Exact Verification of Graph Neural Networks with Incremental Constraint Solving

Minghao Liu, Chia-Hsuan Lu, and Marta Kwiatkowska

University of Oxford, United Kingdom
{minghao.liu,chia-hsuan.lu,marta.kwiatkowska}@cs.ox.ac.uk

Abstract. Graph neural networks (GNNs) are increasingly often employed in high-stakes applications, such as fraud detection or healthcare, but are susceptible to adversarial attacks. A number of techniques have been proposed to provide adversarial robustness guarantees, but support for commonly used aggregation functions in message-passing GNNs is lacking. In this paper, we develop an exact (sound and complete) verification method for GNNs to compute guarantees against attribute and structural perturbations that involve edge addition or deletion, subject to budget constraints. Our method employs constraint solving with bound tightening, and iteratively solves a sequence of relaxed constraint satisfaction problems while relying on incremental solving capabilities of solvers to improve efficiency. We implement GNNEv, a versatile exact verifier for message-passing neural networks, which supports three aggregation functions – sum, max and mean – with the latter two considered here for the first time. Extensive experimental evaluation of GNNEv on real-world fraud datasets (Amazon and Yelp) and biochemical datasets (MUTAG and ENZYMES) demonstrates its usability and effectiveness, as well as superior performance on node classification and competitiveness on graph classification compared to existing exact verification tools on sum-aggregated GNNs.

Keywords: Formal verification · Constraint solving · Graph neural networks · Adversarial robustness

1 Introduction

Graph neural networks (GNNs) have been widely deployed in real-world applications, such as financial fraud detection [53, 71], autonomous driving [14, 15], healthcare treatment [27, 45], and scientific discovery [36, 58]. However, like all other neural network architectures, GNNs are vulnerable to adversarial attacks, where slight perturbations can cause a prediction change [21, 67, 87]. Therefore, it is desirable to ensure adversarial robustness of GNNs for high-stakes applications.

Formal verification is a rigorous methodology to mathematically prove that a system meets its specifications under all possible circumstances [5]. To evaluate the reliability and trustworthiness of neural networks, a number of approaches

have been developed to verify the adversarial robustness of fully connected neural networks [11, 38, 72, 74, 81], convolutional neural networks (CNNs) [10, 35, 69], recurrent neural networks (RNNs) [1, 24, 40], and transformers [9, 66]. Technically, these approaches can be classified into two categories. The first category is *exact* verification, also known as *complete* verification, where the verification task is formulated as a constraint satisfaction problem (CSP) [60] to deterministically prove whether the model is robust or not. The second category is *approximate* verification, also known as (sound but) *incomplete* verification, where the non-convex constraints are typically relaxed to convex ones, which admits efficient computation of a lower bound on the robust region. However, when the lower bound cannot be accurately computed, the outcome of the verification may be inconclusive, which does not offer strong robustness guarantees.

Recently, the concept of adversarial robustness has been extended to GNNs. Graph inputs present additional challenges, in that the perturbations can pertain to both node attributes [2, 89] and graph structure, i.e., the addition and deletion of edges [7, 37, 41, 90], or even node injection [42]. Since graph inputs are encoded using a combination of real-valued and discrete data, typical approaches to provide *deterministic* robustness guarantees are based on Mixed-Integer Programming (MIP) [33, 82]. We also mention methods based on randomised smoothing [8, 56, 63], noting that their guarantees are *probabilistic* and thus not directly comparable.

Despite recent advances, existing exact verification approaches are still limited to GNNs with *sum* aggregation. However, other common aggregations, including *max* and *mean*, have been shown to be necessary and effective in both theory [20, 59] and practice [23, 80], and are supported as standard by the GraphSAGE library [32]. Existing methods also typically focus on graph classification problems in the context of structural perturbations, implemented only for edge deletion for tractability reasons. Node classification tasks have been under-explored, and yet are relied upon in high-stakes applications, for example, fraud detection in financial networks [71] and cyber attack localization in smart grids [31]. For such applications, in order to provide certified robustness guarantees, methods that yield conclusive outcomes are preferable.

We develop an exact verification method for GNNs to provide guarantees against both attribute and structural perturbations subject to local and global budget constraints. We support GNNs for both node- and graph-classification tasks with three commonly used aggregation functions, sum, max and mean, where the latter two are non-linear. We employ constraint solving with bound tightening and iteratively solve a sequence of relaxed CSPs while relying on incremental solving capabilities of solvers to improve efficiency. We implement GNNev, a versatile and efficient exact verifier for GNNs, and conduct extensive experiments on two standard node-classification datasets, Cora and CiteSeer, two real-world fraud datasets, Amazon and Yelp, and two biochemical graph-classification datasets, MUTAG and ENZYMES. The results indicate that GNNev outperforms SCIP-MPNN [33], the only MIP-based exact verifier known to us, on sum aggregation and edge deletion under different model sizes and per-

turbation budgets. On a range of aggregation functions, we show that GNNEv can provide useful insight into the susceptibility of GNNs to adversarial attacks, which is important for deployment in high-stakes applications.

Our novel contributions can be summarised as follows.

- We introduce the first exact verification method for GNNs with two common aggregation functions, max and mean, and design specialised tightened bound propagation strategies to reduce computational cost.
- We propose an algorithm that iteratively encodes GNN layers backwards and utilises incremental constraint solving to speed up performance while maintaining exact verification capability.
- We implement GNNEv, a versatile and efficient exact verifier for GNNs that supports new aggregation functions and edge addition, and show its usability on both structural and attribute perturbations through extensive experiments on real-world datasets.

Due to space constraints, additional details have been relegated to appendices in the full version of the paper [46].

2 Related Work

Adversarial attacks against GNNs. Several works [21, 77, 87, 88] demonstrate that adversarial attacks can induce incorrect predictions by GNNs on node classification. Chang et al. [16], Mu et al. [54] propose black-box attack approaches without accessing any knowledge of the GNN classifiers. Li et al. [44], Zou et al. [86] further improve the efficiency and scalability of adversarial attack algorithms. Another direction is attacks on graph classification [84] and link prediction [17, 85]. We emphasise that a failure to find an attack does not imply robustness. In contrast, we aim to verify that GNNs are not vulnerable to any admissible attack, which provides strong guarantees and can thus be used for robustness certification.

We further remark that, while most studies focus on undirected graphs and claim potential extensions to directed graphs, some works [17, 28], along with our paper, consider both directed and undirected graphs.

Robustness enhancement of GNNs. Adversarial training is a common technique applied to enhance the robustness of GNNs [22, 25, 30, 43, 77]. Wu et al. [75], Zhang and Zitnik [83] aim to enhance the robustness of GNNs by estimating potential adversarial perturbations and adjusting model weights to defend against these perturbations. Moreover, generalised randomised smoothing [8, 70, 84] and partition-ensemble [76] techniques have been developed for GNNs to provide defences against attacks with probabilistic guarantees. Instead, we develop a formal verification algorithm, which is complementary to robustness enhancement and offers deterministic guarantees.

Theoretical analysis of GNN verification. Expressiveness of sum-aggregated GNNs is studied in [4, 6, 55] and mean-aggregated in [64]. The adversarial robustness problem for node classification is shown decidable in Sälzer and Lange [62] under the assumption of bounded degree of input graphs. Benedikt et al. [6] (and Nunn et al. [55]) study verification of certain properties that quantify over all graph inputs proving PSPACE-completeness for GNNs with rational (resp. integer) coefficients, Boolean-attributed input graphs and truncated ReLU activation functions. We focus instead on a practical verification approach applicable to commonly used GNNs, with real-valued coefficients and attributes and ReLU activation functions. Our problem setting is clearly decidable because it only considers a finite and bounded input graph set.

Verification of neural networks. Adversarial robustness verification methods can be classified into complete (exact) methods, such as constraint solving [35, 38, 68], sound though incomplete methods, e.g., convex relaxation [61, 79], which can be strengthened to completeness by employing branch-and-bound procedures [13, 48], and probabilistic methods [19, 49, 73]. Katz et al. [38] prove that exact NN verification is generally NP-hard. More recent works move beyond fully-connected and convolutional architectures, and include MIP-based exact verification of recurrent networks [1] as well as approximate verification of transformers utilising the zonotope abstract domain [9] and linear bound propagation [34]. While some works focus on properties such as differential verification between models [51] and batched verification across inputs [3], we aim to verify the key property of adversarial robustness. Regarding GNN verification, existing approaches concern graph convolutional networks with respect to attribute [2, 89] or structural [7, 37, 41, 90] perturbations. Most are approximate, relying on optimisation [7, 37, 41, 90] or zonotope-based relaxation [41]. Our approach is most similar to the exact MIP-based approach SCIP-MPNN [33] for sum-aggregated message-passing GNNs and edge deletion only. Their method supports static and dynamic (aggressive) bound tightening over the complete network encoding. In contrast, our method also supports max and mean, as well as edge addition, and proceeds by building the encoding incrementally layer by layer while statically tightening the bounds. For node classification tasks involving large numbers of neighbouring nodes, our exact method can greatly reduce the cost of encoding, thus enhancing performance. We also note a recent verifier RobLight [47], which supports structural perturbations only and references our work. Their method relies on a branch-and-bound approach with bound propagation, and is unable to generate certificates for verification that MIP-based methods permit [18].

3 Preliminaries

3.1 Graph Neural Networks

Let $G = \langle V, E, X \rangle$ be an *attributed directed graph*, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of nodes, $E \subseteq V \times V$ is a set of edges with no self-loops, and $X =$

$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is a set of real *attribute vectors* for the nodes with the same dimension. For a node $v \in V$ and a natural number k, we denote the set of *k-hop incoming neighbours of v* by $\mathcal{N}_k(v)$. For simplicity, we write $\mathcal{N}(v)$ when $k = 1$. *Graph neural networks (GNNs)* are a class of neural network architectures designed to operate on graph data. In this paper, we consider node- and graph-classification tasks. Given a class set $C = \{c_1, c_2, \ldots, c_m\}$, for node classification, the goal is to classify a node $v \in V$ into a class within C; for graph classification, the goal is to classify a graph G into a class within C. More precisely, we view a GNN as a function f such that, given an attributed directed graph $G = \langle V, E, X \rangle$ and a node $v \in V$, $f(G, v) \in C$ for node classification, and $f(G) \in C$ for graph classification. The remainder of the formalisation focuses on the node classification case for simplicity. The variant for graph classification is similar, as discussed in Appendix A in the extended version [46].

There are a variety of GNN architectures. Existing formal verification works mostly focus on graph convolutional networks (GCNs) [39], which rely on a specific neighbourhood aggregation scheme and their expressiveness is restricted [78]. In this work, we consider *GraphSAGE* [32], also known as *Message-Passing Neural Networks (MPNNs)* [29], which are a more general GNN architecture because (i) they support the choice of multiple neighbourhood aggregation functions and (ii) the expressive power of MPNNs has been shown to encompass other popular variations such as GCNs and graph attention networks (GATs) [12].

A GNN consists of K layers. The dimension of a GNN is a $(K+1)$ tuple of positive integers $d_0, d_1, \ldots, d_K$. For an input attributed directed graph $G = \langle V, E, X \rangle$, for each node $v \in V$, its real-valued embedding vector $\mathbf{h}_v^{(0)}$ is a d_0-dimension vector initialised to its corresponding attribute vector $\mathbf{x}_v$. For the k-th layer, GNNs compute the d_k-dimensional embedding $\mathbf{h}_v^{(k)}$ by

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}_1^{(k)} \cdot \mathbf{h}_v^{(k-1)} + \mathbf{W}_2^{(k)} \cdot \mathbf{aggr}\left(\left\{\!\!\left\{ \mathbf{h}_u^{(k-1)} \mid u \in \mathcal{N}(v) \right\}\!\!\right\}\right) + \mathbf{b}_1^{(k)}\right), \quad (1)$$

where $\mathbf{W}_1^{(k)}, \mathbf{W}_2^{(k)}$ are learnable parameter matrices with dimension $d_k \times d_{k-1}$ for the k-th layer, $\mathbf{b}_1^{(k)}$ is a learnable parameter vector, $\sigma(x) = \max(0, x)$ is the ReLU activation function, $\mathbf{aggr} \in \{\mathrm{sum}, \mathrm{max}, \mathrm{mean}\}$ is the aggregation function, and $\{\!\{\}\!\}$ denotes a multiset. Finally, the last dimension d_K is exactly m, and the predicted class $\hat{c}_v$ of a node $v \in V$ is obtained by the Softmax function:

$$\hat{c}_v = \arg\max_{c \in C} \left(\mathrm{Softmax}\left(\mathbf{h}_v^{(K)}\right)[c]\right). \quad (2)$$

3.2 Adversarial Robustness of GNNs

Node embeddings are directly affected by attribute perturbations and indirectly affected by structural perturbations through message passing, which may result in prediction instability. Changes in predictions caused by admissible perturbations indicate a lack of *adversarial robustness*. We slightly adapt the definition of adversarial robustness for GNNs in [7] by (i) allowing perturbations in both

graph structure and node attributes, and (ii) removing the redundant fixed edge set. Given an attributed directed graph $G = \langle V, E, X \rangle$, the attacker typically has limited capabilities to perturb it. Let $F \subseteq V \times V$ be a set of *fragile edges*. Intuitively, an attacker can either remove edges in $F \cap E$ or insert edges in $F \backslash E$. For structural perturbations, we assume a global budget Δ and a local budget δ_v for each $v \in V$, both of which are non-negative integers. For attribute perturbations, we work with budgets $\epsilon^l_{v,i}, \epsilon^u_{v,i}$ ($\epsilon^l_{v,i} \leq \mathbf{x}_v[i] \leq \epsilon^u_{v,i}$) for each $v \in V$ and $i \in \{1, 2, \ldots, d_0\}$, which are real values representing the range of perturbation allowed for the i-th dimension of $\mathbf{x}_v \in X$. The admissible perturbation space is defined as follows.

Definition 1 (Admissible perturbation space of graph). *Given an attributed directed graph $G = \langle V, E, X \rangle$, a set of fragile edges $F \subseteq V \times V$, and perturbation budgets Δ, δ, ϵ, the* admissible perturbation space $\mathcal{Q}(G)$ *of G with respect to F and budgets is the set of attributed directed graphs $\tilde{G} = \langle V, \tilde{E}, \tilde{X} \rangle$ that satisfy*

1. $E \backslash F \subseteq \tilde{E} \subseteq E \cup F$,
2. $|E \backslash \tilde{E}| + |\tilde{E} \backslash E| \leq \Delta$,
3. *For every* $v \in V$, $|\mathcal{N}(v) \backslash \tilde{\mathcal{N}}(v)| + |\tilde{\mathcal{N}}(v) \backslash \mathcal{N}(v)| \leq \delta_v$,
4. *For every* $v \in V$ *and* $1 \leq i \leq d_0$, $\epsilon^l_{v,i} \leq \tilde{\mathbf{x}}_v[i] \leq \epsilon^u_{v,i}$,

where we denote the incoming neighbours of v in the graph $\tilde{G}$ by $\tilde{\mathcal{N}}(v)$. Note that this is a general formulation as F can be defined arbitrarily for different purposes, and allows us to consider edge addition when the size of F is moderate[1]. For example, if $F = E$, only edge deletion is allowed, and any existing edge can be deleted. If $F = V \times V$, edges can be added or deleted between any pair of nodes.

Finally, we introduce the formal definition of adversarial robustness of GNNs.

Definition 2 (Adversarial robustness of GNNs). *Given a node-classification GNN f, an attributed directed graph G with admissible perturbation space $\mathcal{Q}(G)$, a target node $t \in V$, and its predicted class $\hat{c}_t \in C$, we say that f is* adversarially robust *for t with class $\hat{c}_t$ if and only if, for every perturbed graph $\tilde{G} \in \mathcal{Q}(G)$, it holds that $f(\tilde{G}, t) = \hat{c}_t$.*

Adversarial robustness verification aims to guarantee that the prediction for a given node will not change under any admissible perturbations, subject to budget constraints, and can be reduced to computing the worst-case margin between the target class and other classes.

4 Exact Verification of GNNs

In this section, we introduce our method, which encodes the verification task as a constraint satisfaction problem (CSP). Next, the bound tightening strategies for max- and mean-aggregated GNNs are proposed. Finally, we illustrate an efficient exact verification algorithm based on incremental solving.

[1] The definition in Hojny et al. [33], Eq. (14) is a special case of ours when $F = E$.

4.1 Encoding

A CSP aims to determine whether there exists an assignment of variables that satisfies a given set of constraints. We encode the exact verification tasks as CSPs, which consist of three parts: input perturbation, GNN architecture, and verification objective. Following existing MIP encodings for GNNs [33, 82], we extend them to accommodate max and mean aggregations, where for mean we use big-M encoding and for max we rely on the solver.

Input perturbation. Consider an input attributed directed graph $G = \langle V, E, X \rangle$, where our goal is to verify the adversarial robustness for the target node $t \in V$. Note that, for K-layer GNNs, only perturbations to nodes in $\mathcal{N}_K(t)$ can affect the prediction of t. Two kinds of perturbations are allowed. The first is attribute perturbation, where the node attributes can be modified within a specified range. For each node $v \in \mathcal{N}_K(t)$, we set d_0 real variables $attr_{v,1}, attr_{v,2}, \ldots, attr_{v,d_0}$ as the attribute values after perturbations. Given the perturbation budgets $\epsilon^l_{v,i}, \epsilon^u_{v,i}$ for the i-th dimension of $\mathbf{x}_v$, the following constraint is applied:

$$\epsilon^l_{v,i} \;\leq\; attr_{v,i} \;\leq\; \epsilon^u_{v,i}. \tag{3}$$

The second type is structural perturbation, which can impact the message-passing mechanism. For each edge $(u, v) \in F$, we set a Boolean variable $pe_{u,v}$ to represent whether (u, v) is perturbed. Given the global budget Δ and the local budget δ_v for each $v \in \mathcal{N}_{K-1}(t)$, we have the constraints

$$\sum_{(u,v)\in F} pe_{u,v} \leq \Delta, \qquad \sum_{(u,v)\in F} pe_{u,v} \leq \delta_v. \tag{4}$$

GNN architecture. We encode the architecture of GNNs as follows. For the k-th layer as shown in Eq. (1), the output embedding of node $v \in \mathcal{N}_{K-k}(t)$ is represented by d_k real variables $h^{(k)}_{v,1}, h^{(k)}_{v,2}, \ldots, h^{(k)}_{v,d_k}$. Let $h^{(0)}_{v,i} = attr_{v,i}$. The encoding for each layer consists of three parts.

First, the neighbours of node v are aggregated to produce the message. We set d_{k-1} real variables $msg^{(k)}_{v,1}, msg^{(k)}_{v,2}, \ldots, msg^{(k)}_{v,d_{k-1}}$ as the message vector. We support three common aggregation functions: sum, max, and mean. Note that the last two functions are non-linear, which accounts for higher computational complexity for verification. Before we give the encoding for the aggregation functions, we set the auxiliary variables $a^{(k)}_{v,i,u}$, which indicate the contribution of node u to $msg^{(k)}_{v,i}$ through the edge (u, v). The following constraints restrict the values of $a^{(k)}_{v,i,u}$: for every $(u, v) \in E \backslash F$,

$$a^{(k)}_{v,i,u} = h^{(k-1)}_{v,i}; \tag{5}$$

for every $(u, v) \in E \cap F$,

$$pe_{u,v} \rightarrow a^{(k)}_{v,i,u} = 0 \quad \text{and} \quad \neg pe_{u,v} \rightarrow a^{(k)}_{v,i,u} = h^{(k-1)}_{v,i}; \tag{6}$$

for every $(u, v) \in F \backslash E$,

$$pe_{u,v} \rightarrow a_{v,i,u}^{(k)} = h_{v,i}^{(k-1)} \quad \text{and} \quad \neg pe_{u,v} \rightarrow a_{v,i,u}^{(k)} = 0. \tag{7}$$

For sum and max aggregation, the constraints are shown in Eq. (8) and (9), respectively.

$$msg_{v,i}^{(k)} = \sum_{(u,v) \in F \cup E} a_{v,i,u}^{(k)}, \tag{8}$$

$$msg_{v,i}^{(k)} = \max_{(u,v) \in F \cup E} a_{v,i,u}^{(k)}. \tag{9}$$

For mean aggregation, the constraints are set as follows:

$$\begin{aligned} deg_v &= \sum_{(u,v) \in F \cap E} (1 - pe_{u,v}) + \sum_{(u,v) \in F \backslash E} pe_{u,v} + |E \backslash F|, \\ deg_v \cdot msg_{v,i}^{(k)} &= \sum_{(u,v) \in F \cup E} a_{v,i,u}^{(k)}, \\ -M \cdot deg_v &\leq msg_{v,i}^{(k)} \leq M \cdot deg_v, \end{aligned} \tag{10}$$

where M is a big real number[2]. Next, we introduce a constraint that represents the pre-ReLU embedding of node v as follows:

$$y_{v,i}^{(k)} = \sum_{j \in [1, d_{k-1}]} \mathbf{W}_1^{(k)}[i, j] \cdot h_{v,j}^{(k-1)} + \mathbf{W}_2^{(k)}[i, j] \cdot msg_{v,j}^{(k)} + \mathbf{b}_1^{(k)}. \tag{11}$$

Finally, we define the post-ReLU embedding, except the last GNN layer, as follows:

$$h_{v,i}^{(k)} = \max\left(y_{v,i}^{(k)}, 0\right). \tag{12}$$

Verification objective. According to Definition 2, our objective is to verify that no admissible attribute or structural perturbation can make the prediction of the GNN inconsistent with the input attributed directed graph for a target node. Let the predicted class of target node t be $\hat{c}_t$, where the set of classes is denoted by C. Then we set the following constraint:

$$\left(\max_{c \in C \backslash \{\hat{c}_t\}} y_{t,c}^{(K)}\right) \geq y_{t,\hat{c}_t}^{(K)}, \tag{13}$$

which expresses the worst-case margin in the input of Softmax.

Let $\mathbf{y}_t^{(K)} = \left[y_{t,1}^{(K)}, y_{t,2}^{(K)}, \ldots, y_{t,|C|}^{(K)}\right]$. Since the Softmax function is monotonically increasing, Eq. (13) implies that there exists $c \in C$ with $c \neq \hat{c}_t$ such that

$$\text{Softmax}\left(\mathbf{y}_t^{(K)}\right)[c] \geq \text{Softmax}\left(\mathbf{y}_t^{(K)}\right)[\hat{c}_t].$$

[2] M is calculated by finding the maximum of the absolute values of the upper and lower bounds of all $a_{v,i,u}^{(k)}$ in Eq. (10).

If Eq. (13) is feasible, this means that a perturbed graph has been found that makes the GNN misclassify node t. Therefore, the robustness verification task is translated to deciding whether the above CSP is unsatisfiable.

Note that the encoding involves at most $5N^2D$ real variables, N^2 Boolean variables, and $8ND + 2$ constraints, where N is the number of nodes in $\mathcal{N}_K(t)$ and $D := \sum_{0 \le i \le K} d_i$. Furthermore, the encoding can be computed in time polynomial in the size of the GNN.

4.2 Bound Tightening for Aggregations

Tightening of variable bounds has been found to be critical to enhancing the efficiency of MIP solvers. Simultaneously, high-quality bounds form the basis of our incremental solving algorithm introduced in Section 4.3. Starting from the input variables (i.e., *attr*), whose bounds have been determined by Eq. (3), we propagate these bounds layer by layer to obtain tight bounds for other variables.

Since tightened bound propagation is straightforward for linear transformations and ReLU functions, our main focus is on tightening the bounds for aggregation functions. We formalise the problem as follows: given three finite sets of variables X_1, X_2, and X_3, the upper and lower bounds for each variable, and a non-negative integer s, compute the upper and lower bounds for the variable

$$z := \mathbf{aggr}(X_1 \cup X_2' \cup X_3'),$$

where $\mathbf{aggr} \in \{\mathrm{sum}, \mathrm{max}, \mathrm{mean}\}$, $X_2' \subseteq X_2$, $X_3' \subseteq X_3$, and $|X_2 \backslash X_2'| + |X_3'| \le s$. Let N be the total size of the sets X_1, X_2, and X_3. The intuition behind the formalisation is as follows: X_1, X_2, and X_3 represent the embedding vectors of nodes connected to the target node via non-fragile edges (E/F), fragile edges ($E \cap F$), and fragile non-edges (F/E), respectively. The attacker can either delete fragile edges, that is, remove elements from X_2, or insert fragile non-edges, that is, select elements from X_3. The number of deletion and insertions is constrained by the budget s.

For a set of variables X and an integer k, let $hi(X, k)$ denote the k-th *largest upper bound* among the variables in X. If $k > |X|$, we define $hi(X, k) = -\infty$. Similarly, let $lo(X, k)$ denote the k-th *smallest lower bound* among the variables in X, with $lo(X, k) = \infty$ when $k \le 0$. Note that $hi(X, k)$ and $lo(X, k)$ can be computed in $O(|X| \log k)$ time by maintaining a max- or min-heap. For multiple values $k_1, k_2, \ldots, k_\ell$, we can reuse the heap, resulting in an overall complexity of $O(|X| \log k)$, where k is the maximum value among $k_1, k_2, \ldots, k_\ell$.

For a variable x, its upper and lower bounds are denoted by $\overline{x}$ and $\underline{x}$ respectively. The case for sum aggregation has been shown by Hojny et al. [33]. We restate their bounds in the context of our formalisation for completeness:

$$\overline{z} = \sum_{x \in X_1 \cup X_2} \overline{x} + \sum_{1 \le i \le s} \max(hi(Y, i), 0),$$
$$\underline{z} = \sum_{x \in X_1 \cup X_2} \underline{x} + \sum_{1 \le i \le s} \min(lo(Y, i), 0),$$

where $Y = \{-x \mid x \in X_2\} \cup X_3$. The upper and lower bounds of z can be computed in time $O(N \log s)$.

Max aggregation. The upper and lower bounds of z for max aggregation can be obtained by case analysis. For the corner case $s = 0$, $z = \max(X_1 \cup X_2)$. Note that, if $X_1 \cup X_2 = \emptyset$, then $\overline{z} = \underline{z} = 0$. We now focus on the case $s > 0$. For the upper bound:

$$\overline{z} = \begin{cases} \max(0, hi(X_2, 1), hi(X_3, 1)), & \text{if } X_1 = \emptyset, s \geq |X_2|, \\ \max(hi(X_1, 1), hi(X_2, 1), hi(X_3, 1)), & \text{otherwise.} \end{cases}$$

On the other hand, for the lower bound:

$$\underline{z} = \begin{cases} \min(0, lo(X_2, 1), lo(X_3, 1)), & \text{if } X_1 = \emptyset, s > |X_2|, \\ \min(0, lo(X_2, 1)), & \text{if } X_1 = \emptyset, s = |X_2|, \\ lo(X_2, |X_2| - s), & \text{if } X_1 = \emptyset, s < |X_2|, \\ lo(X_1, |X_1|), & \text{if } X_1 \neq \emptyset, s \geq |X_2|, \\ \max(lo(X_1, |X_1|), lo(X_2, |X_2| - s)), & \text{if } X_1 \neq \emptyset, s < |X_2|. \end{cases}$$

Note that the upper and lower bounds for z can be computed in time $O(N \log s)$.

Mean aggregation. In this case, we reduce the original problem to a combination of subproblems. For non-negative integers s_2 and s_3, define the variable

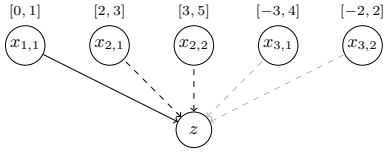
$$z_{s_2,s_3} := \mathbf{mean}(X_1 \cup X_2' \cup X_3'),$$

where $X_2' \subseteq X_2$, $X_3' \subseteq X_3$, $|X_2 \backslash X_2'| = s_2$, and $|X_3'| = s_3$. The main difference is that we now explicitly fix separate budgets for deletions and insertions. The upper and lower bounds of the original problem z are given by

$$\overline{z} = \max_{\substack{s_2+s_3 \leq s \\ 0 \leq s_2 \leq |X_2| \\ 0 \leq s_3 \leq |X_3|}} \overline{z}_{s_2,s_3} \quad \text{and} \quad \underline{z} = \min_{\substack{s_2+s_3 \leq s \\ 0 \leq s_2 \leq |X_2| \\ 0 \leq s_3 \leq |X_3|}} \underline{z}_{s_2,s_3}.$$

For the subproblem z_{s_2,s_3}, since the budgets for deletion and insertion are separate, the upper and lower bounds can be obtained through greedy choices. In the corner case where X_1 is empty, $s_2 = |X_2|$, and $s_3 = 0$, $\overline{z}_{s_2,s_3} = \underline{z}_{s_2,s_3} = 0$. In all other cases, we have $\overline{z}_{s_2,s_3} = \overline{w}_{s_2,s_3}/n_{s_2,s_3}$ and $\underline{z}_{s_2,s_3} = \underline{w}_{s_2,s_3}/n_{s_2,s_3}$, where $n_{s_2,s_3} = |X_1| + |X_2| - s_2 + s_3$ and

$$\overline{w}_{s_2,s_3} = \sum_{x \in X_1} \overline{x} + \sum_{1 \leq i \leq |X_2| - s_2} hi(X_2, i) + \sum_{1 \leq i \leq s_3} hi(X_3, i),$$
$$\underline{w}_{s_2,s_3} = \sum_{x \in X_1} \underline{x} + \sum_{1 \leq i \leq |X_2| - s_2} lo(X_2, i) + \sum_{1 \leq i \leq s_3} lo(X_3, i).$$



	Plain	Tightened
max (*ub*)	$\max(1,3,5,4,2) = \mathbf{5}$	$\max(1,5,4) = \mathbf{5}$
max (*lb*)	$\min(0,2,3,-3,-2) = -3$	$\max(0,2) = \mathbf{2}$
mean (*ub*)	$\frac{1+5+4}{3} = 3.33$	$\frac{1+5+3+4}{4} = \mathbf{3.25}$
mean (*lb*)	$\frac{0-3-2}{3} = -1.67$	$\frac{0+2+3-3}{4} = \mathbf{0.5}$

Fig. 1: An illustration of tightened bound propagation. Solid lines denote the non-fragile edges, black dashed lines denote fragile edges, and grey dashed lines denote fragile non-edges. The tighter bounds are shown in **bold**.

Note that the summations over *lo* and *hi* can be computed incrementally among subproblems and, as there are at most $(s+1)^2$ subproblems, we can compute the upper and lower bounds for z in time $O(s^2 + N \log s)$ using a naïve approach. In fact, we can further improve this result by exploiting the unimodality of the sequence, reducing the computation time to $O((s+N)\log s)$. The detailed algorithm is provided in Appendix B in the extended version [46].

Theorem 1. *The upper and lower bounds provided in this section for max and mean aggregations are tight. That is, there exist sets $X_2' \subseteq X_2$ and $X_3' \subseteq X_3$ with $|X_2 \backslash X_2'| + |X_3'| \leq s$ that achieve these bounds exactly.*

The proof is included in Appendix C in the extended version [46].

Example 1. Consider sets of variables $X_1 = \{x_{1,1}\}$, $X_2 = \{x_{2,1}, x_{2,2}\}$, and $X_3 = \{x_{3,1}, x_{3,2}\}$. Each variable has upper and lower bounds as shown in Fig. 1. Let the budget $s = 1$. We now consider tightened bound propagation:

- For max aggregation, the upper bound is 5 by no operation, the same as the plain method[3]; the lower bound is 2 by deleting $x_{2,2}$, improved by 5 over the plain method.
- For mean aggregation, the upper bound is 3.25 by inserting $x_{3,1}$, improved by 0.08; the lower bound is 0.5 by inserting $x_{3,1}$, improved by 2.17.

For both aggregations, tightened bound propagation reduces the domain of the variable z.

4.3 Verification with Incremental Solving

With tightened bound propagation, we can verify adversarial robustness of GNNs more efficiently. However, in node classification tasks, graphs are typically large in size, and for a node t we typically have that $|\mathcal{N}_{k+1}(t)| \gg |\mathcal{N}_k(t)|$ for the k-th layer, leading to rapid expansion of the encoding and a decrease in efficiency. To improve performance while maintaining exactness of verification, we utilise *incremental solving*, an effective mechanism in existing CSP solvers, to iteratively solve a series of simplified relaxation problems compared to the original problem.

[3] The plain method computes the upper (lower) bounds for max aggregation as the maximum (minimum) bound of all the variables, and for mean aggregation by sorting variable bounds from $X_2 \cup X_3$ in ascending order and finding the maximum (minimum) mean value among all postfix (prefix) sequences.

Algorithm 1 Exact Verification of GNNs with Incremental Solving

Input: Trained GNN f with K layers, attributed directed graph $G = \langle V, E, X \rangle$, target node t (for node classification), fragile edge set F, perturbation budgets Δ, δ, ϵ.
Output: Verification result.

1: $Bounds \leftarrow \{\underline{attr}, \overline{attr}\}$;
2: **for** k from 1 to K **do**
3: $\quad \varphi_k \leftarrow \text{EncodeGNNLayer}(f, k, G, F, \Delta, \delta, \epsilon)$;
4: $\quad Bounds \leftarrow \text{BoundPropagation}(\varphi_k, Bounds)$;
5: **end for**
6: $\varphi_{obj} \leftarrow \text{EncodeObjective}(\varphi_K, t)$; //Also encode final layer for graph classification.
7: $S_\Theta \leftarrow \text{InitIncSolver}()$;
8: $\Phi \leftarrow \varphi_{obj}$;
9: **for** k from K to 1 **do**
10: $\quad \Phi \leftarrow \Phi \wedge \varphi_k$;
11: $\quad (result, \Theta) \leftarrow \text{IncSolve}(S_\Theta, \Phi, Bounds)$;
12: $\quad$ **if** $result =$ `unsat` **then**
13: $\quad\quad$ **return** `robust`;
14: $\quad$ **end if**
15: **end for**
16: **return** `non-robust`;

Algorithm 1 describes our verification method for GNNs that leverages incremental solving. First, the CSP is formulated and the bounds of variables are derived based on the rules of bound tightening of Section 4.2 and layer-by-layer propagation (lines 1–6). Next, we solve the formula Φ iteratively using an incremental solving mechanism (lines 7–15). Initially, $\Phi = \varphi_{obj}$. For node classification, φ_{obj} only contains the variables representing the final embeddings of the target node t. For graph classification, φ_{obj} contains the variables representing the sum of final embeddings of all nodes, along with the final summation and linear transformation layer. In each iteration, new variables and the corresponding constraints are added to Φ, and a MIP solver S_Θ is then called (line 11), where Θ represents the generated cuts, which are redundant constraints produced by the solver to prune the search space. If Φ is found to be unsatisfiable, then the GNN has been verified to be `robust`. Otherwise, the encoding of the previous layer is added to Φ. Finally, if the result of the last iteration remains satisfiable, then the GNN is verified as `non-robust`. Note that, when each additional layer is added in the graph, the number of neighbouring nodes can increase significantly. Therefore, our algorithm effectively reduces the size of the encoding in early iterations, which leads to a more efficient exact verification process.

Finally, we show the correctness of Algorithm 1 with the proof in Appendix D in the extended version [46].

Theorem 2. *Given a GNN f, an attributed directed graph G, and a perturbation space $\mathcal{Q}(G)$, f is adversarially robust (for a node t, if it is a node-classification GNN) if and only if Algorithm 1 returns* `robust`.

5 Experimental Evaluation

5.1 Implementation and Setup

We implement our method as GNNEv, a versatile and efficient exact[4] verifier specifically designed for message-passing GNNs[5]. GNNEv calls the underlying Gurobi 11.0.3 solver[6] for MIP solving. Compared to available exact GNN verifiers, GNNEv has three advantages: (i) it accepts three commonly used aggregation functions, sum, max, and mean, which broadens usability, (ii) it allows for both node attribute perturbation and edge addition/deletion, making it applicable to a wider range of attack scenarios, and (iii) it is designed as a Python library that accepts trained models built by the SAGEConv module in PyTorch Geometric [26], and thus can be used directly in the same environment as model implementation and training.

To evaluate performance, we trained a batch of GraphSAGE models on two citation network datasets, Cora and CiteSeer [65], two real-world fraud datasets, Amazon [50] and Yelp [57], and two biochemical datasets, MUTAG and ENZYMES [52]. The first four datasets are for node classification and the last two are for graph classification. We set the number of GNN layers $K = 3$ and aggregation functions $\mathbf{aggr} \in \{\text{mean}, \text{max}, \text{sum}\}$. We regard robustness verification for each node/graph as a task, which checks prediction stability for any set of admissible perturbations up to a given budget. For evaluation purposes, we run experiments with attribute and structural perturbation separately, but GNNEv supports their simultaneous use. All experiments were conducted on a server with Intel Xeon Gold 6252 @ 2.10GHz CPU, 252GB RAM, and Ubuntu 18.04 OS. A verifier, including both GNNEv and the baselines, ran each task on a single CPU core for fair comparison. The time limit for each task was set to 300s. More details of the datasets and experimental setups can be found in Appendix E in the extended version [46].

5.2 Comparison with Baselines on Sum-aggregation

We compare GNNEv to SCIP-MPNN [33] on edge deletion[7], the only MIP-based exact verifier immediately applicable to adversarial robustness of GNNs[8]. SCIP-MPNN includes five variants with different underlying solvers and bound tightening strategies. To make fair comparison, we use the two versions that also

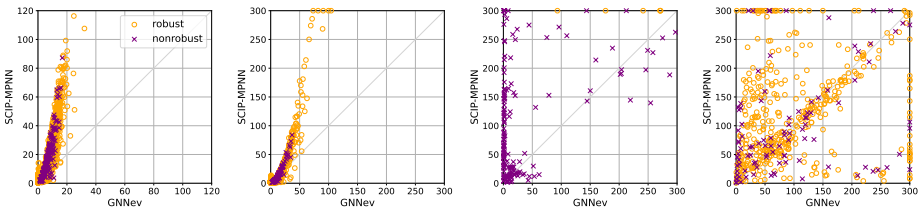
[4] For edge addition/deletion, GNNEv is a conditional exact verifier with respect to a given fragile edge set F.

[5] https://github.com/minghao-liu/GNNev

[6] https://www.gurobi.com

[7] We note that SCIP-MPNN does not implement edge addition and its interface does not support attribute perturbations.

[8] We remark that RobLight [47] outperforms our tool but it is not MIP-based, limited to structural perturbations only and does not allow certificate generation in the sense of [5].



(a) Cora (b) CiteSeer (c) MUTAG (d) ENZYMES

Fig. 2: Comparison of runtime (in seconds) for sum-aggregated GNNs on node classification (a-b) and graph classification (c-d). Each data point represents a robustness verification task with $\Delta = 2$. A point above the diagonal line indicates that GNNev outperforms SCIP-MPNN on that task.

call Gurobi: GRBbasic and GRBsbt[9]. For each task, we ran these two versions separately and reported the shorter runtime as the result of SCIP-MPNN. To align with the restrictions of SCIP-MPNN, we utilised GNNs with sum aggregation, only allowed edge deletions (i.e., $F = E$), and adapted our verification objective[10].

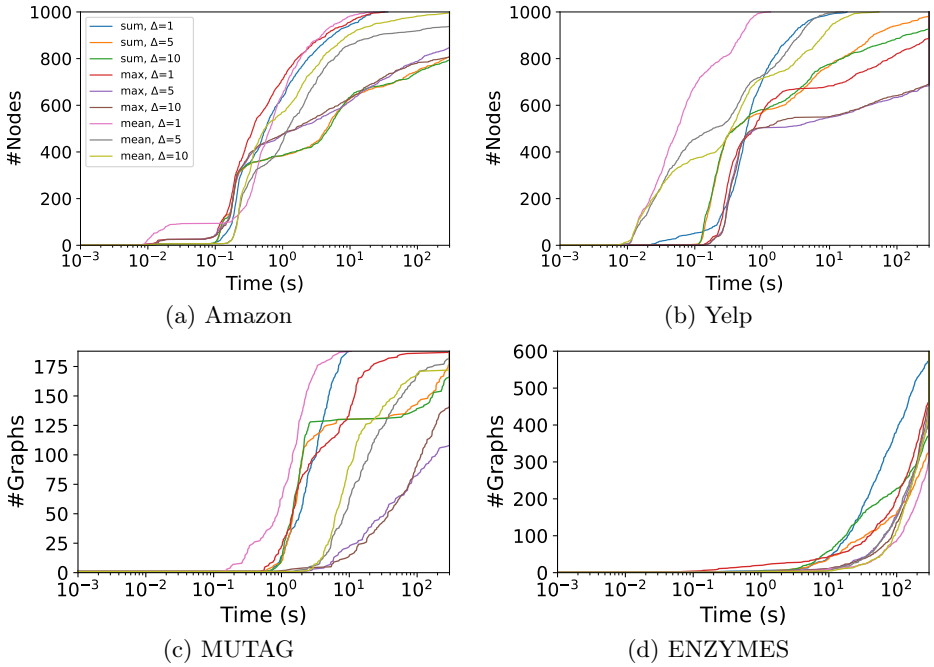
Results. Fig. 2 demonstrates the comparison of runtime between GNNev and SCIP-MPNN with edge deletions. For node-classification datasets, Cora and CiteSeer, GNNev solved all tasks within 40s and 120s, respectively, outperforming SCIP-MPNN on the vast majority of tasks. Note that GNNev also solved 5 challenging tasks on which SCIP-MPNN failed within the time limit on CiteSeer. For graph-classification datasets, GNNev demonstrated competitive performance. On MUTAG, GNNev solved most tasks within seconds, likely due to the simpler and more efficient encoding produced through incremental solving. On ENZYMES, a highly challenging dataset, GNNev also performed well compared to SCIP-MPNN by completing 6.1% of the solvable tasks that SCIP-MPNN could not complete within the time limit, and reducing runtime on 54.8% of the solvable tasks. Our experiments thus indicate that GNNev is a highly promising approach for exact verification. See Appendix F in the extended version [46] for detailed comparison results on different perturbation budgets.

5.3 Performance on All Aggregation Functions

Next, we systematically evaluated the performance of GNNev on sum-, max- and mean-aggregated GNNs with structural perturbations. For this experiment

[9] We excluded the three variants based on SCIP, noting that the experimental results in [33] showed that GRBbasic and GRBsbt have notably shorter runtime.

[10] According to Hojny et al. [33], Eq. (11) and Section 4.2, given the predicted class $\hat{c}_t$ for target node t, SCIP-MPNN aims to verify that $\mathbf{h}_t^{(K)}[\hat{c}_t] > \mathbf{h}_t^{(K)}[(\hat{c}_t + 1)\%|C|]$ holds for all perturbed graphs. This simpler objective, which provides weaker robustness guarantees than ours stated in Eq. (14), is used for tractability reasons.



(a) Amazon (b) Yelp

(c) MUTAG (d) ENZYMES

Fig. 3: The number of tasks solved by GNNEv on real-world node-classification (a-b) and graph-classification (c-d) datasets plotted against runtime under different aggregations and budgets. Only edge deletions are allowed ($F = E$). The verification objective is as defined in Section 4.1, Eq. (13). Full results are shown in Fig. 5 in Appendix F in the extended version [46].

we set the strong verification objective as Eq. (13), which is more challenging to compute than Hojny et al. [33], Eq. (11). Due to the large number of nodes in Amazon and Yelp, we randomly selected 1,000 nodes for verification. Two fragile edge sets F were used: (i) *edge deletions* are allowed for all edges, i.e., $F = E$, and (ii) *edge additions* are allowed for selected set, i.e., for each node v, several sampled non-edges $(u, v) \notin E$ are added to F. See Appendix F in the extended version [46] for more details.

Results. Fig. 3 illustrates the runtime of GNNEv on real-world node- and graph-classification datasets under different aggregations and budgets Δ for edge deletion; results on additional datasets and for edge addition are in Fig. 5,6 in Appendix F in the extended version [46]. First, MUTAG and ENZYMES are more challenging, as incremental solving can reduce the number of variables representing both nodes and layers for node classification, whereas it can only reduce the number of variables representing layers for graph classification. Second, across different aggregations, there was a performance drop for max aggregation when $\Delta \geq 5$, likely due to Gurobi struggling to efficiently process max constraints

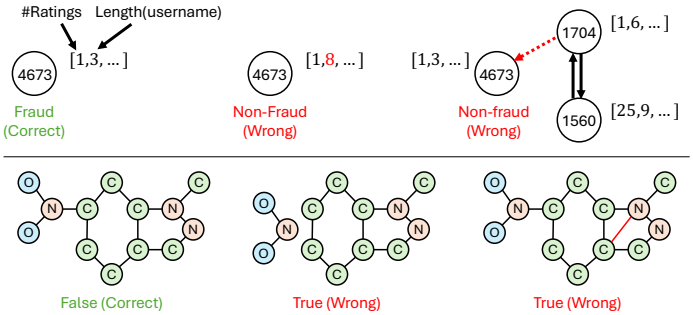


Fig. 4: Adversarial examples found by GNNEv. Top: Either modifying an attribute (here length of username) or adding an edge caused a node in the Amazon dataset to be misclassified as non-fraudulent. Bottom: Adding or deleting an edge on a graph in the MUTAG dataset leads to wrong prediction of having the given property even if the perturbed graphs are invalid.

using internal big-M encoding. Moreover, for sum aggregation, the performance on Amazon and Yelp dropped compared to Cora and CiteSeer. This can be attributed to the much larger gap between upper and lower bounds of variables, which is most pronounced for sum aggregation in Table 5 in Appendix F in the extended version [46], where we see a significant increase in the gap after each layer. Finally, we conducted ablation experiments to show the effectiveness of bound tightening and incremental solving, with results detailed in Appendix F in the extended version [46].

5.4 Adversarial Robustness Case Study

We further use GNNEv to analyse the robustness of GNNs for both structural and attribute perturbations, especially in high-stakes domains such as fraud detection and scientific discovery. First, we performed evaluation of the number of (non-)robust tasks verified plotted against an increasing global structural perturbation budget, shown in Fig. 7,8 in Appendix F in the extended version [46]. In most cases, instances were attacked successfully with a small budget. Some GNNs exhibited quite weak robustness. Specifically, we found that mean-aggregated GNNs were more vulnerable to adversarial perturbations than other types. Moreover, GNNEv was executed on Amazon and Yelp to verify mean-aggregated GNNs with attribute perturbation enabled. From the results shown in Fig. 9 in Appendix F in the extended version [46], even perturbing a single dimension of attributes can cause models to make wrong predictions in up to 75.8% of tasks.

Within these results, we identified a number of adversarial attacks on non-robust instances, which demonstrate the vulnerability of GNNs. Fig. 4 illustrates two cases from Amazon and MUTAG, respectively. In Amazon, one of the attributes is the length of username, which is easily manipulated in practice. We

configured GNNEv to only allow perturbations of this attribute within the range $[1, 10]$. The results showed that 29.1% of nodes were successfully attacked for the mean-aggregated GNN. In MUTAG, edge addition and deletion can easily lead to invalid structures, but GNNEv found that mean-aggregated GNNs reported 12.7% false positive results when $\Delta = 1$. These demonstrate that GNNEv is able to gain useful insight into the susceptibility of GNN models to adversarial attacks, which is important for pre-deployment analysis for high-stakes applications.

6 Conclusion and Future Work

We have developed an exact robustness verification method for message-passing GNNs supporting a range of aggregation functions, which is implemented in GNNEv, a versatile and efficient verifier. Our approach is based on a reduction to a constraint satisfaction problem with bound tightening, solved incrementally. The support for two common aggregation functions, max and mean, is formulated here for the first time. Experiments on a range of datasets show that GNNEv can verify adversarial robustness of GNNs to attribute and structural perturbations. It outperforms state-of-the-art MIP-based baselines on node classification in both efficiency and functionality while remaining competitive on graph classification, and delivers strong performance on real-world fraud and biochemical datasets. One practical limitation of our approach is that the computational complexity grows quickly with the size of the fragile edge set. Future work will include mitigating the impact of the fragile edge sets on performance by combining heuristic search and distributed frameworks, and developing hybrid approaches to integrate probabilistic verification for improving scalability.

Acknowledgments. The authors would like to thank Michael Benedikt for the insightful comments and suggestions. This work was supported by the EPSRC Prosperity Partnership FAIR (grant number EP/V056883/1) and ELSA: European Lighthouse on Secure and Safe AI project (grant agreement No. 101070617 under UK guarantee). MK receives funding from the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115).

References

1. Akintunde, M.E., Kevorchian, A., Lomuscio, A., Pirovano, E.: Verification of RNN-based neural agent-environment systems. In: AAAI, pp. 6006–6013 (2019)
2. An, D., Zhang, H., Zhao, Q., Liu, J., Shi, J., Huang, Y., Yang, Y., Liu, X., Qin, S.: Graph convolutional network robustness verification algorithm based on dual approximation. In: ICFEM, pp. 146–161 (2024)
3. Banerjee, D., Xu, C., Singh, G.: Input-relational verification of deep neural networks. Proc. ACM Program. Lang. **8**(PLDI), 1–27 (2024)
4. Barceló, P., Kostylev, E.V., Monet, M., Pérez, J., Reutter, J.L., Silva, J.P.: The logical expressiveness of graph neural networks. In: ICLR (2020)

5. Barrett, C.W., Henzinger, T.A., Seshia, S.A.: Certificates in AI: Learn but verify. Commun. ACM **69**(1), 66–75 (2026)
6. Benedikt, M., Lu, C., Motik, B., Tan, T.: Decidability of graph neural networks via logical characterizations. In: ICALP, pp. 127:1–127:20 (2024)
7. Bojchevski, A., Günnemann, S.: Certifiable robustness to graph perturbations. In: NeurIPS, pp. 8317–8328 (2019)
8. Bojchevski, A., Klicpera, J., Günnemann, S.: Efficient robustness certificates for discrete data: Sparsity-aware randomized smoothing for graphs, images and more. In: ICML, pp. 1003–1013 (2020)
9. Bonaert, G., Dimitrov, D.I., Baader, M., Vechev, M.T.: Fast and precise certification of transformers. In: PLDI, pp. 466–481 (2021)
10. Boopathy, A., Weng, T., Chen, P., Liu, S., Daniel, L.: CNN-Cert: An efficient framework for certifying robustness of convolutional neural networks. In: AAAI, pp. 3240–3247 (2019)
11. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of ReLU-based neural networks via dependency analysis. In: AAAI, pp. 3291–3299 (2020)
12. Bronstein, M.M., Bruna, J., Cohen, T., Velickovic, P.: Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. CoRR **abs/2104.13478** (2021)
13. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: NeurIPS, pp. 4795–4804 (2018)
14. Cai, P., Wang, H., Sun, Y., Liu, M.: DiGNet: Learning scalable self-driving policies for generic traffic scenarios with graph neural networks. In: IROS, pp. 8979–8984 (2021)
15. Casas, S., Gulino, C., Liao, R., Urtasun, R.: SpAGNN: Spatially-aware graph neural networks for relational behavior forecasting from sensor data. In: ICRA, pp. 9491–9497 (2020)
16. Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Zhu, W., Huang, J.: A restricted black-box adversarial framework towards attacking graph embedding models. In: AAAI, pp. 3389–3396 (2020)
17. Chen, J., Zhang, J., Chen, Z., Du, M., Xuan, Q.: Time-aware gradient attack on dynamic network link prediction. IEEE Trans. Knowl. Data Eng. **35**(2), 2091–2102 (2023)
18. Cheung, K.K.H., Gleixner, A.M., Steffy, D.E.: Verifying integer programming results. In: IPCO, pp. 148–160 (2017)
19. Cohen, J., Rosenfeld, E., Kolter, J.Z.: Certified adversarial robustness via randomized smoothing. In: ICML, pp. 1310–1320 (2019)
20. Corso, G., Cavalleri, L., Beaini, D., Liò, P., Velickovic, P.: Principal neighbourhood aggregation for graph nets. In: NeurIPS (2020)
21. Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., Song, L.: Adversarial attack on graph structured data. In: ICML, pp. 1123–1132 (2018)
22. Dai, Q., Shen, X., Zhang, L., Li, Q., Wang, D.: Adversarial training methods for network embedding. In: WWW, pp. 329–339 (2019)

23. Dehmamy, N., Barabási, A., Yu, R.: Understanding the representation power of graph neural networks in learning graph topology. In: NeurIPS, pp. 15387–15397 (2019)
24. Du, T., Ji, S., Shen, L., Zhang, Y., Li, J., Shi, J., Fang, C., Yin, J., Beyah, R., Wang, T.: Cert-RNN: Towards certifying the robustness of recurrent neural networks. In: CCS, pp. 516–534 (2021)
25. Feng, F., He, X., Tang, J., Chua, T.: Graph adversarial training: Dynamically regularizing based on graph structure. IEEE Trans. Knowl. Data Eng. **33**(6), 2493–2504 (2021)
26. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. CoRR **abs/1903.02428** (2019)
27. Gao, C., Yin, S., Wang, H., Wang, Z., Du, Z., Li, X.: Medical-knowledge-based graph neural network for medication combination prediction. IEEE Trans. Neural Networks Learn. Syst. **35**(10), 13246–13257 (2024)
28. Geisler, S., Schmidt, T., Şirin, H., Zügner, D., Bojchevski, A., Günnemann, S.: Robustness of graph neural networks at scale. In: NeurIPS, pp. 7637–7649 (2021)
29. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: ICML, pp. 1263–1272 (2017)
30. Gosch, L., Geisler, S., Sturm, D., Charpentier, B., Zügner, D., Günnemann, S.: Adversarial training for graph neural networks: Pitfalls, solutions, and new directions. In: NeurIPS (2023)
31. Haghshenas, S.H., Hasnat, A., Naeini, M.: A temporal graph neural network for cyber attack detection and localization in smart grids. In: ISGT, pp. 1–5 (2023)
32. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NeurIPS, pp. 1024–1034 (2017)
33. Hojny, C., Zhang, S., Campos, J.S., Misener, R.: Verifying message-passing neural networks via topology-based bounds tightening. In: ICML (2024)
34. Huang, P., Wei, D., Isac, O., Wu, H., Wu, M., Barrett, C.: Parameterized abstract interpretation for transformer verification. In: AAAI (2026)
35. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV, pp. 3–29 (2017)
36. Jha, K., Saha, S., Singh, H.: Prediction of protein–protein interaction using graph neural networks. Scientific Reports **12**(1), 8360 (2022)
37. Jin, H., Shi, Z., Peruri, V.J.S.A., Zhang, X.: Certified robustness of graph convolution networks for graph classification under topological attacks. In: NeurIPS (2020)
38. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: CAV, pp. 97–117 (2017)
39. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
40. Ko, C., Lyu, Z., Weng, L., Daniel, L., Wong, N., Lin, D.: POPQORN: Quantifying robustness of recurrent neural networks. In: ICML, pp. 3468–3477 (2019)

41. Ladner, T., Eichelbeck, M., Althoff, M.: Formal verification of graph convolutional networks with uncertain node features and uncertain graph structure. Trans. Mach. Learn. Res. **2025** (2025)
42. Lai, Y., Zhu, Y., Pan, B., Zhou, K.: Node-aware bi-smoothing: Certified robustness against graph injection attacks. In: IEEE SP, pp. 2958–2976 (2024)
43. Li, J., Peng, J., Chen, L., Zheng, Z., Liang, T., Ling, Q.: Spectral adversarial training for robust graph neural network. IEEE Trans. Knowl. Data Eng. **35**(9), 9240–9253 (2023)
44. Li, J., Xie, T., Chen, L., Xie, F., He, X., Zheng, Z.: Adversarial attack on large scale graph. IEEE Trans. Knowl. Data Eng. **35**(1), 82–95 (2023)
45. Li, M.M., Huang, K., Zitnik, M.: Graph representation learning in biomedicine and healthcare. Nat. Biomed. Eng. **6**(12), 1353–1369 (2022)
46. Liu, M., Lu, C., Kwiatkowska, M.: Exact verification of graph neural networks with incremental constraint solving. CoRR **abs/2508.09320** (2025)
47. Lu, C., Tan, T., Benedikt, M.: Robustness verification of graph neural networks via lightweight satisfiability testing. CoRR **abs/2510.18591** (2025)
48. Lu, J., Kumar, M.P.: Neural network branching for neural network verification. In: ICLR (2020)
49. Marzari, L., Cicalese, F., Farinelli, A.: Probabilistically tightened linear relaxation-based perturbation analysis for neural network verification. J. Artif. Intell. Res. **84** (2025)
50. McAuley, J.J., Leskovec, J.: From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews. In: WWW, pp. 897–908 (2013)
51. Mohammadinejad, S., Paulsen, B., Deshmukh, J.V., Wang, C.: DiffRNN: Differential verification of recurrent neural networks. In: FORMATS, pp. 117–134 (2021)
52. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: TUDataset: A collection of benchmark datasets for learning with graphs. In: ICML 2020 Workshop on Graph Representation Learning and Beyond (2020)
53. Motie, S., Raahemi, B.: Financial fraud detection using graph neural networks: A systematic review. Expert Syst. Appl. **240**, 122156 (2024)
54. Mu, J., Wang, B., Li, Q., Sun, K., Xu, M., Liu, Z.: A hard label black-box adversarial attack against graph neural networks. In: CCS, pp. 108–125 (2021)
55. Nunn, P., Sälzer, M., Schwarzentruber, F., Troquard, N.: A logic for reasoning about aggregate-combine graph neural networks. In: IJCAI, pp. 3532–3540 (2024)
56. Osselin, P., Kenlay, H., Dong, X.: Structure-aware robustness certificates for graph classification. In: UAI, pp. 1596–1605 (2023)
57. Rayana, S., Akoglu, L.: Collective opinion spam detection: Bridging review networks and metadata. In: KDD, pp. 985–994 (2015)
58. Reiser, P., Neubert, M., Eberhard, A., Torresi, L., Zhou, C., Shao, C., Metni, H., van Hoesel, C., Schopmans, H., Sommer, T., Friederich, P.: Graph neural networks for materials science and chemistry. Commun. Mater. **3**(1), 93 (2022)

59. Rosenbluth, E., Tönshoff, J., Grohe, M.: Some might say all you need is sum. In: IJCAI, pp. 4172–4179 (2023)
60. Rossi, F., Van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier (2006)
61. Salman, H., Yang, G., Zhang, H., Hsieh, C., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: NeurIPS, pp. 9832–9842 (2019)
62. Sälzer, M., Lange, M.: Fundamental limits in formal verification of message-passing neural networks. In: ICLR (2023)
63. Scholten, Y., Schuchardt, J., Geisler, S., Bojchevski, A., Günnemann, S.: Randomized message-interception smoothing: Gray-box certificates for graph neural networks. In: NeurIPS (2022)
64. Schönherr, M., Lutz, C.: Logical characterizations of GNNs with mean aggregation. CoRR **abs/2507.18145** (2025)
65. Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. AI Mag. **29**(3), 93–106 (2008)
66. Shi, Z., Zhang, H., Chang, K., Huang, M., Hsieh, C.: Robustness verification for transformers. In: ICLR (2020)
67. Tao, S., Cao, Q., Shen, H., Huang, J., Wu, Y., Cheng, X.: Single node injection attack against graph neural networks. In: CIKM, pp. 1794–1803 (2021)
68. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (2019)
69. Tran, H., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using ImageStars. In: CAV, pp. 18–42 (2020)
70. Wang, B., Jia, J., Cao, X., Gong, N.Z.: Certified robustness of graph neural networks against adversarial structural perturbation. In: KDD, pp. 1645–1653 (2021)
71. Wang, D., Qi, Y., Lin, J., Cui, P., Jia, Q., Wang, Z., Fang, Y., Yu, Q., Zhou, J., Yang, S.: A semi-supervised graph attentive network for financial fraud detection. In: ICDM, pp. 598–607 (2019)
72. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: NeurIPS, pp. 6369–6379 (2018)
73. Weng, L., Chen, P., Nguyen, L.M., Squillante, M.S., Boopathy, A., Oseledets, I.V., Daniel, L.: PROVEN: Verifying robustness of neural networks with a probabilistic approach. In: ICML, pp. 6727–6736 (2019)
74. Weng, T., Zhang, H., Chen, H., Song, Z., Hsieh, C., Daniel, L., Boning, D.S., Dhillon, I.S.: Towards fast computation of certified robustness for ReLU networks. In: ICML, pp. 5273–5282 (2018)
75. Wu, H., Wang, C., Tyshetskiy, Y., Docherty, A., Lu, K., Zhu, L.: Adversarial examples for graph data: Deep insights into attack and defense. In: IJCAI, pp. 4816–4823 (2019)
76. Xia, Z., Yang, H., Wang, B., Jia, J.: GNNCert: Deterministic certification of graph neural networks against adversarial perturbations. In: ICLR (2024)
77. Xu, K., Chen, H., Liu, S., Chen, P., Weng, T., Hong, M., Lin, X.: Topology attack and defense for graph neural networks: An optimization perspective. In: IJCAI, pp. 3961–3967 (2019)

78. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: ICLR (2019)
79. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.: Automatic perturbation analysis for scalable certified robustness and beyond. In: NeurIPS (2020)
80. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: KDD, pp. 974–983 (2018)
81. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: NeurIPS, pp. 4944–4953 (2018)
82. Zhang, S., Campos, J.S., Feldmann, C., Walz, D., Sandfort, F., Mathea, M., Tsay, C., Misener, R.: Optimizing over trained GNNs via symmetry breaking. In: NeurIPS (2023)
83. Zhang, X., Zitnik, M.: GNNGuard: Defending graph neural networks against adversarial attacks. In: NeurIPS (2020)
84. Zhang, Z., Jia, J., Wang, B., Gong, N.Z.: Backdoor attacks to graph neural networks. In: SACMAT, pp. 15–26 (2021)
85. Zhou, K., Michalak, T.P., Waniek, M., Rahwan, T., Vorobeychik, Y.: Attacking similarity-based link prediction in social networks. In: AAMAS, pp. 305–313 (2019)
86. Zou, X., Zheng, Q., Dong, Y., Guan, X., Kharlamov, E., Lu, J., Tang, J.: TDGIA: Effective injection attacks on graph neural networks. In: KDD, pp. 2461–2471 (2021)
87. Zügner, D., Akbarnejad, A., Günnemann, S.: Adversarial attacks on neural networks for graph data. In: KDD, pp. 2847–2856 (2018)
88. Zügner, D., Günnemann, S.: Adversarial attacks on graph neural networks via meta learning. In: ICLR (2019)
89. Zügner, D., Günnemann, S.: Certifiable robustness and robust training for graph convolutional networks. In: KDD, pp. 246–256 (2019)
90. Zügner, D., Günnemann, S.: Certifiable robustness of graph convolutional networks under structure perturbations. In: KDD, pp. 1656–1665 (2020)