Node Injection Link Stealing Attack

Oualid Zari
Eurecom
Biot, France
oualid.zari@eurecom.fr

Javier Parra-Arnau
Karlsruhe Institute of Technology
Karlsruhe, Germany
javi.parra-arnau@kit.edu

Ayşe Ünsal
Eurecom
Biot, France
ayse.unsal@eurecom.fr

Melek Önen
EURECOM
Biot, France
melek.onen@eurecom.fr

***Abstract*—In this paper, we present a stealthy and effective attack that exposes privacy vulnerabilities in Graph Neural Networks (GNNs) by inferring private links within graph-structured data. Focusing on dynamic GNNs, we propose to inject new nodes and attach them to a particular target node to infer its private edge information. Our approach significantly enhances the F_1 score of the attack beyond the current state-of-the-art benchmarks. Specifically, for the Twitch dataset, our method improves the F_1 score by 23.75%, and for the Flickr dataset, it records a remarkable improvement, where the new performance is more than three times better than the state-of-the-art. We also propose and evaluate defense strategies based on differentially private (DP) mechanisms relying on a newly defined DP notion, which, on average, reduce the effectiveness of the attack by approximately 71.9% while only incurring a minimal average utility loss of about 3.2%.**

I. Introduction

Graph-structured data has become increasingly prevalent, especially in social networks, biological systems, and recommendation engines. Graph Neural Networks (GNNs) have emerged as powerful tools for analyzing such data, providing remarkable performance in various tasks. However, these advantages come with significant privacy risks as the graph structure often contains sensitive information. For instance, links in social networks might reveal users' interests, beliefs, or personal attributes, potentially causing serious privacy breaches [1].

This paper advances the understanding of edge privacy in GNNs by developing a novel link-stealing attack, named Node Injection Link Stealing (NILS) attack, and proposing a tailored Differential Privacy notion to protect against it. We focus on training GNNs for node classification tasks, where the model processes graph structure and node features to produce class membership predictions at inference time.

Previous studies like the Linkteller attack [2] showed that probing node features and analyzing GNN outputs can reveal graph links. We propose a stronger adversary who enhances this approach by adding new nodes and querying the model with malicious input features, aiming to infer and steal graph connections.

The NILS attack mimics sending a friend request on social media to uncover and analyze a target's connections, exploiting changes in content recommendations or interactions upon establishing new connections. We explore defense strategies, mainly DP mechanisms, proposing a new privacy notion, *one-node-one-edge privacy*, to counter such attacks.

We make the following contributions: We propose the NILS attack for inferring private links by injecting new nodes, link them to target nodes, and analyze changes in GNN output. Our evaluations demonstrate superior performance over existing methods like LinkTeller [2] and link-stealing [3]. We also introduce a new privacy notion and evaluate DP-based defenses to balance privacy preservation and model utility.

For a more comprehensive discussion and additional results, see the extended version of this paper on arXiv: https://arxiv.org/abs/2307.13548.

A. Background on Graph Neural Networks

GNNs [4] have emerged as a powerful class of machine learning models specifically designed to handle graph-structured data. They have gained considerable attention due to their ability to effectively learn and capture complex patterns in graph data, showing significant performance across a wide range of tasks, such as node classification [5], [6], link prediction [7], and graph classification [8], [9]. A particular focus of the current paper is the task of node classification, where the objective is to assign labels to individual nodes based on their features and the overall graph structure.

A graph $G = (V, E)$ is defined as a collection of nodes V and edges E. Nodes represent data points such as users in social networks or proteins in biological networks, while edges represent relationships or interactions between the nodes. Graphs can be represented using an adjacency matrix $A \in \mathbb{R}^{n\times n}$, where $n = |V|$ is the number of nodes in the graph, and $A_{ij} = 1$ if there exists an edge between nodes i and j, and $A_{ij} = 0$ otherwise. Additionally, nodes exhibit a set of features, which can be represented by vectors containing d elements. In social networks, these features may include demographic information such as age, gender, and location, as well as user interests and preferences. To capture these features, the feature matrix $X \in \mathbb{R}^{n\times d}$ includes essential information about the characteristics of each node in the graph.

GNNs primarily operate by employing a message-passing mechanism [4] that allows nodes to exchange and aggregate information from their local neighborhoods. This iterative process helps GNNs capture local and global structural information in the graph. For instance, in the context of graph convolutional networks (GCNs) [10], the most representative and well-established GNN models, their core architecture consists of a series of graph convolutional layers, which can be formulated as follows:

$$H^{(0)} = X, \quad H^{(l+1)} = \sigma\left(\hat{A}H^{(l)}W^{(l)}\right), \quad H^{(L)} = P. \quad (1)$$

Here, $H^{(0)}$ denotes the node feature matrix X; $H^{(l)} \in \mathbb{R}^{n\times d_l}$ is the hidden node representation matrix at layer l, where L is the total number of layers; and $P \in \mathbb{R}^{n\times\ c}$ represents the prediction scores for each potential class or label associated with the queried nodes, where c reprensents the number of classes; $W^{(l)} \in \mathbb{R}^{d_l\times d_{l+1}}$ is the learnable weight matrix for layer l; $\sigma(\cdot)$ is an activation function (e.g., ReLU), and $\hat{A}$ is a normalized adjacency matrix.

GNNs usually handle dynamic graph data as in real-life scenarios such as social network applications or recommendation systems, where graphs usually evolve over time: New nodes or edges may be introduced with the goal of making predictions for these actual nodes. When a new node is added to the graph, both the adjacency matrix $A \in \mathbb{R}^{n\times n}$, and the feature matrix $X \in \mathbb{R}^{n\times d}$ are updated. The adjacency matrix expands to $A' \in \mathbb{R}^{(n+1)\times(n+1)}$, while the feature matrix becomes $X' \in \mathbb{R}^{(n+1)\times d}$, incorporating the new node's connections and

features, respectively. Once the graph is updated, the GNN performs inference on the modified graph, using the message-passing mechanism described earlier.

II. Node injection link stealing attack

A. Node injection link stealing attack

In this paper, we focus on a particular attack named as *link stealing attack*, where an adversary without access to the adjacency matrix aims to learn whether a particular edge exists or not. Here, we consider a GNN application in which a server has already trained the GNN using a specific dataset and offers access to this GNN through a black-box API. In this context, the black-box API is an interface provided by the server that enables users to interact with the pre-trained GNN model without directly accessing its internal components, such as the model architecture, parameters, or graph structure. Users can submit prediction queries using node IDs. If a new node needs to be added to the graph, users can employ a *connect* query to attach the node to the graph before querying its prediction based on its ID. The API processes input data into output predictions, ensuring that the model's underlying computations remain hidden from the user. Users can query this GNN for the purpose of node classification. Hence, the query consists of the queried node's ID and the output of this query is the vector of prediction scores for this particular node. The users do not have the knowledge of the edges of this graph. Hence the only information that a user knows is the set of nodes' IDs. We consider an adversary, $\mathcal{A}$, who assumes the role of a GNN user. Her objective is to determine the neighbors of a specific *target node*, v_t, selected from a set of *target nodes*, $V_{\mathcal{A}}$, within the graph. This is done based on the GNN's predictions for the node set $V_{\mathcal{A}}$. In simpler terms, $\mathcal{A}$ aims to identify the neighbors of the target node v_t that are included in the target set nodes $V_{\mathcal{A}}$.

We should note that if the adversary aims to identify all the links within the graph, then the set of target nodes $V_{\mathcal{A}}$ becomes the set containing all the nodes of the graph V. To achieve this, the adversary may need to perform multiple node injections, targeting different nodes from the graph each time. However, the practicality of such an approach is debatable. The adversary's selection of target nodes reflects her background knowledge about these nodes. For instance, in the context of social networks, the adversary's background knowledge could include information such as users' interests. This information can guide the adversary in selecting target nodes $V_{\mathcal{A}}$ that are more likely to be connected. In our attack scenario, we choose the target nodes uniformly at random. The adversary $\mathcal{A}$ is able to obtain the predictions of the target nodes $V_{\mathcal{A}}$ by sending the server their corresponding IDs through the provided API. In addition, the adversary $\mathcal{A}$ is able to use the *connect* query to connect a node v_m to a target node v_t. In general, we assume that the adversary does not have access to the features of the nodes in the graph, with the exception of certain attack strategies described hereafter.

Adversary $\mathcal{A}$ can *connect* new nodes and further query the prediction scores of a set of nodes $V_{\mathcal{A}}$ in the graph. While adding this new node v_m, $\mathcal{A}$ can choose which existing node v_t it actually connects to and hence try to discover its neighbors. More formally, NILS is composed of the following steps:

1) $\mathcal{A}$ first queries the prediction scores of the target nodes $V_{\mathcal{A}}$ and receives the corresponding prediction matrix P of the target nodes $V_{\mathcal{A}}$.
2) $\mathcal{A}$ generates malicious features of a malicious node v_m based on the obtained prediction matrix P.
3) Next, $\mathcal{A}$ sends a *connect* query to inject the malicious node v_m. The query has the following parameters: the features x_m of the new node, and the ID of the target node v_t the adversary wishes to connect v_m to.
4) The server adds this malicious node v_m to the graph and links it to the target node v_t.
5) $\mathcal{A}$ queries back the server for new prediction matrix P' of the target nodes $V_{\mathcal{A}}$ and obtains it.
6) With access to P and P', $\mathcal{A}$ computes the L_1 distance between $P(v)$ and $P'(v)$ of each node v in $V_{\mathcal{A}}$. A significant change in the prediction scores of a node v indicates a high probability of being a neighbor with v_t. If the difference exceeds a threshold R, the adversary infers that node v is a neighbor of v_t.

The decision threshold R is determined through an extensive parameter tuning process, aiming for an optimal trade-off between precision and recall in identifying the true neighbors of the target node. This balance is represented by the F_1 score. We evaluate various candidate values of R, selecting the one that yields the highest F_1 score as the optimal threshold. The results reported in our study are based on the optimal value of R. NILS attack strategy is outlined in Algorithm 1.

Algorithm 1: Node Injection Link Stealing Attack

Input: set of nodes $V_{\mathcal{A}}$ and target node v_t.
Output: the identified neighbors of v_t by the adversary.
P = GNN($V_{\mathcal{A}}$, $X_{V_{\mathcal{A}}}$) ▷ Step 1
Generate malicious features x_m of node v_m ▷ Step 2
Connect node v_m to v_t. ▷ Step 3-4
P' = GNN($V_{\mathcal{A}} \cup v_m$, $X_{V_{\mathcal{A}}} \cup x_m$) ▷ Step 5
for *each node* v *in* $V_{\mathcal{A}}$ **do**
  $D(v) = \|P(v) - P'(v)\|_1$ ▷ Step 6
  if $D(v) \geq R$ **then**
    v is a neighbor of v_t
  end
  else
    v is not a neighbor of v_t
  end
end

To assess the impact of introducing a malicious node v_m on GNN predictions, we explore five strategies for generating its feature vector x_m:

1) **All-ones strategy**: Assigns $x_m = \mathbf{1}$, creating a dense feature vector that may significantly alter predictions but lacks stealth due to its density.
2) **All-zeros strategy**: Uses $x_m = \mathbf{0}$, leading to a sparse vector that subtly changes GNN outputs, enhancing stealth.
3) **Identity strategy**: Sets $x_m = x_t$, copying the target node's features, which disrupts the model's predictions for neighboring nodes; stealth varies with the similarity between the target and malicious nodes.
4) **Max attributes strategy**: Forms x_m by taking the maximum attributes from nodes of different classes than the target node, resulting in pronounced feature exaggerations and potential detection risks.
5) **Class representative strategy**: Chooses x_m from a node with the highest confidence in a class not shared with the target, subtly altering neighboring predictions and potentially increasing stealth.

Additionally, we introduce the **LinkTeller Influence** strategy, modifying the target node's features by a small value α

($x_m = x_t + \alpha$), blending their perturbation approach with our model.

B. Defense

LapGraph is a DP defense strategy typically adopted for link stealing attacks [2]. In the specific context of our work, we adapt LapGraph to effectively protect against our node injection link stealing attack under the one-node-one-edge-level DP framework. This unique application is crucial because the adversary in our model adds a malicious node to the graph, connected through a single edge, aiming to exploit the adjacency matrix's sensitivity to reveal confidential links. By perturbing the adjacency matrix using the Laplace mechanism and binarizing it, where the top-N largest values are set to 1 and the rest to 0, LapGraph obscures the changes induced by such node injections.

The post-processing property of DP, which LapGraph leverages, ensures that the privacy of the edge information is maintained even when an adversary observes the GNN's output predictions. Furthermore, each addition of a new node triggers a regeneration of the adjacency matrix according to the LapGraph method, thereby reinforcing the privacy protections incrementally through the sequential composition of DP. This adaptation is particularly pertinent for our one-node-one-edge-level DP as it addresses the nuanced threat posed by node injections—a scenario where traditional edge-level DP mechanisms might fall short due to the increased sensitivity and potential for information leakage

III. Evaluation of the attack and defense

The success rates of these strategies, as shown in Table I, reveal that the All-ones, Max attributes, and Class representative strategies are the most effective in causing significant changes in the predictions of the target node's neighbors. These results suggest that injecting nodes with high-valued or class-specific features can effectively disrupt the model's output predictions. Conversely, the All-zeros, and Identity strategies exhibit relatively lower success rates, as shown in Table I. While these strategies offer certain benefits in terms of stealthiness, their impact on the graph structure and predictions is less pronounced, highlighting a trade-off between attack effectiveness and stealthiness. These findings underscore the importance of considering both the effectiveness and stealthiness of malicious feature generation strategies when devising link inference attacks on GNNs.

Method	Twitch-FR	Twitch-RU	Flickr
Class Rep.	0.94 ± 0.01	0.83 ± 0.06	0.96 ± 0.06
Max Attr.	0.99 ± 0.00	0.98 ± 0.02	$\mathbf{1.00 \pm 0.00}$
All-ones	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.97 \pm 0.01}$	0.99 ± 0.02
All-zeros	0.58 ± 0.02	0.48 ± 0.01	0.71 ± 0.07
Identity	0.81 ± 0.02	0.69 ± 0.01	0.95 ± 0.07
Influence NILS	0.81 ± 0.02	0.70 ± 0.01	0.89 ± 0.10
Influence LinkTeller [2]	0.80 ± 0.02	0.74 ± 0.01	0.32 ± 0.13

Table I: F_1 scores and standard deviations for different attack methods and datasets.

Hereafter, we present the results of experiments conducted to evaluate the performance of NILS attack in comparison to the LinkTeller attack, using an identical experimental setup. Our focus is on analyzing the optimal attacks for both approaches, which involved accurately estimating the number of neighbors of the target set nodes. The results, summarized in Table II, demonstrate that our attack outperforms LinkTeller on both Twitch datasets (TWITCH-FR and TWITCH-RU). Furthermore, our method exhibits a substantial improvement over LinkTeller on the Flickr dataset, achieving nearly double the precision and recall values. Notably, our attack demonstrates stable performance across varying node degrees, with only a marginal decrease in effectiveness for high-degree target nodes. This can be attributed to the smaller influence that each neighboring node has on the aggregation of the GCN layer when the target node degree is high. Overall, our proposed NILS attack demonstrates consistently a superior performance as opposed to the LinkTeller attack. We further compare our attack with link-stealing attacks introduced in [3], where the authors' various attack strategies rely on different types of background knowledge available to the adversary, such as node attributes and shadow datasets. Specifically, in their Attack-2, the adversary has access to both the features and prediction scores of the nodes. Utilizing this information, the adversary creates two types of attacks: LSA2-attr and LSA2-post. LSA2-attr calculates distances between node attributes, while LSA2-post computes distances between node prediction scores (posteriors). It is important to highlight that these two attacks align closely with our threat model, as both assume that the adversary has access to the features and prediction scores of the target node. This similarity in assumptions renders these attacks particularly relevant for comparison with our proposed NILS attack. The attacks are executed under the transductive setting, where training and inference occur on the same graph. As shown in Table III, our proposed NILS attack outperforms the LSA2-post and LSA2-attr attacks constructed in [3]. However, our attack performance is nearly equivalent to that of LinkTeller. These results demonstrate that NILS attack maintains effectiveness under the transductive setting, just as in the inductive setting.

Figure 1 presents the F_1 score of the attack for various ε values. We observe that applying LapGraph reduces the effectiveness of NILS. The F_1 score becomes almost zero when the privacy budget ε is small. However, for large ε, LapGraph provides moderate protection, but the attack's F_1 score remains significantly lower than in the non-private case where DP is not applied. For comparison, in the LinkTeller [2] attack, where LapGraph is applied only once to ensure edge-level DP, LapGraph offers limited protection when ε is large, allowing LinkTeller to achieve a success rate nearly as high as in the non-private case. Conversely, in our scenario, where LapGraph is also applied after the adversary's node injection, LapGraph provides stronger protection.The application of LapGraph during inference makes it more challenging for the adversary to distinguish between the target node's neighbors and non-neighbors, as the prediction scores of all target nodes change after each inference query. Consequently, the distances between the prediction scores P and P', before and after the node injection, become noisier due to LapGraph's application following the node injection.

To provide insights about the privacy-utility tradeoff of LapGraph, we present in Figure 2 the utility of the GCNs for different values of the privacy budget. We observe that the utility increases when ε increases, as expected. Large values of $\varepsilon \geq 7$ give a better utility close to that in the non-private vanilla case. Therefore, carefully choosing an ε will give fairly good utility and a certain level of protection against NILS attack.

IV. Conclusion

In this paper, we have presented a powerful new NILS at- tack—a link-stealing attack using node injection against GNNs. Our results have demonstrated the superior perfor-

Dataset	Method	low		uncontrained		high	
		precision	recall	precision	recall	precision	recall
TWITCH-FR	NILS (Ours)	100.0 ±0.0	100.0 ±0.0	99.13 ±0.8	99.57 ±0.35	99.91 ±2.6	100.0 ±0.0
	LinkTeller	92.5 ±5.4	92.5 ±5.4	84.1 ±3.7	78.2 ±1.9	83.2 ±1.4	80.6 ±6.7
TWITCH-RU	NILS (Ours)	100.0 ±0.0	100.0 ±0.0	96.45 ±0.4	98.34 ±0.7	99.77 ±0.1	99.37 ±0.1
	LinkTeller	78.8 ±1.9	92.6 ±5.5	71.8 ±2.2	78.5 ±2.4	89.7 ±1.7	65.7 ±3.9
Flickr	NILS (Ours)	100.0 ±0.0	100.0 ±0.0	99.11 ±1.7	95.83 ±5.0	93.72 ±3.1	78.9 ±1.9
	LinkTeller	51.0 ±7.0	53.3 ±4.7	33.8 ±13.3	32.1 ±13.3	18.2 ±4.5	18.5 ±6.1

Table II: Comparative performance of our proposed attack NILS and LinkTeller across three datasets (TWITCH-FR, TWITCH-RU, and Flickr) under low, unconstrained, and high constraint settings. The results are presented in terms of precision and recall with corresponding standard deviations

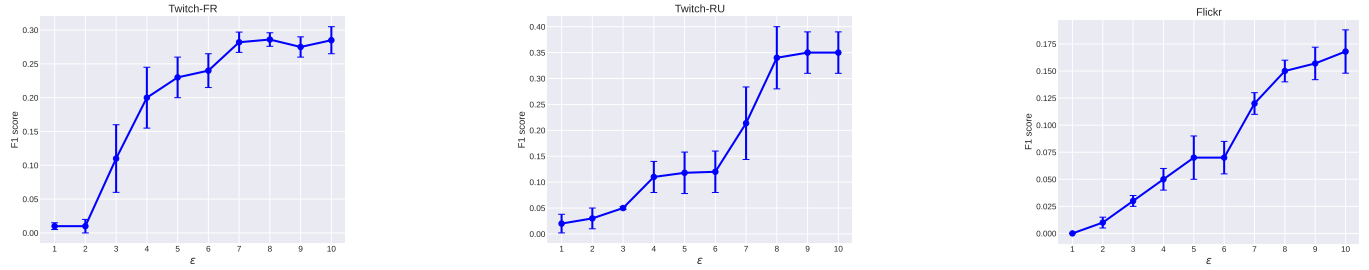

Figure 1: F_1 score of the attack for different values of ε.

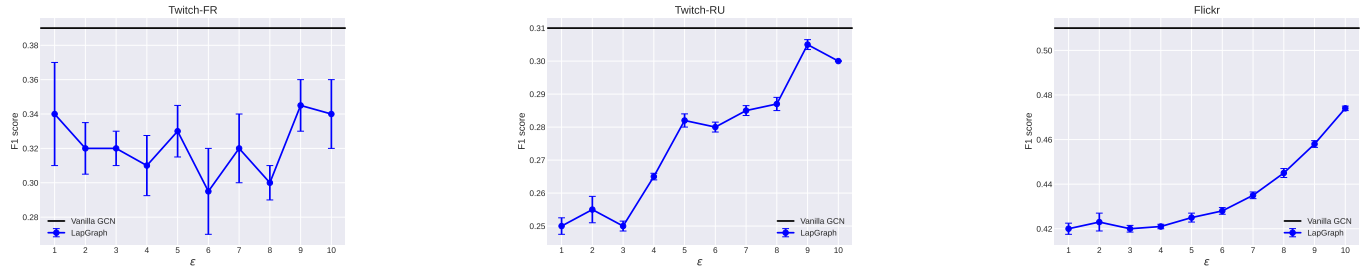

Figure 2: F_1 score utility of the GCN for different values of ε.

Method	Cora		Citeseer		Pubmed	
	precision	recall	precision	recall	precision	recall
NILS (Ours)	99.7 ±0.2	99.6 ±0.3	97.4 ±0.2	98.2 ±0.1	99.7 ±0.0	100.0 ±0.0
LinkTeller	99.5 ±0.1	99.5 ±0.1	99.7 ±0.0	99.7 ±0.0	99.7 ±0.0	99.7 ±0.0
LSA2-post	86.7 ±0.2	86.7 ±0.2	90.1 ±0.2	90.1 ±0.2	78.8 ±0.1	78.8 ±0.1
LSA2-attr	73.6 ±0.1	73.6 ±0.1	80.9 ±0.1	80.9 ±0.1	82.4 ±0.1	82.4 ±0.1

Table III: Comparative performance of NILS attack with LinkTeller [2] and link-stealing attacks in [3] across three datasets (Cora, Citeseer, and Pubmed).

Dataset	Method	Depth-2		Depth-3	
		precision	recall	precision	recall
TWITCH-FR	NILS (Ours)	99.13 ±0.8	99.57 ±0.35	85.06 ±1.2	81.56 ±1.2
	LinkTeller	84.1 ±3.7	78.2 ±1.9	50.1 ±5.1	46.6 ±5.0
TWITCH-RU	NILS (Ours)	96.45 ±0.4	98.34 ±0.7	78.78 ±3.8	76.35 ±9.3
	LinkTeller	71.8 ±2.2	78.5 ±2.4	45.7 ±2.2	50.0 ±2.8

Table IV: Success rates of the attack for different depths in comparison with LinkTeller [2]. We use the all-ones strategy and Twitch-FR dataset.

mance of NILS compared to previous attacks, further emphasizing the vulnerabilities of GNNs regarding edge information leakage. We have also evaluated NILS against differentially private GNNs, ensuring a one-node-one-edge-level DP notion specifically designed to protect against our proposed attack.

REFERENCES

[1] C. Cadwalladr and E. Graham-Harrison, "Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach," *The Guardian*, Mar 2018. [Online]. Available: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election

[2] F. Wu, Y. Long, C. Zhang, and B. Li, "Linkteller: Recovering private edges from graph neural networks via influence analysis," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2005–2024.

[3] X. He, J. Jia, M. Backes, N. Z. Gong, and Y. Zhang, "Stealing links from graph neural networks." in *USENIX Security Symposium*, 2021, pp. 2669–2686.

[4] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan 2009.

[5] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[6] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[7] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[8] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[9] Z. Wang, Q. Cao, H. Shen, B. Xu, M. Zhang, and X. Cheng, "Towards efficient and expressive gnns for graph classification via subgraph-aware weisfeiler-lehman," in *Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event*, ser. Proceedings of Machine Learning Research, B. Rieck and R. Pascanu, Eds., vol. 198. PMLR, 2022, p. 17. [Online]. Available: https://proceedings.mlr.press/v198/wang22b.html

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl