δ for each graph G_i. Thus, the domain of modified graphs is constrained as :

$$\| \boldsymbol{A}_i' - \boldsymbol{A}_i \|_0 + \| \boldsymbol{X}_i' - \boldsymbol{X}_i \|_0 \leq \Delta,$$

where $\boldsymbol{A}_i'$ and $\boldsymbol{X}_i'$ is the perturbed adjacency matrix and node feature matrix for graph G_i'. In the following sections, we omit the subscript i for graph G_i for simplicity.

Adversarial attacks have various taxonomies from the perspectives of perturbation type (feature attack and structure attack), attacker's knowledge (white-box attack and black-box attack), and the stage where attacks happen (evasion attack and poisoning attack). A desired general framework should be able to deliberate most situations mentioned above, which is also the aim of this work.

4 Methodology

In this section, we start by introducing the global-to-local attack challenge. In order to tackle this challenge, we propose to first localize potential perturbations to top-important nodes and then perform attacks targeting these important nodes. The whole attack process is decomposed into two steps: 1) *node importance estimation*, and 2) *adversarial example generation*. In this way, we are able to first transfer the focus from classification on graphs to the contribution of each node and design perturbations locally afterward.

4.1 Global-to-Local Attack Challenge

Generating adversarial examples toward graph classification is intrinsically a global-to-local problem. The goal of attackers is to fool the GNNs from correct predictions on graph-level labels. However, the adversarial attacks must be localized to node and edge levels. The global-to-local problem is non-trivial to solve since graph-level predictions and node-level attacks are implicitly bridged via the pooling functions in GNNs. As empirical evidence, we observe that existing methods either give more attention to high-degree nodes or easily get trapped around one single node, which makes the adversarial examples noticeable and then undesirable.

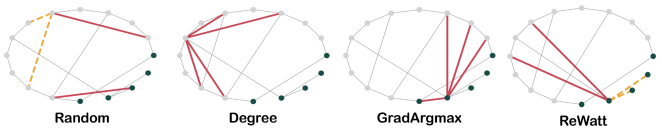


Fig. 3: **An example of structure attack on MUTAG dataset with edge attack proportion=**20%**.** Added edges are shown in red lines and deleted edges are shown in orange dashed lines.

TABLE 1: Average degree of the selected nodes in diverse feature attack methods.

Dataset	*Random*	*Degree*	*Betweenness*	*RWCS*	*GC_RWCS*
MUTAG	2.23	3.00	2.99	2.99	2.72
COX2	2.00	3.24	3.01	3.18	2.49

Table 1 shows the comparison of the average node degree of nodes selected by different attack methods. Compared with *Random* which selected nodes to attack randomly, the other attack methods tend to select nodes with higher degrees, which makes the attack process more noticeable.

In addition, we visualize selected important nodes and generated structural perturbations from various baselines in Figure 3. For example, for *Degree*, the perturbed edges are selected based on node degree. This may result in irrelevant perturbation between nodes and edges. Meanwhile, we can observe that the adversarial edges produced by *GradArgmax* and *ReWatt* are trapped near one single node, which further implies the deficiency of these two methods that extract merely local information.

As a result, this paper quantifies the contribution of nodes at the local level to graph classification tasks at the global level and reversely conducts effective adversarial attacks at the local level to destroy the global level classification performance. Our methods could better utilize the graph-level classification information to localize to nodes/edges not only important to the classification but also irrelevant to several super influential nodes.

4.2 Node Importance Estimation

The node importance estimation helps to localize potential perturbations into high-important nodes in a graph. Specifically, after finishing model training, we determine the contribution of nodes from the local level to graph classification in a way inspired by Graph CAM and its variant [19].

Graph CAM. As a useful method that provides explainability for graph classification, Graph CAM has been well studied. Since the weight matrix of the output fully connected layer can represent the importance of the features of each dimension for graph classification, Graph CAM builds a heat-map matrix by projecting back the weight matrix to the node representation in the final graph convolution layer to indicate the importance of each node for graph classification. This heat-map matrix is calculated as:

$$\boldsymbol{L}_{CAM} = \mathrm{ReLU}(\boldsymbol{h}^{(L)} \boldsymbol{W}^T), \tag{3}$$

where $\boldsymbol{W} \in \mathbb{R}^{C \times D_L}$ is the same weight matrix as in Eq. equation 2, and $\boldsymbol{h}^{(L)} \in \mathbb{R}^{n \times D_L}$ denotes node representation in the final graph convolution layer for one graph as shown in Eq. equation 1. The k-th element in c-th row of $\boldsymbol{W}$ indicates the importance of feature k for predicting label c.

A variant of Graph CAM, Graph Grad CAM, uses gradients with respect to each hidden convolutional layer and each class. Then, the calculation of gradients $\boldsymbol{\alpha}^l \in \mathbb{R}^{D^l \times C}$ replaces weights $\boldsymbol{W}^T$ in CAM to construct the heat-map matrix for each layer. At last, by taking the average over the heat-map matrix of all graph convolution layers, the heat-map matrix is calculated as:

$$\boldsymbol{\alpha}^l = \frac{1}{n}\sum_v \frac{\partial \boldsymbol{z}^T}{\partial \boldsymbol{h}_v^{(l)}}, \quad \boldsymbol{L}_{Grad-CAM} = \frac{1}{L}\sum_l \mathrm{ReLU}(\boldsymbol{h}^{(l)}\boldsymbol{\alpha}^l),$$

where $\boldsymbol{z} \in \mathbb{R}^C$ is the prediction logits, $\boldsymbol{h}_v^{(l)} \in \mathbb{R}^{D^l}$ is the hidden embedding for node v in the l-th graph convolution layer. The i-th entry in c-th column of $\boldsymbol{L}_{CAM}$ indicates the relative importance for node i resulting from classifying G_i into class c.

Though having great explainability, directly using Graph CAM still has two limitations. Firstly, the number of fully connected layers is fixed to one due to the restriction on matrix multiplication in Graph CAM. Secondly, the hidden size must be kept the same for all hidden convolutional layers for Graph Grad CAM. As we will show in experiments, these architecture restrictions do not deteriorate classification performance on clean graphs. Also, they do not hinder the transferability of our proposed attack methods.

Ranked CAM Matrix. We calculate the ranked CAM matrix based on the CAM heat-map matrix. The whole process is summarized in Algorithm 1. After getting the CAM heat-map matrix, We first rank each column in descending order and get the corresponding nodes ranking matrix $\boldsymbol{U}_{CAM}^{orig} \in \mathbb{R}^{n \times C}$ in line 1. This implies the class-specific view of node importance ranking. Then, we exploit $\boldsymbol{U}_{CAM}^{orig}$ to calculate a global-level nodes ranking vector $\boldsymbol{u}_{global} \in \mathbb{R}^n$ in line 2. Specifically, we go through each row in $\boldsymbol{U}_{CAM}^{orig}$ and use the highest ranking among all columns for each node until all nodes are included in $\boldsymbol{u}_{global}$. Finally, we concatenate these two ranking sources of nodes to get the final ranked CAM matrix $\boldsymbol{U}_{CAM} \in \mathbb{R}^{n \times (C+1)}$.

Algorithm 1: Generating ranked CAM matrix.

Input: Heat-map matrix $\boldsymbol{L}_{CAM}$.
Output: Ranked CAM matrix $\boldsymbol{U}_{CAM}$.
1 $\boldsymbol{U}_{CAM}^{orig} \leftarrow \mathrm{column\_rank}(\boldsymbol{L}_{CAM})$;
2 $\boldsymbol{u}_{global} \leftarrow \mathrm{global\_rank}(\boldsymbol{L}_{CAM})$;
3 $\boldsymbol{U}_{CAM} \leftarrow \mathrm{concatenate}([\boldsymbol{U}_{CAM}^{orig}, \boldsymbol{u}_{global}])$;
4 Return $\boldsymbol{U}_{CAM}$;

Because the CAM heat-map matrix can precisely demonstrate the importance of each node for graph classification tasks, after the ranking operation on CAM heat-map, each column in $\boldsymbol{U}_{CAM}$ indicates one type of view for nodes' importance ranking. We could identify the most influential nodes for the whole graph classification process through different views of the ranked CAM matrix and generate adversarial examples accordingly. Since the adversarial attack depends on Graph CAM, we name our framework as **CAM** based **A**ttack and its variant ***CAMA-Grad*** when using Graph Grad CAM.

4.3 Adversarial Example Generation

With access to the ranked CAM matrix $\boldsymbol{U}_{CAM}$, we call each column of $\boldsymbol{U}_{CAM}$ as the ranked CAM vector, denoted as $\boldsymbol{U}^c, c = 1, ..., C+1$. How do we generate adversarial examples with a series of ranked CAM vectors? Here, we heuristically propose two attack algorithms towards ***CAMA*** (for feature attack and structure attack) and ***CAMA-subgraph*** (for structure attack only). For *CAMA*, in the overall adversarial perturbation, we repeat using our algorithms for each column of the ranked CAM matrix $\boldsymbol{U}^c$ until a successful attack. For *CAMA-subgraph*, we only need the column of the predicted label in the ranked CAM matrix to select the candidate perturbations. Both two algorithms have their grad version *CAMA-Grad* and *CAMA-subgraph-Grad*. The difference between algorithms and their grad version lies only in calculating the CAM heat-map matrix.

4.3.1 Feature Attack

For feature perturbations, we set both global-level and local-level perturbation budgets. In global-level budgets, we assume only a few nodes of one particular graph are available. These nodes are called target nodes. In local-level budgets, we constrain the number of features to be adjusted.

Given the limitation of modified node amount r, target nodes are selected by the first r nodes in the ranked CAM vector $\boldsymbol{U}^c$. A small constant noise ϵ is added to each feature of target nodes for perturbation, while ϵ relies on the attacker's knowledge of node features. Specifically, given the information of the training process, the number of adjusted features K and adjusted magnitude λ, noise ϵ_j added for the j-th feature could be calculated following [31] as:

$$\epsilon_j = \begin{cases} \lambda \cdot \mathrm{sign}\left(\sum_{i=1}^n \frac{\partial l(f_\theta(G),c)}{\partial X_{ij}}\right), \\ \quad \text{if} j \in \text{argtop-K}\left(\left[\left|\sum_{i=1}^n \frac{\partial l(f_\theta(G),c)}{\partial X_{il}}\right|\right]_{l=1,2,...,D}\right) \\ 0, \text{ otherwise.} \end{cases}$$

We replace Carlili-Wagner loss in [31] with cross-entropy loss. The overall number of perturbations is $rK \leq \Delta$. We summarize the process of *CAMA* for generating feature perturbations in Algorithm 2.

Algorithm 2: *CAMA* for feature perturbations.

Input: Graph $G = (\boldsymbol{A}, \boldsymbol{X})$ with n nodes; number of nodes limit r; ranked nodes vector $\boldsymbol{U}^c$; feature noise ϵ_j, where $j = 1, 2, ..., D$.
Output: Modified feature matrix $\boldsymbol{X}$.
1 Initialize modified feature matrix $\boldsymbol{X}' \leftarrow \boldsymbol{X}$;
$C_{nodes} \leftarrow \boldsymbol{U}^c[:r]$;
2 **for** u *in* C_{nodes} **do**
3 $\quad \boldsymbol{X}'_j[u] \leftarrow \boldsymbol{X}_j[u] + \epsilon_j, \quad j = 1, 2, ..., D$
4 **return** $\boldsymbol{X}'$;

4.3.2 Structure Attack

Structure attack is more comprehensive compared with feature attack, considering the complexity of connectivity in graphs. To this end, we specially design two structure attack algorithms: *CAMA* and *CAMA-subgraph*, with the help of the ranked CAM matrix. *CAMA* is an efficient algorithm that performs attacks via simply flipping edges among top-ranked vital nodes in the ranked CAM matrix. *CAMA-subgraph* then takes a step further to attack by learning a subgraph mask to select edges for perturbation.

***CAMA*: straightforwardly flipping edges among most important nodes.** To generate structure perturbations, we assume edges among nodes of higher activation importance are more influential in graph classification tasks and intuitively flip edges among them. With the known ranked nodes' influence on graph classification, we flip edges among nodes that have a higher ranking. Furthermore, we exploit node similarity to enhance attack ability aside from the information from the graph structure. The similarity score is calculated as follows.

Given a learned embedding of node $\boldsymbol{h}^{emb}$, similarity $\boldsymbol{S}$ between nodes u and v is calculated with cosine distance:

$$\boldsymbol{S}[u,v] = \boldsymbol{S}[v,u] = \cos(\boldsymbol{h}_u^{emb}, \boldsymbol{h}_v^{emb}).$$

We constrain the operation of adding/deleting edges within the similarity constraint. Under the graph homophily assumption and with the calculated similarity matrix $\boldsymbol{S}$, we choose to add edges between low-similarity node pairs and delete edges between high-similarity node pairs:

$$\begin{cases} \boldsymbol{A}'[u,v] - \boldsymbol{A}[u,v] = 1, \boldsymbol{A}[u,v] = 0 \text{ and } \boldsymbol{S}[u,v] \leq s_1; \\ \boldsymbol{A}'[u,v] - \boldsymbol{A}[u,v] = -1, \boldsymbol{A}[u,v] = 1 \text{ and } \boldsymbol{S}[u,v] \geq s_2. \end{cases}$$

Our attacking strategy is in heuristic way by increasing the ranking number each time, iteratively finding candidate pairs of nodes, and flipping edges between new target nodes and old ones within perturbation budget and similarity restriction. In each iteration, we increase ranking number i by one and add a new node u_i, which ranked i-th in vector $\boldsymbol{U}^c$, into target nodes set C_{nodes}. In the end, we flip edges between new target nodes and old ones within Δ. The overall procedure for structure perturbations is summarized in Algorithm 3.

***CAMA-subgraph*: attack with subgraph mask learning.** In order to further exploit the local information from a subgraph perspective, we propose an end-to-end adversarial structure attack model with subgraph mask learning.

For each graph G, we obtain a subgraph G_{sub} by keeping $p\%$ top ranked nodes $\mathcal{V}_{sub}, |\mathcal{V}_{sub}| = \lfloor p\%|\mathcal{V}| \rfloor$ in the nodes rank vector with view of predicted label c (the c-th column $\boldsymbol{U}^c$ in the ranked CAM matrix). Then, we limit potential edge perturbations $\boldsymbol{M} = \mathcal{V}_{sub} \times \mathcal{V}_{sub}$ within the subgraph. With the edge perturbation candidates $\{m_{uv} | u, v \in \mathcal{V}_{sub}, \sum_{uv} m_{uv} \leq \Delta\}$, the adversarial examples are calculated as follows:

$$c_{uv} = 1 - 2a_{uv} \tag{4}$$

$$a'_{uv} = \begin{cases} a_{uv} + c_{uv}\sigma(m_{uv}), & u \in \mathcal{V}_{sub}, v \in \mathcal{V}_{sub} \\ a_{uv}, & others, \end{cases} \tag{5}$$

Algorithm 3: *CAMA* for structure perturbations.

Input: Graph $G = (\boldsymbol{A}, \boldsymbol{X})$ with n nodes; modification budget Δ; Similarity matrix $\boldsymbol{S}$; similarity restriction parameter s_1, s_2; ranked nodes vector $\boldsymbol{U}^c$.
Output: Modified adjacency matrix $\boldsymbol{A}'$.

```
1  Initialize remaining perturbation number
   n_perturbs ← Δ, modified adjacency matrix A' ← A,
   target nodes set C_nodes = U^c[0], and current rank
   index i = 1.
2  while (i ≤ n) and (n_perturbs > 0) do
3      u_i ← U^c[i];
4      for v in C_nodes do
5          if similarity_constraint((u_i, v); S, s_1, s_2) then
6              A'[u_i, v] ← 1 − A[u_i, v],
7              n_perturbs ← n_perturbs − 1;
8              if n_perturbs == 0 then
9                  break;
10     C_nodes ← [C_nodes, u_i];
11     i ← i + 1;
12 Return A';
```

where $\sigma(\cdot)$ is the sigmoid function to map mask values into zero and one. The larger value of m_{uv}, the more attack importance to perturb edge a_{uv}.

Given a trained victim model $f_{\boldsymbol{\Theta}}$, we minimize the attack loss l_{atk} for each graph with the victim model's parameters unchanged to learn the subgraph mask m_{uv}:

$$\min l_{atk} = l_{cw} + \lambda_{ent} l_{ent}, \tag{6}$$

where l_{cw} denotes for CW-loss, and l_{ent} represents the mean entropy of each element m_{uv}. l_{cw} aims to achieve a successful attack [29] while l_{ent} encourages the masking value of $\sigma(m_{uv})$ to be binary [39]. Hyper-parameter λ_{ent} balances the influence of l_{cw} and l_{ent} in the total loss function.

Specifically, given the ground truth label c_{yt} of the graph, the detailed designs of l_{cw}, l_{ent} are:

$$l_{cw} = \max(z_{c_{yt}} - \max_{c' \neq c_{yt}} z_{c'}, 0), \tag{7}$$

$$\begin{aligned} l_{ent} = -\frac{1}{|\boldsymbol{M}|} \sum_{u,v \in \mathcal{V}_{sub}} \big(& \sigma(m_{uv}) \log \sigma(m_{uv}) \\ & + (1 - \sigma(m_{uv})) \log(1 - \sigma(m_{uv})) \big), \end{aligned} \tag{8}$$

where the hyper-parameter η is the confidence size controlling how many entries in m_{uv} could be free of penalization.

Algorithm 4 shows the whole attacking process of structure attack with subgraph mask training, and we denote it as ***CAMA-subgraph***. First, we select top-ranked nodes in $\boldsymbol{U}^c$ to formulate a subgraph and limit the edge perturbation within the subgraph in line 1. Secondly, for each training epoch, we minimize the attack loss l_{atk} to train the subgraph mask $\boldsymbol{M}$ as shown in line 4. Then, we select the top-ranked mask $\boldsymbol{M}_\Delta$ within the perturbation budget Δ in line 6. In lines 7-9, we flip edges for nodes pair selected in $\boldsymbol{M}_\Delta$ to generate the adversarial example. Finally, we test the attack performance of generated adversarial examples in lines 11-12.

Algorithm 4: *CAMA-subgraph* for structure attack.

Input: Graph $G = (\boldsymbol{A}, \boldsymbol{X})$ with n nodes; the ground truth label c_{gt} of graph G; ranked nodes vector of the predicted label $\boldsymbol{U}^c$; subgraph proportion $p\%$; victim model $f_{\boldsymbol{\Theta}}$; total training epoch number T; the perturbation budget Δ;
Output: Modified Adjacency matrix $\boldsymbol{A}'$.

Initialize perturbation candidate subgraph $\mathcal{V}_{sub} = \{u | u \in U^c[:n_{sub}]\}$, where $n_{sub} = \lfloor p\% |\mathcal{V}| \rfloor$.
for t *in* $1, 2, ..., T$ **do**
  // Train subgraph mask
  $\min_{\boldsymbol{M}} l_{atk} = l_{cw} + \lambda_{ent} l_{ent}$;
  // Generate the adversarial example
  select top Δ perturbations $\boldsymbol{M}_\Delta \sim Bernoulli(\boldsymbol{M})$;
  for $(u, v) \in \{(u, v) | m_{uv} \in \boldsymbol{M}_\Delta\}$ **do**
    $a'_{uv} \leftarrow 1 - a_{uv}$;
  $G' \leftarrow (\boldsymbol{A'}, \boldsymbol{X})$;
  // Test the adversarial example
  if $\arg\max_c f_{\boldsymbol{\Theta}}(G') \neq c_{gt}$ **then**
    break;
return $\boldsymbol{A}'$;

4.4 Complexity Analysis

We analyze the complexity of the proposed framework by using *CAMA* as an example. Given a graph with n nodes as target, the main complexity lies in the preparation of inputs:

- The original nodes ranking matrix $\boldsymbol{U}^{orig}_{CAM}$ (**Algorithm 1**): The complexity of line 1 is $\mathcal{O}(Cn\log(n)) = \mathcal{O}(n\log(n))$, since the number of classes is always much less than that of nodes. Then the complexity from line 2 to 6 is $\mathcal{O}(Cn)$. Thus the total complexity of Algorithm 1 is $\mathcal{O}(n\log(n) + Cn)$;
- Feature noise ϵ_j, where $j = 1, 2, ..., D_L$: The complexity of getting all ϵ is $\mathcal{O}(nD_L + nK) = \mathcal{O}(nD_L)$, since K is seleted from D_L;
- Similarity matrix $\boldsymbol{S}$: The complexity of having similarity matrix is $\mathcal{O}(n^2 D_L)$.

Then we analyze the complexity of Algorithm 2 and Algorithm 3 accordingly, note that all constraints have no effects on the complexity since they can be checked in constant time:

Feature attack (Algorithm 2). The complexity from line 3 to line 5 is $\mathcal{O}(r)$. Thus, the total complexity of Algorithm 2 is combining it with $\boldsymbol{U}^c$ and all ϵ, which is $\mathcal{O}(n \times \max(D_L, \log(n)))$.

Structure attack (Algorithm 3). The complexity from line 2 to line 11 is $\mathcal{O}(\min(n^2, \Delta))$. Thus, combining with the complexity of similarity matrix $\boldsymbol{S}$, the total complexity of Algorithm 3 is $\mathcal{O}(\min(n^2 D_L, \Delta)) = \mathcal{O}(\Delta)$, since the modification budget Δ is controlled to restrict the access from attackers and strictly smaller than n^2.

Through our analysis of the complexity above, we can find that *CAMA* enjoys computational efficiency, especially in comparison with the complexity of target GNNs.

5 Experiments

In this section, we evaluate the effectiveness of our proposed methods on the graph classification task under the white-box and black-box settings. We further conduct sensitivity analysis for hyper-parameters and provide the poisoning black-box attack performance.

5.1 Experimental Setups

Datasets. We evaluate our attack strategies on five chemical graph classification benchmarks: MUTAG, PROTEINS, NCI1, COX2 [40], and three social network datasets: IMDB-BINARY, IMDB-MULTI, DBLP_v1. Among chemical graphs, node features consist of node attributes and node labels: in PROTEINS and COX2, we use both node labels and attributes, while in the others, we only use one-hot node labels as node features. For social networks, node features are initialized with the node degree. The dataset statistics can be found in Table 2.

TABLE 2: Dataset statistics.

Dataset	#Graphs	#Classes	Avg. #Nodes	Avg. #Edges
MUTAG	188	2	17.93	19.79
PROTEINS	1,113	2	39.06	72.82
NCI1	4,110	2	29.87	32.3
COX2	467	2	41.22	43.45
IMDB-BINARY	1,000	2	19.77	96.53
IMDB-MULTI	1,500	3	13.00	65.94
DBLP_v1	19,456	2	10.48	19.65

Graph Classifiers. We use four state-of-the-art GNNs for graph classification: GCN, GIN, IGNN, and g-U-Nets. Only one fully connected layer is adopted for all configurations, and no dropout layer is used after graph pooling. The same global sum-pooling readout function is applied for all models. For GCN, we use 5 GCN convolutional layers. For GIN, we set $\epsilon = 0$ (also called GIN-0) and use 5 GIN convolution layers. For IGNN, we use 3 IGNN convolution layers and tune hyper-parameter $\kappa \in \{0.7, 0.98\}$. We fix the size of hidden dimensions as 64. g-U-Nets have a different architecture due to their hierarchical nature. Here, we use the node representation of the last layer before the readout function to calculate the CAM heat-map matrix. We apply four (graph pooling) gPool layers with 90%, 70%, 60%, and 50% node proportions and ignore the max-pooling layer in its readout function since global max-pooling is poorer at localization compared to GAP [37]. We implement these GNNs with Pytorch Geometric (PyG)[1].

Baselines. We compare our methods with representative feature attack baselines which select perturbation nodes from various perspectives (*Random*, *Degree*, *Betweenness*, *RWCS*, etc.). For all baselines under feature attacks, the same feature noises in Section 4.3.1 are added to selected nodes, the difference only lies in the nodes selected process. We also compare *CAMA* with representative white-box (*PGD*, *PR-BCD*) and black-box (*ReWatt*, *Grabnel*) structure attack methods. Every baseline we compared either released source code or made it available upon request. The detailed baselines are described as follows.

- (structure/feature) *Random* [15]: *Random* randomly selects nodes to perturb and edges to insert/delete.
- (structure/feature) *Degree* [41]: *Degree* chooses nodes with top degrees and insert/delete edges among them.
- (structure) *GradArgmax* [15]: *GradArgmax* greedily selects perturbation edges by gradients of each pair of nodes, which works only for structure attack.

1. https://github.com/rusty1s/pytorch_geometric

TABLE 3: Summary of the change in classification accuracy (in %) compared to the clean graph under **white-box attack** for **chemical datasets**. Lower is better. Best performances are shown in **bold** markers.

Dataset	**MUTAG**				**PROTEINS**				**NCI1**				**COX2**			
Models	GCN	GIN-0	IGNN	g-U-Nets	GCN	GIN-0	IGNN	g-U-Nets	GCN	GIN-0	IGNN	g-U-Nets	GCN	GIN-0	IGNN	g-U-Nets
Clean	83.04	89.85	81.46	88.89	78.17	77.81	77.99	77.54	78.98	77.59	75.06	72.24	88.87	83.51	83.51	83.08
Feature Attack																
Random[15]	-5.35	-5.29	-7.43	-7.02	-1.26	-0.63	-4.04	-0.90	-16.06	-19.03	-33.92	-55.57	-17.32	-3.42	-7.05	-11.54
Degree[41]	-4.82	-7.40	-7.43	-8.66	-1.61	-0.81	-4.58	-0.99	-17.27	-23.63	-37.40	-59.37	-22.67	-4.28	-8.54	-13.90
PageRank [31]	-4.30	-3.18	-6.37	-6.99	-1.53	-0.72	-4.67	-0.99	-18.96	-21.99	-37.83	-59.95	-25.48	-6.00	-10.28	-12.40
Betweenness [31]	-5.88	-8.97	-6.90	-8.13	-1.44	-0.72	-4.49	-0.90	-15.70	-19.17	-34.87	-57.64	-17.10	-4.08	-7.70	-13.91
GC-RWCS [31]	-6.43	-7.92	-6.90	-8.13	-1.53	-0.81	-4.13	-0.99	-16.64	-22.58	-33.72	-57.40	-20.10	-3.88	-7.91	-13.70
RWCS [31]	-5.35	-7.92	-6.90	-7.57	-1.71	-0.63	-4.58	-0.99	-17.52	-24.21	-35.47	-58.74	-23.75	-5.99	-8.78	-13.06
CAMA	-10.64	**-9.53**	**-10.12**	**-11.78**	-2.24	-1.44	**-6.56**	**-2.25**	**-33.58**	**-36.08**	-56.74	-69.61	**-52.68**	-9.69	**-27.83**	**-27.64**
CAMA-Grad	**-11.70**	**-9.53**	**-10.12**	-11.73	**-2.60**	**-1.53**	-6.29	**-2.25**	-31.70	-35.57	**-56.76**	**-69.90**	-52.23	**-15.47**	-22.89	-24.40
Structure Attack																
Random [15]	-4.82	-16.43	-5.26	-2.13	-0.99	-4.13	-1.53	-0.54	-9.49	-10.97	-6.37	-4.31	-6.43	-3.84	-2.14	-4.93
Degree [41]	8.48	-16.43	-7.92	-3.27	-0.72	-6.91	-1.53	-0.09	-8.08	-15.13	-5.79	-4.31	-6.87	-9.83	-4.07	-5.56
GradArgmax[15]	-7.98	-43.33	-7.37	-2.13	-1.88	-7.63	-2.96	-1.08	-10.90	-12.31	-10.85	-7.45	-17.17	-16.24	-13.08	-11.99
PR-BCD [42]	-17.54	-55.76	**-19.68**	-6.99	-4.85	-33.25	-3.42	-3.78	-47.84	-19.85	-46.50	-23.36	-55.22	-52.55	-30.80	-32.11
CAMA	-11.08	-47.07	-11.64	-9.18	-3.23	-9.44	-2.88	-1.80	-20.68	-22.43	-15.74	-9.88	-22.48	-18.89	-13.93	-12.64
CAMA-Grad	-11.64	-50.20	-12.72	-5.85	-2.78	-9.16	-3.24	-1.53	-23.51	-22.29	-16.69	-8.76	-24.86	-18.85	-13.28	-15.82
CAMA-subgraph	**-25.44**	-74.44	-18.62	-7.49	**-6.91**	**-33.43**	**-6.02**	-3.69	**-61.44**	-54.40	**-49.68**	**-23.77**	**-57.85**	**-58.43**	**-34.88**	-33.82
CAMA-subgraph-Grad	-23.86	**-75.55**	**-19.68**	**-10.24**	-5.84	-32.81	-5.93	**-3.87**	-61.24	**-55.67**	-48.98	-21.92	-54.41	**-58.43**	-32.29	**-35.32**

TABLE 4: Summary of the change in classification accuracy (in %) compared to the clean graph under **white-box attack for social networks**. Lower is better. Best performances are shown in **bold** markers.

Dataset	**IMDB-BINARY**			**IMDB-MULTI**			**DBLP_v1**		
Models	GCN	GIN	g-U-Nets	GCN	GIN	g-U-Nets	GCN	GIN	g-U-Nets
Clean	73.67	74.22	73.89	50.00	50.59	48.30	90.47	91.52	93.55
Random [15]	-0.78	-7.55	-0.78	-0.96	-9.48	-0.45	-0.37	-3.21	-0.32
Degree [41]	-1.67	-18.00	-2.78	-1.63	-14.37	-2.37	-0.37	-4.02	-0.31
GradArgmax [15]	-4.34	-19.22	-3.33	-2.82	-14.52	-1.11	-0.32	-4.29	-0.51
PGD [15]	-2.57	-22.82	-1.79	-2.00	-29.12	-0.57	-1.33	-11.24	-1.10
PR-BCD [42]	-5.87	-17.42	-7.99	-4.80	-20.79	-2.50	-0.90	-8.40	-0.78
ReWatt [32]	-6.07	-3.22	-6.99	-6.53	-2.79	-3.03	-1.50	-2.18	**-2.08**
CAMA	-2.11	-15.22	-2.33	-3.11	-11.48	-1.19	-0.72	-4.42	-0.50
CAMA-Grad	-2.78	-15.55	-1.56	-3.26	-13.33	-1.04	-0.80	-4.98	-0.60
CAMA-subgraph	**-7.77**	-21.52	-8.59	-7.73	**-30.39**	**-4.10**	-1.44	**-11.54**	-1.18
CAMA-subgraph-Grad	-7.37	**-22.82**	**-8.79**	**-8.07**	-30.19	-3.97	**-1.52**	-10.89	-1.13

TABLE 5: Summary of the change in classification accuracy (in %) compared to the clean graph under **black-box attack**. Lower is better.

Dataset	**MUTAG**			**PROTEINS**			**NCI1**			**COX2**		
Models	GIN-0	IGNN	g-U-Nets	GIN-0	IGNN	g-U-Nets	GIN-0	IGNN	g-U-Nets	GIN-0	IGNN	g-U-Nets
Clean	89.85	81.46	88.89	77.81	77.99	77.54	77.59	75.06	72.24	83.51	83.51	83.08
Feature Attack												
Random [15]	-2.13	-4.24	-4.85	-0.54	-3.59	-0.45	-6.59	-9.32	-13.14	-2.56	-7.26	-9.19
Degree [41]	-2.66	-4.24	**-6.52**	-0.63	-4.13	-0.54	-9.46	-10.19	-14.89	-3.85	-7.69	-10.69
PageRank [31]	-2.66	-3.71	-4.85	-0.45	-3.96	-0.45	-9.10	-12.82	-15.26	-5.57	-8.33	-10.06
Betweenness [31]	-3.71	-3.71	**-6.52**	-0.45	-3.87	-0.36	-7.57	-12.14	-13.97	-3.64	-6.83	-11.55
GC-RWCS [31]	-2.66	-3.71	**-6.52**	-0.54	-3.60	-0.45	-7.47	-11.51	-13.82	-3.88	-6.84	-9.83
RWCS [31]	-2.66	-3.71	-5.97	-0.45	-3.96	-0.45	-9.29	-11.36	-14.16	-5.99	-6.84	-10.27
CAMA	**-4.24**	-6.93	-5.38	-0.90	-5.57	**-1.26**	**-17.13**	**-23.72**	**-24.28**	**-12.88**	**-22.04**	**-22.26**
CAMA-Grad	-3.71	**-8.01**	-4.27	**-0.99**	**-5.84**	-1.17	-15.23	-22.65	-24.06	-12.44	-19.03	-20.35
Structure Attack												
Random [15]	-16.43	-5.26	-2.13	-4.13	-1.53	-0.54	-10.97	-6.37	-4.31	-3.84	-2.14	-4.93
Degree [41]	-16.43	-7.92	-3.27	-6.91	-1.53	-0.09	-15.13	-5.79	-4.31	-9.83	-4.07	-5.56
GradArgmax [15]	-12.75	-9.53	-2.72	-5.48	-1.17	-0.90	-8.88	-6.67	-3.75	-9.82	-4.48	-5.12
ReWatt [32]	-6.84	-3.68	-9.12	-2.61	-0.81	**-1.17**	-7.57	-4.94	-8.23	-7.70	-2.57	-13.07
Grabnel [33]	-42.39	-11.11	-2.66	-8.53	-2.25	-0.90	-26.32	-15.55	-5.52	-11.13	-6.41	-12.24
CAMA	-47.07	-11.64	**-9.18**	-9.44	-2.88	-1.35	-22.43	-15.74	**-9.88**	-18.89	**-13.93**	-12.64
CAMA-Grad	-50.20	**-12.72**	-5.85	-9.16	**-3.24**	-1.08	-22.29	-16.69	-8.76	-18.85	-13.28	**-15.82**
CAMA-subgraph	-59.53	-11.69	-3.77	**-24.89**	-2.25	-0.99	-25.84	-15.81	-7.52	**-53.73**	-12.20	-11.57
CAMA-subgraph-Grad	**-60.03**	-10.64	-5.44	-23.63	-2.61	**-1.17**	**-26.69**	**-17.42**	-9.32	-53.51	-13.68	-8.56

- (structure) *PGD* [14]: *PGD* performs project gradient descent topology attacks and is an effective white-box attack algorithm.
- (structure) *PR-BCD* [42]: *PR-BCD* conducts sparsity-aware first-order optimization attacks based on randomized block coordinate descent and is able to attack larger graphs.
- (structure) *ReWatt* [32]: *ReWatt* conducts rewiring opera-

tions to perform structure attacks and uses reinforcement learning to find the optimal rewiring operations. We select *ReWatt* as the representative of the state-of-the-art black-box optimization baseline.
- (structure) *GRABNEL* [33]: *GRABNEL* is a powerful black-box attack method on graph classification tasks based on the bayesian optimization.
- (feature) *PageRank* [31]: *PageRank* is a graph centrality metric. Here, we attack nodes with the top-ranked Pagerank scores.
- (feature) *Betweenness* [31]: *Betweenness* is a graph centrality metric. Here, we attack nodes with the top-ranked Betweenness scores.
- (feature) *RWCS* [31]: *RWCS* is a practical feature attack algorithm based on an importance score similar to PageRank by using the connection between the GNNs' backward propagation and random walks.
- (feature) *GC-RWCS* [31]: *GC-RWCS* is a variant of *RWCS*, which uses the greedy correction procedure on top of the RWCS strategy.
- (feature, structure) *GraphAttacker* [36]: GraphAttacker performs attacks based on the generative adversarial network and three key components: the multi-strategy attack generator, the similarity discriminator, and the attack discriminator.
- (structure) *Projective Ranking* [34]: *Projective Ranking* exploits mutual information to consider the long-term benefits of perturbations and generates adversarial samples.
- (feature, structure) *Attack on the HGP Model* [35]: *Attack on the HGP Model* aims to fool the pooling operator in hierarchical GNN-based graph classification models.

Perturbation restrictions and hyper-parameters. For feature attack, we set feature adjusted magnitude $\lambda = 0.1$. We select 10% of nodes in one graph to perturb, and 10% of features are modified for each dataset. For structure attack, we set the perturbation budget $\Delta = \lceil 10\%|E_i| \rceil$ for each graph G_i, where $|E_i|$ denotes the number of edges in graph G_i. For *ReWatt*, the number of rewiring operations is set to $\lfloor 0.5\Delta \rfloor$ with at least one rewiring, which is kept the same setting as [32]. Besides, in the similarity restriction, we use the first hidden layer to calculate nodes similarity $\boldsymbol{h}^{emb} = \boldsymbol{h}^{(1)}$, fix $s_2 = 0.95$ and tune $s_1 \in \{0.95, 0.9, 1\}$. For *CAMA-subgraph*, we set total training epochs as 30, the subgraph graph proportion $p\% = 50\%$, $\lambda_{ent} = 1$.

We conduct the untargeted attack and evaluate them on test graphs. Specifically, we perform 10-fold cross-validation in each classification process and report the average validation accuracy within the cross-validation. This configuration follows [22] on graph classification, resulting from the unstable training of small-sized datasets such as MUTAG.

5.2 Adversarial Attack on Graph Classification

We first compare *CAMA* and *CAMA-subgraph* to multiple baselines under the white-box attack. We train on clean graphs for each graph classifier, generate perturbed graphs on validation sets, and calculate prediction accuracy using the trained graph classifiers. Full results under the white-box setting for chemical datasets are provided in Table 3, for social networks are demonstrated in Table 4.

In feature attack, our proposed methods perform better by a high margin on all datasets and all graph classification models, which implies our methods can select the most influential nodes for graph classification tasks. In structure attack, *CAMA* and *CAMA-subgraph* outperform the other baselines in all situations. Meanwhile, the subgraph mask training algorithm (*CAMA-subgraph*) outperforms the simple heuristic flip edge method (*CAMA*) by a large margin. Actually, the choice of *CAMA* and *CAMA-subgraph* is to balance the attack efficiency and effectiveness. These results demonstrate the high attack effectiveness of *CAMA*. More interestingly, the grad version *CAMA-Grad* achieves excellent performance close to *CAMA* but does not guarantee better performance.

We also observe that the attack results vary from different datasets and graph classifiers. The accuracy decreases the least on the PROTEINS dataset when suffering attacks. Interestingly, graph classifiers tend to behave differently when they are attacked by structure and feature perturbations. For example, IGNN is more robust facing structural perturbations while more vulnerable under feature attack.

5.3 Transferability of Attack

In real-world applications, model parameters usually are not available. Thus, to evaluate *CAMA* under a more realistic and general situation and further explore the transferability of various attacking methods, we validate our attack strategies under the black-box attack setting for four datasets. Specifically, we use GCN as the surrogate model, generate adversarial examples by targeting GCN, and then evaluate the other GNNs on the perturbed graphs. The detailed results are provided in Table 5.

First, we could see our approaches surpass the other baselines in most situations. The perturbations generated by *CAMA* and *CAMA-subgraph* consistently demonstrate strong transferability on four graph classification datasets under the black-box attack setting. For *CAMA-subgraph*, we could also see a significant performance improvement of *CAMA-subgraph* over *CAMA*. The calculation of the ranked CAM matrix and the selected subgraph is important. As a result, the attack performance of a black-box attack may exceed a white-box attack due to an efficient ranked CAM matrix of the GCN surrogate model. Moreover, we could see that the attack performance of *ReWatt* is unstable. It does work with some datasets, like NCI1, while it fails for the other datasets. Second, compared with the white-box attack, our approaches have a more significant advantage over baselines like *GradArgmax*. This indicates that our methods have a more vital attack ability when transferring to other GNNs. Besides, the results show that perturbations against a surrogate model with typical architecture could also generalize to the hierarchical graph classifier like g-U-Nets.

5.4 Localization Effectiveness

Here, we show the effectiveness of *CAMA* in the global-to-local attack challenge raised in Sec. 4.1. For the feature attack, we show the average node degree selected by *CAMA* and other baselines in Table 6. In comparison to baselines such as *RWCS*, the average degree of nodes selected by *CAMA* and *CAMA_Grad* are lower and closer to that of *Random*, which

TABLE 6: Comparison of average degrees for selected nodes under various feature attack methods.

Dataset	*Random*	*RWCS*	*GC_RWCS*	***CAMA***	***CAMA-Grad***
MUTAG	2.23	2.99	2.72	2.12	2.28
COX2	2.00	3.18	2.49	2.26	2.35
PROTEINS	3.74	4.73	4.13	4.11	3.96
NCI1	2.14	2.96	2.64	1.92	1.91

indicates our proposed methods are more unnoticeable. What's more, we visualize the perturbation nodes and edges selected by *CAMA* in Figure 4 as a comparison with Figure 3. The edges chosen by *CAMA* are not trapped in one node compared to *Degree*, *GradArgmax*, and *ReWatt*.

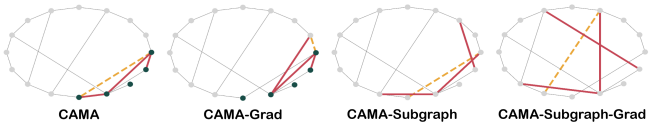


Fig. 4: **An example of *CAMA* on MUTAG dataset with edge attack proportion=**20%**.** Green nodes are selected by *CAMA*, indicating their strong influences on graph classification. Added edges are shown in red lines and deleted edges are shown in orange dashed lines.

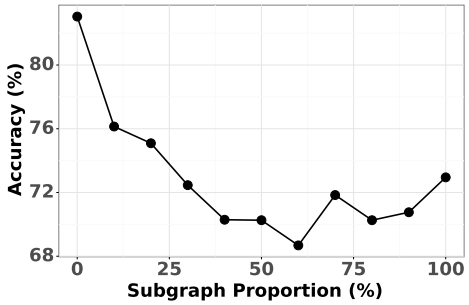


Fig. 5: Line-plot for attack performance under *CAMA-subgraph* for different subgraph proportion. We record 10-fold testing results on the MUTAG dataset using GCN as the graph classifier. Lower is better.

Insights of target nodes chosen by *CAMA*. We compare the top 5 nodes selected by *CAMA* and *Degree* and report statistics in Table 7. The relatively small average degree and closeness centrality value differentiate *CAMA* from centrality-based methods. Through the total variation and number of edges, we find that nodes chosen by *CAMA* have higher connectivity and smoothness (smaller total variation). Besides, we provide an example of edge perturbations on baselines in Figure 3.

TABLE 7: Statistics for selected nodes by *CAMA* and *Degree*.

Method	Avg. Degree	Avg. Closeness	Total Variation	No. Edges
Degree	2.8	0.25	12	1
CAMA	2.4	0.22	8	2

5.5 Sensitivity Analysis

Sensitivity analysis for subgraph proportion p in *CAMA-subgraph*. The choice of subgraph proportion in *CAMA-subgraph* is crucial. A larger proportion means more perturbation candidates but also more noise, while a smaller proportion may face perturbation candidates deficiency. An efficient subgraph selection could help the attacker localize the essential subgraph nodes and edges. Figure 5 shows the attack performance of *CAMA-subgraph* with various subgraph proportions on MUTAG. We could see a clear drop tendency when the subgraph proportion gets smaller from 100%, which indicates the effectiveness of locating the subgraph with the ranked CAM vector. For MUTAG, the best proportion is 60%, and the accuracy drop is 14.36% under this structure attack perturbation setting.

Sensitivity Analysis for Hyper-parameter s_1 and s_2 in *CAMA*. We perform a sensitivity analysis over s_1 and s_2 in Table 8 and set GIN as the victim model on the MUTAG dataset. s_1 controls the edge insertion and s_2 controls the edge deletion. $s_1 = 1, s_2 = 0$ represents no restriction on edge insertion/deletion. We could find that controlling the edge insertion is more helpful for successful attacks in contrast to edge deletion.

TABLE 8: Sensitivity analysis for hyper-parameter s_1 and s_2. Lower is better.

Hyper-parameter	clean	0	0.2	0.4	0.6	0.8	1
s_1 (fix s_2=0)	83.04	67.72	67.72	66.64	67.16	66.64	71.40
s_2 (fix s_1=1)	83.04	71.40	71.40	71.93	71.93	71.93	72.46

Perturbations budget for white-box attack. We analyze the changes in accuracy with respect to the perturbation budget Δ and the adjusted magnitude λ in Figure 6. Not surprisingly, the prediction accuracy decreases with a higher number of perturbations or larger values of adjusted magnitude. In all settings of hyper-parameters, we can observe that *CAMA* and *CAMA-Grad* show remarkable advantages over all the other baselines. Meanwhile, from the figure on the right, we can observe the accuracy drops dramatically when the adjusted magnitude λ gets larger for *CAMA* and *CAMA-Grad*.

5.6 Poisoning Black-box Attack

We also evaluate our methods under poisoning black-box attacks. We select GIN as the victim model and retrain it on perturbed graphs generated from the surrogate GCN. Additionally, we compare *CAMA* with a more powerful attacker, project gradient descent topology attack (*PGD*) [14]. *PGD* was originally designed for node classification tasks. We extend its application domain to graph classification. We use cross entropy loss and fix epoch numbers to 10 in our experiment under *PGD* topology attack. Figure 7 shows the final attack results. Coordinating with the results of the evasion attack above, the strong transferability of *CAMA* and *CAMA-Grad* still concludes. However, the method using purely gradient information like *GradArgmax* and *PGD* may damage the attacking performance when transferring to other models.

5.7 Computational Efficiency Analysis

To cooperate with our complexity analysis, we demonstrate the computational efficiency of *CAMA* and *CAMA-subgraph* by reporting the average running time over 10 times in

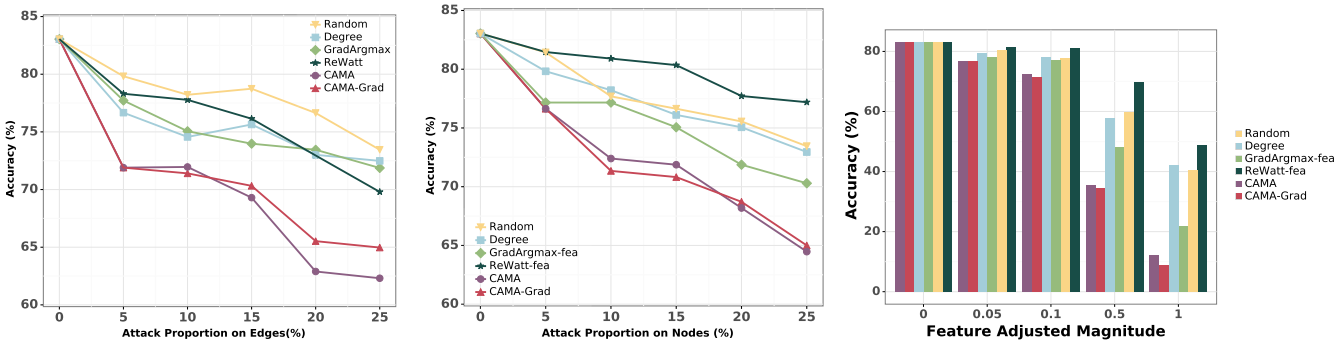


Fig. 6: **Attack results with different perturbation hyper-parameters.** All experiments are conducted on the MUTAG dataset using GCN. Lower accuracy is better. Left: Attack results with increasing perturbation proportion of edges. Middle: attack results with increasing perturbation proportion of nodes. Right: Attack results with increasing adjusted magnitude values.

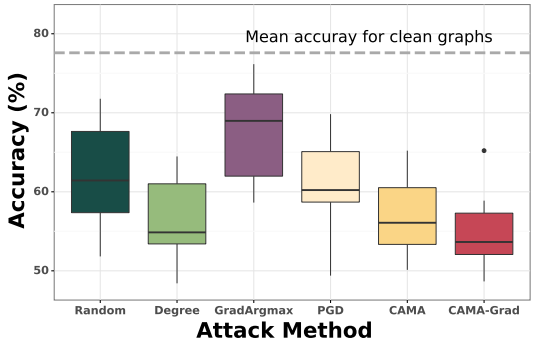


Fig. 7: Box-plot for **poisoning** structural perturbations under black-box attack. We use GIN as the victim model and record 10-fold testing results on the NCI1 dataset. Lower is better.

comparison with all baselines. For structure attack, *Random* runs in 0.37 seconds, *GradArgmax* runs in 0.24 seconds, *ReWatt* finishes for 30.01 seconds, while *CAMA* runs in 0.79 seconds and *CAMA-Subgraph* runs in 34.23 seconds. Under the feature attack setting, *Random* finishes in 0.63 seconds, *Degree* runs in 0.61 seconds, and *CAMA* finishes in 0.67 seconds.

We can find that *CAMA* can finish within 1 second, which is consistent with our complexity analysis and implies that the scalability of our proposed approaches could not be an issue. The time cost of *CAMA-subgraph* is comparable with *ReWatt*. Both methods need end-to-end training, but *CAMA-subgraph* has better attack performance.

6 Conclusion

We revisit adversarial attacks on GNNs for graph classification in this paper. We establish a general attack framework focusing on graph classification which considers comprehensive attack settings under white-box and black-box attacks and performs both structure and feature attacks. We first estimate the importance of nodes towards the graph classification by Class Activation Mapping and its variant. Then, we heuristically design algorithms to generate adversarial examples for both feature and structure attacks with the ranking information of nodes. Experiments show that the proposed attack strategies significantly outperform existing approaches on various graph classifiers under multiple settings. Our general framework can also serve as a simple yet novel baseline for future works in evaluating the robustness of graph classification tasks.

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *international conference on learning representations*, 2017.

[2] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 5171–5181.

[3] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *international conference on machine learning*, pp. 1263–1272, 2017.

[4] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[5] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake news detection on social media using geometric deep learning," *arXiv preprint arXiv:1902.06673*, 2019.

[6] L. G. Gómez, B. Chiem, and J.-C. Delvenne, "Dynamics based features for graph classification," *arXiv preprint arXiv:1705.10817*, 2017.

[7] R. Kim, C. H. So, M. Jeong, S. Lee, J. Kim, and J. Kang, "Hats: A hierarchical graph attention network for stock movement prediction," *arXiv preprint arXiv:1908.07999*, 2019.

[8] T. Magelinski, D. Beskow, and K. M. Carley, "Graph-hist: Graph classification from latent feature histograms with application to bot detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5134–5141.

[9] Z. Li, J. Tang, and T. Mei, "Deep collaborative embedding for social image understanding," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 9, pp. 2070–2083, 2018.

[10] Z. Li, J. Tang, L. Zhang, and J. Yang, "Weakly-supervised semantic guided hashing for social image retrieval," *International Journal of Computer Vision*, vol. 128, pp. 2265–2278, 2020.

[11] Z. Li, Y. Sun, L. Zhang, and J. Tang, "Ctnet: Context-based tandem network for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, pp. 9904–9917, 2021.

[12] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 66–74.

[13] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, "Data mining for credit card fraud: A comparative study," *Decision Support Systems*, vol. 50, no. 3, pp. 602–613, 2011, On quantitative methods for detection of financial fraud, ISSN: 0167-9236.

[14] K. Xu, H. Chen, S. Liu, *et al.*, "Topology attack and defense for graph neural networks: An optimization perspective," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[15] H. Dai, H. Li, T. Tian, *et al.*, "Adversarial attack on graph structured data," in *International conference on machine learning*, PMLR, 2018, pp. 1115–1124.
[16] H. Chang, Y. Rong, T. Xu, *et al.*, "A restricted black-box adversarial framework towards attacking graph embedding models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3389–3396.
[17] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples for graph data: Deep insights into attack and defense," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, AAAI Press, 2019, pp. 4816–4823.
[18] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *Proceedings of the 36th International Conference on Machine Learning, ICML*, ser. Proceedings of Machine Learning Research, PMLR, 2019.
[19] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, "Explainability methods for graph convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 772–10 781.
[20] H. Chang, Y. Rong, T. Xu, *et al.*, "Spectral graph attention network with fast eigen-approximation," *arXiv preprint arXiv:2003.07450*, 2020.
[21] C. Guan, Z. Zhang, H. Li, *et al.*, "Autogl: A library for automated graph learning," *arXiv preprint arXiv:2104.04987*, 2021.
[22] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks," in *ICLR 2019 : 7th International Conference on Learning Representations*, 2019.
[23] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui, "Implicit graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 11 984–11 995.
[24] J. Zhou, G. Cui, S. Hu, *et al.*, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.
[25] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *neural information processing systems*, pp. 4801–4811, 2018.
[26] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 723–731.
[27] H. Gao and S. Ji, "Graph u-nets," in *International Conference on Machine Learning*, 2019, pp. 2083–2092.
[28] W. Jin, Y. Li, H. Xu, Y. Wang, and J. Tang, "Adversarial attacks and defenses on graphs: A review and empirical study," *arXiv preprint arXiv:2003.00653*, 2020.
[29] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2847–2856.
[30] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *International Conference on Learning Representations (ICLR)*, 2019.
[31] J. Ma, S. Ding, and Q. Mei, "Towards more practical adversarial attacks on graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.
[32] Y. Ma, S. Wang, T. Derr, L. Wu, and J. Tang, "Graph adversarial attack via rewiring," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21, 2021, 1161–1169.
[33] X. Wan, H. Kenlay, B. Ru, A. Blaas, M. Osborne, and X. Dong, "Adversarial attacks on graph classifiers via bayesian optimisation," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.
[34] H. Zhang, B. Wu, X. Yang, *et al.*, "Projective ranking: A transferable evasion attack method on graph neural networks," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021.
[35] H. Tang, G. Ma, Y. Chen, *et al.*, "Adversarial attack on hierarchical graph pooling neural networks," *arXiv preprint arXiv:2005.11560*, 2020.
[36] J. Chen, D. Zhang, Z. Ming, and K. Huang, *Graphattacker: A general multi-task graphattack framework*, 2021. arXiv: 2101.06855 [cs.LG].
[37] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.
[38] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," *arXiv preprint arXiv:2012.15445*, 2020.
[39] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, *Gnnexplainer: Generating explanations for graph neural networks*, 2019. arXiv: 1903.03894 [cs.LG].
[40] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," in *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. arXiv: 2007.08663. [Online]. Available: www.graphlearning.io.
[41] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos, "Gelling, and melting, large graphs by edge manipulation," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 245–254.
[42] S. Geisler, T. Schmidt, H. Şirin, D. Zügner, A. Bojchevski, and S. Günnemann, "Robustness of graph neural networks at scale," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 7637–7649.

Xin Wang is currently an Assistant Professor at the Department of Computer Science and Technology, Tsinghua University. He got both his Ph.D. and B.E degrees in Computer Science and Technology from Zhejiang University, China. He also holds a Ph.D. degree in Computing Science from Simon Fraser University, Canada. His research interests include relational media big data analysis, multimedia intelligence and recommendation in social media. He has published over 100 high-quality research papers in top journals and conferences including IEEE TPAMI, IEEE TKDE, ACM TOIS, ICML, NeurIPS, ACM KDD, ACM Web Conference, ACM SIGIR and ACM Multimedia etc. He is the recipient of 2017 China Postdoctoral innovative talents supporting program. He received the ACM China Rising Star Award in 2020 and IEEE TCMC Rising Star Award in 2022.

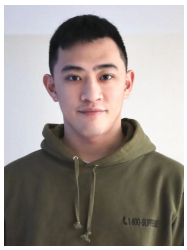
Heng Chang is currently pursuing a Ph.D. Degree in the Tsinghua-Berkeley Shenzhen Institute at Tsinghua University. He received his B.S. from the Department of Electronic Engineering, Tsinghua University in 2017. His research interests focus on representation learning, adversarial robustness, and machine learning on graph/relational structured data. He has published several papers in prestigious conferences/journals including NeurIPS, AAAI, TheWebConf, TKDE, TPAMI, *etc*.

Beini Xie is currently an M.A. student in the Tsinghua-Berkeley Shenzhen Institute at Tsinghua University. She received her B.S from the Statistical Department, Renmin University in 2020. Her research interests include adversarial robustness, machine learning and neural architecture search on graph/relational structured data.

Tian Bian is currently a Ph.D. student at The Chinese University of Hong Kong. He received his B.E. degree from Southwest University in 2018 and his Master's degree from Tsinghua University in 2021. His research interests focus on graph neural networks and their applications such as social media analysis. He has served as a reviewer for ICML, NeurIPS, AAAI, etc.

Shiji Zhou is currently a Ph.D. candidate at Tsinghua University. He received his B.S. from the Chinese Academy of Science-Beihang Hua Luogeng Mathematics Honors Class, Beihang University in 2017. His research covers online learning and multi-objective optimization, as reflected in his publications on top-tier conferences and journals, including NeurIPS, AISTATS, TMM. He has been the co-organizer of the 3rd Human in the Loop Learning (HILL) workshop on ICML. He has served as a reviewer for ICML, NeurIPS, JSAC, etc.

Daixin Wang received his Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018, and he is currently working as an algorithm expert on Ant Group. He has authored or coauthored more than 10 papers in conferences such as KDD, AAAI, and IJCAI, and journals such as the IEEE Transactions on Multimedia. His research interests include graph learning and multimodal learning.



Zhiqiang Zhang is currently a Staff Engineer at Ant Group. His research interests mainly focus on graph machine learning. He has led a team to build an industrial graph machine learning system, AGL, in Ant Group. He has published more than 30 papers in top-tier machine learning and data mining conferences, including NeurIPS, VLDB, SIGKDD, and AAAI.

Wenwu Zhu is currently a Professor in the Department of Computer Science and Technology at Tsinghua University, the Vice Dean of National Research Center for Information Science and Technology. Prior to his current post, he was a Senior Researcher and Research Manager at Microsoft Research Asia. He was the Chief Scientist and Director at Intel Research China from 2004 to 2008. He worked at Bell Labs New Jersey as a Member of Technical Staff during 1996-1999. He received his Ph.D. degree from New York University in 1996.

His current research interests are in the area of data-driven multimedia networking and multimedia intelligence. He has published over 350 referred papers and is inventor or co-inventor of over 50 patents. He received ten Best Paper Awards, including ACM Multimedia 2012 and IEEE Transactions on Circuits and Systems for Video Technology in 2001 and 2019.

He served as EiC for IEEE Transactions on Multimedia (2017-2019), the chair of the steering committee for IEEE Transactions on Multimedia (2019-2021), and the Associate EiC for IEEE Transactions for Circuits and Systems for Video Technology. He serves as General Co-Chair for ACM Multimedia 2018 and ACM CIKM 2019, respectively. He is an ACM Fellow, AAAS Fellow, IEEE Fellow, SPIE Fellow, and a member of The Academy of Europe (Academia Europaea).