Toward the Analysis of Graph Neural Networks

Thanh-Dat Nguyen*
thanhdatn@student.unimelb.edu.au
University of Melbourne
Melbourne, Victoria, Australia

Thanh Le-Cong*
thanh.ld164834@sis.hust.edu.vn
HUST
Hanoi, Vietnam

ThanhVu H. Nguyen
tvn@gmu.edu
George Mason University
Fairfax, Virginia, USA

Xuan-Bach D. Le
bach.le@unimelb.edu.au
University of Melbourne
Melbourne, Victoria, Australia

Quyet-Thang Huynh
thanghq@soict.hust.edu.vn
HUST
Hanoi, Vietnam

ABSTRACT

Graph Neural Networks (GNNs) have recently emerged as an effective framework for representing and analyzing graph-structured data. GNNs have been applied to many real-world problems such as knowledge graph analysis, social networks recommendation, and even COVID-19 detection and vaccine development. However, unlike other deep neural networks such as Feedforward Neural Networks (FFNNs), few verification and property inference techniques exist for GNNs. This is potentially due to dynamic behaviors of GNNs, which can take arbitrary graphs as input, whereas FFNNs which only take fixed size numerical vectors as inputs.

This paper proposes GNN-Infer, an approach to analyze and infer properties of GNNs by extracting influential structures of the GNNs and then converting them into FFNNs. This allows us to leverage existing powerful FFNNs analyses to obtain results for the original GNNs. We discuss various designs of GNN-Infer to ensure the scalability and accuracy of the conversions. We also illustrate GNN-Infer on a study case of node classification. We believe that GNN-Infer opens new research directions for understanding and analyzing GNNs.

KEYWORDS

Property Inference, Graph Neural Networks, Neural Network Conversion

ACM Reference Format:
Thanh-Dat Nguyen, Thanh Le-Cong, ThanhVu H. Nguyen, Xuan-Bach D. Le, and Quyet-Thang Huynh. 2022. Toward the Analysis of Graph Neural Networks. In *New Ideas and Emerging Results (ICSE-NIER'22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3510455.3512780

*Both authors contributed equally to this research.

ICSE-NIER'22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9224-2/22/05…$15.00
https://doi.org/10.1145/3510455.3512780

1 INTRODUCTION

Deep Neural Networks (DNNs) have emerged as one of the most effective and modern approaches in solving real-world problems. DNNs have been used to solve common problems such as movies recommendations, image recognition to important problems such as airplane collision control and "fake" news and information detection. Just like software, DNN models can be misused and attacked (e.g., small perturbations to the inputs can result in misclassification [13, 18, 26]). Thus, over the last decade, researchers have developed property inference [6], formal [9, 10] and robustness [4] verification techniques to analyze DNNs, e.g., verifying that for a specific input region, a DNN will result in a specific classification, and more recently, inferring properties or facts [6] to help explain behaviors of a DNN, which is typically treated as a blackbox.

Despite the proliferation of DNNs analyses, most effective ones focus on certain types of DNNs, such as Feed-forward Neural Network (FFNNs) [9], which have fixed structure and fixed-size vectors of numbers as inputs. One of the more complicated DNNs that has recently been used in practice is Graph Neural Networks (GNNs). GNNs take inputs as *graphs* of various sizes (even each of the nodes in the graph is attached with information encoded as a vector of numbers) and have dynamic computations (i.e., dynamic computation graphs) depending on the structure and information from the input graphs. GNNs have been applied to solve many practical problems, e.g., knowledge graphs analysis [15], recommendation system for social networks [19], chemical and protein classification [5, 12], reasoning the structure of graphics and images [16], and even advanced COVID-19 detection [14, 24] and vaccine development [2, 7, 22].

Similar to standard DNNs, complex GNNs are often used as a blackbox and can be vulnerable to adversary attacks, hightlighting the concerns about safety, fairness, and privacy of the GNNs [3, 17, 23]. For example, a new COVID vaccine developed by unknown and attacked-prone ML techniques can only further increase doubts and hesitancy from the public. However, unlike popular FFNNs, in which there are many effective formal analyses, to the best of our knowledge, few techniques exist for GNNs, potentially due to the vast differences between the two types of networks.

In this paper, we propose GNN-Infer, an approach to analyze GNNs, both in verification and property inference, by converting GNNs to FFNNs and reusing developed techniques for FFNNs. While analyses explicitly designed for GNNs can be more efficient, they can also be more difficult and time-consuming to develop due to the differences between two types of networks. Thus, we believe

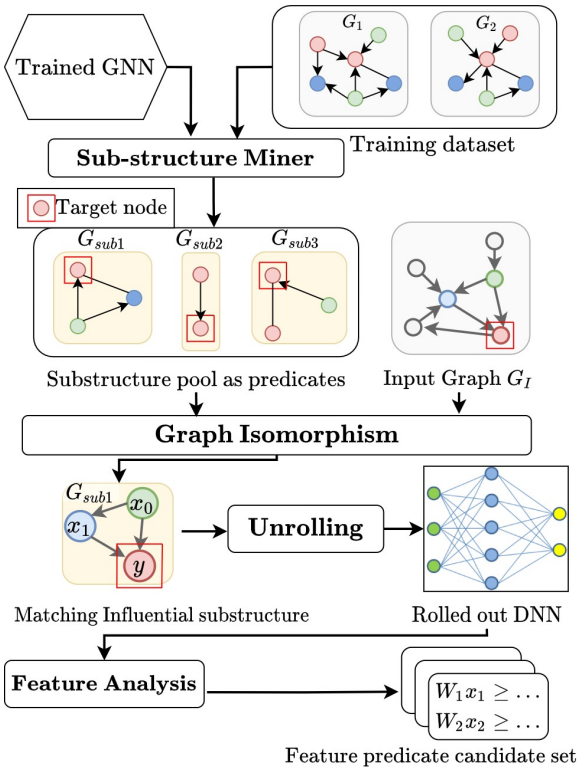

Figure 1: Overview of GNN-Infer

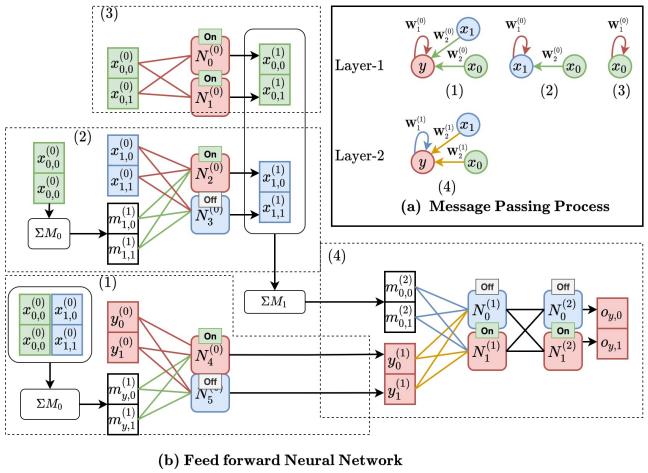

Figure 2: GNN message passing and unrolling

our approach of leveraging existing efficient techniques and tools can be achieved quicker and also as effective (e.g., by using existing powerful FFNN tools). Indeed, the GNN-Infer approach is similar to techniques in software engineering and formal verification that encode the analysis task as a logical formula that can be efficiently analyzed by existing constraint solving techniques and tools (e.g., SAT and SMT solvers).

Fig. 1 gives an overview of GNN-Infer. The main challenge in analyzing GNNs and converting them to FFNNs is that the input graphs of a GNN can have various topological structures and the GNN itself also has a dynamic computation graph depending on its input graphs. To solve this challenge, we first mine influential substructures of input graphs to summarize the structural input space of a GNN. Then for each substructure, which represents a class of input graphs, we "unroll" the structure to create an equivalent FFNN for each update operation of the GNN, and then combine these FFNNs into a final FFNN representing the original GNN operating over input graphs captured by the substructure. Finally, we extend the existing DNN analyses to the FFNN of each substructure and obtain results for the original GNN.

2 TECHNICAL APPROACH

Existing verification techniques for DNNs check that the DNN satisfies a user-supplied property (e.g., a certain range over inputs results in a certain output). In contrast, a property inference technique aims to automatically infer such properties from the DNN. In both cases, the property to be verified or inferred has the form `pre` $\Longrightarrow$ `post`, where `pre` is a condition over the inputs and `post` is certain requirement on the outputs. GNN-Infer aims to verify and infer the `pre` condition for some specific `post` condition, e.g., we want to find input condition that causing the nerual network to classify input images as "dog".

To illustrate GNN-Infer, we consider GNN models for the standard problem of graph node classification, which takes as input a graph G and gives a classification c for each node $v \in G$. For such GNNs, the `pre` are *input properties*, which are logical *predicates* capturing common structures and features[1] of the input graphs that lead to a certain classification of a target node. Below we use a concrete example given in Fig. 2 to describe GNN-Infer.

2.1 Substructure Mining

Unlike an FFNN, a GNN does not have a fixed structure: it can take arbitrary graphs and the behavior of GNN itself also changes depending on the structure of the inputs (e.g., the influence of a node depends on its neighbors). Thus, the direct, naïve way of converting a GNN to an FFNN does not scale as it would result in a different FFNN for each different input graph, and the FFNN can also be large if the input graph is large.

To solve this challenge, GNN-Infer creates FFNNs that support *classes* of input graphs. We leverage existing works in network graphs and GNNs such as GNNExplainer and PGExplainer [11, 20] to mine common and influential *substructures* from sample input graphs. These substructures are subgraphs that contain nodes, edges, and features that likely affect the outcome of target nodes' predictions. Importantly, these substructures are compact, which are crucial for achieving FFNNs with manageable sizes.

Fig. 1 illustrates how GNN-Infer mines influential substructures (*sub-structure miner*). Given a trained GNN model and a set of input graphs that have the desired classification, we use an existing

[1]Node features are attributes of nodes, e.g., if we take a node "professor" in academic graph, its attributes may be "name", "citations", "affiliation", and encoded as a numerical vector such as $\{0.1, 0.3, 0.4, 0.5\}$.

tool such as GNNExplainer to extract influential substructures. For example, from the two input graphs G_1 and G_2 in Fig. 1, GNNExplainer would extract three substructures $G_{sub1}, G_{sub2}, G_{sub3}$ that are common from the input graphs and contribute significantly in the prediction of the target red-color node.

GNN-Infer thus focuses on analyzing GNNs over input graphs in existing training samples. The higher quality the samples we have, the better set of influential substructures we learn—this leads to larger and more accurate classes of supported graphs. Just as with most DNNs, sources of obtaining training samples vary, e.g., public benchmarks. If available, we can also reuse input graphs that were used to train the GNN models we are analyzing.

2.2 Structure Predicates and Graph Isomorphism

Many existing DNNs and program analyses encode problems into logical formulae that can be reasoned about using constraint solving. Here, we encode the obtained graph substructures as logical predicates σ_{struct}, so that we can leverage existing automating reasoning tools such as SAT and SMT solvers. An important use for these *structure predicates* is to check if an input graph contains the considered substructures. If it does, we are confident that our approach and result will hold; and if it does not, we can support it by adding it to our training data to learn about its influential substructures. These predicates are also a crucial part of the inferred properties that help explain the behaviors of the GNN to the user.

To determine if an input graph satisfies a substructure, we check if the graph and the substructure, which is also a graph, is *isomorphic*. By adapting existing work such as CFL-match [1], we can apply logical reasoning over the obtained structure predicates to check if there exist a mapping from the querying substructure graph to some subgraph of input graph that would make both graphs isomorphic.

Fig. 1 shows these steps. For each obtained substructure, we create a predicate capturing that structure by using standard graph isomorphic checking. As an example structure predicate ensuring present of G_{sub1} which has three nodes x_0 (green), x_1 (blue) and y (red) to be matched to input graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, can has the following form (note that we have omitted node-label checking for readability):

$$\begin{aligned}\sigma_{struct,sub1}(V, E) = \exists x_0, x_1, y \in V :\\ \{(x_0, x_1), (x_0, x_1), (x_0, y)\} \subseteq E\end{aligned} \tag{1}$$

Concerning the implementation side, this can be done with existing analysis tools by transforming from a graph problem to a satisfiability problem (e.g., nodes represented as boolean variables and edges as logical connections among variables). Notice if we perform graph isomorphism checking on some input graph such as G_1 in the Figure 1, we will see that it is isomorphic to the predicate of substructure G_{sub1} because they share the same substructure.

Obviously, all trained input sets would be isomorphic to at least one of the structure predicates.

2.3 GNN Unrolling

After obtaining influential substructures and their corresponding substructural predicates, we can now create an FFNN to represent the GNN model. As it turns out, it is actually straightforward to convert a GNN with a fixed substructure directly to an FFNN. We assume our GNN uses the the popular *message passing process*[5] adopted in most types of GNNs. This process works by updating the value of a node in the graph based on the information of its neighboring nodes. Then, to create an FFNN from a GNN with a set of substructures, we essentially create a FFNN to simulate how message passing is done on a substructure using the "unroll" technique similar to one introduced in [8] for RNN unrolling. Finally, we combine all FFNNs to obtain a final FFNN representing the original GNN (that supports graph inputs isomorphic to the considered substructures).

Figure 2 shows how message passing works on the substructure G_{sub1} obtained in Fig. 1. Again, G_{sub1} consists of 3 nodes x_0, x_1 and y, for illustration purposes, we use a GNN with 2-layer and the weight W to represent the values of features of each node. For layer 1, the GNN contains three message passing processes labelled (1), (2), (3) that correspond to the three nodes y, x_1 , and x_0. The results of these message passing processes are updated as newly computed node features of y, x_1 and x_0, which are used for next layer. For final layer 2, we only need to consider target node y's message passing from the result of (1), (2) (3), followed by a simple linear transformation and we have a message processing process labeled (4) in Figure 2.

Now, we unroll each message passing process in layer i of the GNN into a corresponding i-layer FFNN. For the three message-passing processes in Layer 1 in Figure 2a, we obtain the three 1-layer FFNNs shown in Figure 2b and for the message passing process in layer 2 in Figure 2a, we have the 2-layer FFNN shown in Figure 2b (4). Finally, we connect these individual FFNNs to construct a final (large) FFNN as shown in Figure 2b to represent the original GNN.

Using Existing FFNN analyses. We can apply existing analyses for FFNNs to our rolled out FFNN. For instance, we can apply the Prophecy tool [6] to infer properties for FFNNs. This work derives predicates over the inputs of an FFNN, which convex regions over inputs values, that map to a desired output classification. We can also apply FFNN verification tools such as Marabou (the successor of the popular Reluplex work [9]) to check if an inferred or user-supplied property is correct.

For the running example in Figure 2, given some specific weights in Figure 2a, running Prophecy on the resulting FFNN can gives the following predicates representing a convex region over the inputs space that result in the desired classification of the target node in the GNN. Here the input $x_{i,j}$ of the FFNN represents the feature j of node x_i of the GNN.

$$\begin{aligned}\sigma_{inps} = &(x_{0,0} + x_{1,0} - x_{0,1} - x_{1,1} > 0)\\ &\wedge(x_{0,0} + x_{1,0} - 2x_{0,1} - 2x_{1,1} \leq 0)\\ &\wedge(x_{0,0} - x_{0,1} > 0) \wedge (x_{0,0} - 2x_{0,1} > 0)\\ &\wedge(x_{0,0} + x_{1,0} + x_{2,0} - x_{0,1} - x_{1,1} - x_{2,1} > 0)\\ &\wedge(x_{0,1} + x_{1,1} + x_{2,1} - 2x_{0,0} - 2x_{1,0} - 2x_{2,0} \leq 0)\end{aligned} \tag{2}$$

2.4 Equivalent Analysis

Ideally, the mined influential substructures truly represent the behaviors of the considered GNN and the obtained FFNN is thus equivalent to the GNN. In practice, this does not happen as many

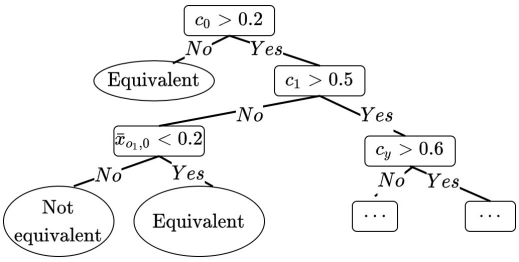

Figure 3: Example of using decision tree to predict whether substructure computation will be equivalent to the input graphs

nodes, especially those that are not part of the influential substructures but are neighbors with those in the substructure, can influence the final GNN classification result. Thus, we want to analyze how these neighboring nodes can directly affect those in the influential substructures and thus the final result.

Our experiences with GNNs show that a node in a influential substructure is affected by their "outside" neighbors (those that are not in the substructures) in two ways: the number of outside neighbors it has comparing to its total number of neighbors (**connectivity ratio**) and the **mean contribution** of the outside neighbors.

Given this knowledge, we compute additional conditions over substructures to make them represent the GNN more accurately. To do this, we use decision trees to compute predicates over the two features representing connectivity and mean contribution. We split the input graphs (e.g., used in the beginning to obtain substructures) into those that are and are not isomorphic to the substructures. With respect to each influential substructure, we collect the supporting input graph set from the training dataset. Following this, for each input in the supporting graph set, we perform two predictions of target node y's output: 1) using only the influential substructure and 2) using the full input graph. We collect statistics on **connectivity ratio** and **mean contribution** to predict whether the output on target node y remains the same throughout two scenario.

Using this set of training data, we can leverage decision tree to determine conditions over the two features representing connectivity ratio and mean contribution that lead to equivalent or non-equivalent classification. Each paths in the tree represents an additional predicate that can help strengthen the substructures, allowing them to represent the GNN more accurately.

For example, the decision tree in Fig. 3 produces several predicates such as

$$\sigma_{feat\_equiv} := c_0 > 0.2 \wedge c_1 \leq 0.5 \wedge \bar{x}_{o_1,0} < 0.2 \tag{3}$$

which says that node 0 with connectivity ratio > 0.2 , node 1 with connectivity ratio <= 0.5, and the mean contribution of feature 0 in node $0 \geq 0.2$ are likely needed as conjunction for the feature predicate on the substructure holds for all graphs. Thus, this approach allows us to obtain a set σ_{inps} of predicates to strengthen the substructure predicates, ensuring they represent the GNN more accurately.

2.5 GNN Property

Finally, our approach produces a *property* σ of a given GNN in form

$$\sigma_{struct}\sigma \wedge \sigma_{inps} \wedge \sigma_{feat\_equiv} \implies Q,$$

where σ_{struct} is the predicate capturing graph isomorphism (Section 2.2), σ_{inp} are input properties of the converted FFNN (Section 2.3), σ_{feat_equiv} is the additional constraints helping the FFNN more accurate to the original GNN (Section 2.4), and Q is the output property of some target node y (e.g. $o_{y,1} < o_{y,2}$). This means for an an input graph with target node y that is isomorphic to some of the mined substructure (satisfies σ_{struct}), has certain requirements about neighboring substructure nodes (satisfies σ_{feat_equiv}), with node features lie within certain regions (satisfies σ_{inps}), then this graph will have the property Q on its target node.

3 FUTURE PLAN

Currently, we only have applied our ideas on several small examples by hand. We are implementing these ideas and evaluate the approach with existing GNN benchmarks.

We anticipate several challenges that would arise in this direction. First, for complex GNNs, the converted FFNNs might be too too large and contain non-trivial, e.g., nonlinear-arithmetic. These would give difficulties to standard FFNN verification tools such as Reluplex or Marabou. Second, we use sample inputs to mine substructures and feature predicates, and thus can obtain inaccurate results. While we might be able to obtain groundtruths or manually check results of small GNNs, we will not have an effective way to formally verify our results on complex and real-world GNNs. Third, obtaining realistic benchmarks for GNNs might be more difficult as they are not as readily available and well-studied as benchmarks of FFNNs. However, we can start with existing dataset from the literature such as those from [16] for autnomous driving and [5, 25] for drug interactions. Moreover, we plan to use GNN-Infer's inferred properties to analyze adversarial attacks and thus can compare our work with existing work on GNN adversarial attacks (e.g. Nettack, GNN Meta Attack [26–28]) and adversarial defense such as GNNGuard [21].

Finally, there is always a chance that the proposed approach does not work well in practice, e.g., GNN-Infer does not scale or becomes too inaccurate for converting complex GNNs. It might be that designing algorithms directly to solve GNNs would give more benefits in the long run. However, as with any research problems, especially new ones with few existing attempts, we have to start somewhere, and converting it to something we already know how to solve seems to be a good place to start.

4 ACKNOWNLEDGEMENT

We thank the anonymous reviewers for helpful comments. This material is based in part upon work supported by the Australian Research Council's Discovery Early Career Researcher Award under project number DE220101057 and the National Science Foundation under grant numbers 1948536 and 2107035.

REFERENCES

[1] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient subgraph matching by postponing Cartesian products. *Proceedings of the ACM*

SIGMOD International Conference on Management of Data 26-June-20 (2016), 1199–1214.

[2] Mark Cheung and José MF Moura. 2020. Graph Neural Networks for COVID-19 Drug Discovery. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 5646–5648.

[3] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. In *International conference on machine learning*. PMLR, 1115–1124.

[4] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18. https://doi.org/10.1109/SP.2018.00058

[5] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. (4 2017).

[6] Divya Gopinath, Hayes Converse, Corina S. Pasareanu, and Ankur Taly. 2019. Property Inference For Deep Neural Networks. *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (4 2019), 797–809.

[7] Kang-Lin Hsieh, Yinyin Wang, Luyao Chen, Zhongming Zhao, Sean Savitz, Xiaoqian Jiang, Jing Tang, and Yejin Kim. 2020. Drug repurposing for covid-19 using graph neural network with genetic, mechanistic, and epidemiological validation. *Research Square* (2020).

[8] Yuval Jacoby, Clark Barrett, and Guy Katz. 2020. Verifying Recurrent Neural Networks using Invariant Inference. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12302 LNCS (4 2020), 57–74.

[9] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[10] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 443–452.

[11] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized Explainer for Graph Neural Network. *Advances in Neural Information Processing Systems* 33 (2020).

[12] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. 2019. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. (11 2019).

[13] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. 2020. Adversarial Attacks and Defenses in Deep Learning. *Engineering* 6, 3 (mar 2020), 346–360.

[14] Pritam Saha, Debadyuti Mukherjee, Pawan Kumar Singh, Ali Ahmadian, Massimiliano Ferrara, and Ram Sarkar. 2021. GraphCovidNet: A graph neural network based model for detecting COVID-19 from CT scans and X-rays of chest. *Scientific Reports* 11, 1 (2021), 1–16.

[15] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. 2020. Multi-Task Learning for Dense Prediction Tasks: A Survey. (2020), 1–20.

[16] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. 2018. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6857–6866.

[17] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. 2019. Adversarial Examples for Graph Data: Deep Insights into Attack and Defense. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 4816–4823.

[18] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural Attack for Pre-trained Models of Code. In *Proceedings of the ACM/IEEE 44nd International Conference on Software Engineering* (Pittsburgh, USA) *(ICSE '22)*. Association for Computing Machinery, 12 pages. https://doi.org/10.1145/3510003.3510146

[19] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[20] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNNExplainer: Generating Explanations for Graph Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc.

[21] Xiang Zhang and Marinka Zitnik. 2020. GNNGuard: Defending Graph Neural Networks against Adversarial Attacks. *Advances in Neural Information Processing Systems* 2020-Decem, NeurIPS (jun 2020). arXiv:2006.08149

[22] Yadi Zhou, Fei Wang, Jian Tang, Ruth Nussinov, and Feixiong Cheng. 2020. Artificial intelligence in COVID-19 drug repurposing. *The Lancet Digital Health* (2020).

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

[24] Shixiang Zhu, Alexander Bukharin, Liyan Xie, Mauricio Santillana, Shihao Yang, and Yao Xie. 2021. High-resolution Spatio-temporal Model for County-level COVID-19 Activity in the US. *ACM Transactions on Management Information Systems (TMIS)* 12, 4 (2021), 1–20.

[25] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34, 13 (2018), i457–i466.

[26] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Vol. 2019-Augus. ACM, New York, NY, USA, 2847–2856.

[27] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Vol. 2019-Augus. ACM, New York, NY, USA, 2847–2856. https://doi.org/10.1145/3219819.3220078

[28] Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. *7th International Conference on Learning Representations, ICLR 2019* (2019), 1–15. arXiv:1902.08412