SpotTarget: Rethinking the Effect of Target Edges for Link Prediction in Graph Neural Networks

Jing Zhu*
University of Michigan, Ann Arbor
jingzhuu@umich.edu

Yuhang Zhou*
University of Maryland, College Park
tonyzhou@umd.edu

Vassilis N. Ioannidis
AWS AI Research and Education
ivasilei@amazon.com

Shengyi Qian
University of Michigan, Ann Arbor
syqian@umich.edu

Wei Ai
University of Maryland, College Park
aiwei@umd.edu

Xiang Song
AWS AI Research and Education
xiangsx@amazon.com

Danai Koutra
University of Michigan, Ann Arbor
dkoutra@umich.edu

ABSTRACT

Graph Neural Networks (GNNs) have demonstrated promising outcomes across various tasks, including node classification and link prediction. Despite their remarkable success in various high-impact applications, we have identified three common pitfalls in message passing for link prediction, especially within industrial settings. Particularly, in prevalent GNN frameworks (e.g., DGL and PyTorch-Geometric), the target edges (i.e., the edges being predicted) consistently exist as message passing edges in the graph during training. Consequently, this results in *overfitting* and *distribution shift*, both of which adversely impact the generalizability to test the target edges. Additionally, during test time, the failure to exclude the test target edges leads to implicit *test leakage* caused by neighborhood aggregation. In this paper, we analyze these three pitfalls and investigate the impact of including or excluding target edges on the performance of nodes with varying degrees during training and test phases. Our theoretical and empirical analysis demonstrates that low-degree nodes are more susceptible to these pitfalls. These pitfalls can have detrimental consequences when GNNs are implemented in production systems. To systematically address these pitfalls, we propose SpotTarget, an effective and efficient GNN training framework. During training, SpotTarget leverages our insight regarding low-degree nodes and excludes train target edges connected to at least one low-degree node. During test time, it emulates real-world scenarios of GNN usage in production and excludes *all* test target edges. Our experiments conducted on diverse real-world datasets, demonstrate that SpotTarget significantly enhances GNNs, achieving up to a 15× increase in accuracy in sparse graphs. Furthermore, SpotTarget consistently and dramatically improves the performance for low-degree nodes in dense graphs.

ACM Reference Format:
Jing Zhu*, Yuhang Zhou*, Vassilis N. Ioannidis, Shengyi Qian, Wei Ai, Xiang Song, and Danai Koutra. 2023. SpotTarget: Rethinking the Effect of Target Edges for Link Prediction in Graph Neural Networks. In *Proceedings of KDD MLG'23.* ACM, New York, NY, USA, 11 pages. https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

KDD MLG'23, August, 2023, Long Beach CA, USA

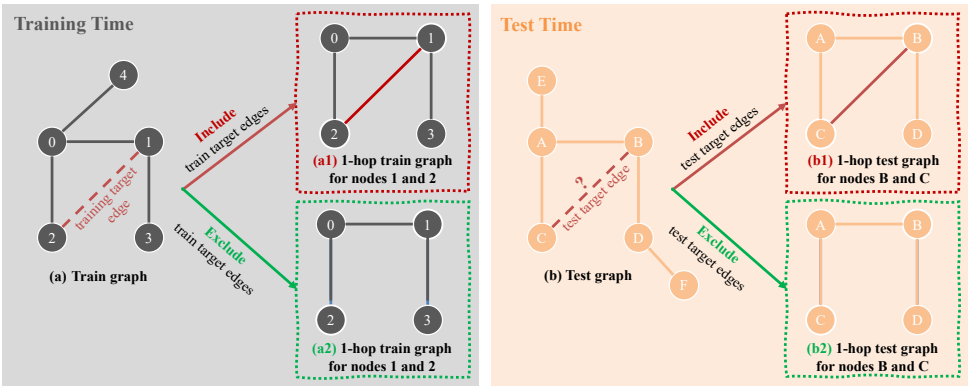
ACM ISBN XXXXXXXXXXXXXXXXXX…$15.00
https://doi.org/XX.XXXX/XXXXXXX.XXXXXXX

1 INTRODUCTION

Graphs or networks serve as fundamental representations for relational data, and their analysis is useful in numerous scientific and industrial applications. Link prediction, the task of predicting whether a link is likely to form between two nodes or entities in a graph, has wide-ranging downstream applications, including drug repurposing, recommendation systems, and knowledge graph completion [1, 3, 15, 16, 18, 31]. Link prediction is widely employed as a pre-training technique to generate high-quality entity representations applicable to diverse business applications. Techniques for solving this task vary, ranging from heuristics (e.g., predicting links based on the number of common neighbors between node pairs) to graph neural network (GNN) models. GNN models utilize message passing and leverage both the graph structure and node features. Recently, GNN-based approaches, which frame the link prediction problem as a binary classification task over node pairs, have achieved state-of-the-art performance in numerous high-impact applications and have emerged as the go-to approach in both industry and academia [12, 14, 32, 34].

However, in most existing GNN models, the *target edges* (i.e., the edges whose existence or absence is predicted) exist consistently in the training graph and serve as message passing edges during training. This results in two pitfalls during training: (P1) overfitting and (P2) distribution shift. We observe that these pitfalls are prevalent in the majority of GNN-based frameworks used for link prediction. For instance, PyTorch Geometric (PyG)[9], a widely used library, lacks support for excluding target edges during the construction of mini-batch graphs for training. Although DGL[26] offers support for excluding target edges through one of its sampler arguments, the significance of excluding these target links is frequently ignored by the community. These pitfalls are prevalent in both industry and academia, where researchers frequently develop application code using open-source frameworks such as PyG and DGL. Consequently, it is easy to overlook the problems associated with overfitting and distribution shift. In addition to training time, failure to exclude the test target edges from the test graph leads to (P3) implicit test leakage through neighborhood aggregation.

* equal contribution



Training Target Edges	Test Target Edges	Results	Pitfalls
Include (a1)	-	✗	(P1) Overfitting
Include (a1)	Exclude (b2)	✗	(P2) Distribution shift
-	Include (b1)	✗	(P3) Leakage
Exclude (a2)	Exclude (b2)	✓	–

Figure 1: Common pitfalls in message passing for link prediction when the training and/or test target links are included as message passing edges. [Left] Training time: Given a toy train graph and training target edge e_{12} (a), illustration of the impact of the inclusion (a1) and exclusion (a2) of e_{12} in the 1-hop induced train graph for nodes 1 and 2, which is used for message passing. [Right] Test time: The same illustration for a given test graph and test target edge e_{BC} (b). [Table]: Overview of the three main pitfalls and when they arise: (P1) When train target links are included, GNNs *overfit* on them instead of making predictions based on the graph structure and features. (P2) When train target links are present but test target edges are absent, there is a *distribution shift* between training and testing. (P3) The presence of test target links results in *implicit test leakage*. These pitfalls can be resolved by excluding the target links from both the training and test message passing graphs.

While some popular benchmarks [11] have deliberately addressed this issue during their construction, it remains a frequently overlooked problem for applications that heavily rely on proprietary data. Specifically, deployed systems are more prone to experiencing these pitfalls. **In this paper, we emphasize the significance of excluding target edges during both training and testing time, quantify the adverse impact of their inclusion on the performance of deployed solutions, and introduce an effective and efficient GNN framework that mitigates all these pitfalls.**

Example 1.1. We give an illustrative example of the pitfalls in message passing during training and test time in Fig. 1. During training, the existence of training target edges during message passing leads to (P1) overfitting and (P2) distribution shift. In Fig. 1(a), e_{12} is a training target edge for which we want to predict the existence. When this edge is not excluded from the training graph, GNNs would use the message passing graph shown in Fig. 1(a1) for nodes v_B and v_C, which leads to overfitting on e_{12} and memorizing its existence instead of learning to predict it based on the graph structure and node features. Moreover, in a realistic testing scenario as in Fig. 1(b) where the goal is to predict whether the edge e_{BC} exists or not, GNNs would use the message passing graph shown in Fig. 1(b2) for nodes v_B and v_C, which are connected by a missing edge e_{BC}. This leads to distribution shift: There is discrepancy between the message passing graphs used during training and testing despite the similarity between the train and test graphs as well as the target links. Pitfalls (P1) and (P2) lead to poor performance for GNN models and inability to effectively generalize to (truly) unobserved links at test time or in production.

At test time, since most popular GNN frameworks do not explicitly emphasize the importance of excluding the test target edges from the test graph, it is easy to overlook the fact that the test target edges included by default in message passing causes data leakage. In our example, if the test target edge e_{BC} is used in the message passing graph for nodes v_B and v_C (Fig. 1(b1)), when performing neighborhood aggregation, the target node v_B would aggregate the messages from v_C and vice versa, resulting in a higher likelihood of predicting v_B and v_C forming an edge compared to the case where e_{BC} does not exist on the message passing graph. But in real-world applications in deployment, the goal is to predict the *future* links that are *not observed* in the data, so the inclusion of test target links corresponds to implicit data leakage.

To address three identified pitfalls, we present SpotTarget, a framework that systematically excludes the target edges during both training and testing. SpotTarget builds upon the crucial observation that low-degree nodes are more susceptible to these pitfalls due to the substantial relative degree changes resulting from the inclusion of target edges, in contrast to high-degree nodes. This is because, for high-degree nodes, the erroneously-considered additional message passing edges (target links) have minimal impact on performance, as they constitute a small fraction of the edges involved in message passing and neighborhood aggregation. Therefore, addressing these pitfalls is crucial in industrial settings, especially for e-commerce applications, where the observed data is incomplete and highly sparse, containing numerous low-degree nodes [20]. While excluding all training targets may seem like an ideal solution, we explored that it significantly disrupts the mini-batch graph and harms the GNN learning. Theoretical and empirical

evidence demonstrate that excluding the target links connected to at least one low-degree node achieves the optimal balance between avoiding pitfalls (P1)–(P2) and acquiring strong node representations from resilient mini-batch graphs during training. During test time, we assert the significance of emulating real-world scenarios and preventing leakage by excluding *all* target edges from the test graph. Our framework SpotTarget consists of two modules: (1) an edge sampler that excludes target edges connected to at least one low-degree node, and (2) a leakage check module that verifies the exclusion of target test edges for any user-defined dataset and automatically excludes them if necessary. We are currently in the process of integrating our plug-and-play framework into the DGL library [25]. Our contributions can be summarized as follows:

- **Systematic Analysis of Target Edges' Effects**: Focusing on link prediction, we conduct the first theoretical and empirical analysis to examine the impact of including target edges as message passing edges during training and test time. Our key insight reveals that low-degree nodes are more susceptible to the pitfalls of including the target edges as message passing edges.
- **Unified Framework**: We present SpotTarget, the first unified GNN training framework that addresses the pitfalls by automatically excluding target edges that are incident to at least one low-degree node during training and excludes all target edges during testing to ensure robust model training without any implicit data leakage.
- **Rigorous Experiments**: To quantify the impact of target edges during training and test time, we performed extensive experiments on a diverse range of datasets, including widely-used link prediction benchmarks as well as real-world datasets. Additionally, we conducted a specific analysis of the impact on the low-degree nodes in dense datasets. Our findings demonstrate that SpotTarget achieves up to a 15 $\times$ increase in the accuracy of GNN models on sparse graphs. Moreover, it notably enhances the performance of low-degree nodes on dense graphs.

2 RELATED WORK

Link Prediction using GNNs. Graph neural networks (GNNs) are popular neural network architectures that learn representations by capturing the interactions between objects. While perhaps most often used for node- or graph-level classication, the applications of graph neural networks have expanded to include edge-level inference tasks like link prediction. Methods that use GNNs for link prediction mainly fall into two categories: Graph Autoencoder (GAE)-based methods and enclosing subgraph-based methods. GAE-based methods use GNNs as the encoder of nodes, and edges are decoded by their nodes' encoding vectors using score functions [5, 14, 24, 30, 36]. For enclosing subgraph-based methods including SEAL [32, 34], IGMC [33], GraIL [23], TCL-GNN [29] first extract an enclosing subgraph for the target edge , apply GNNs to encode the node representations of nodes in enclosing subgraph, and then aggregate the node representations by pooling methods. Subsequently, the learned subgraph features are utilized as inputs to a classifier to predict the presence of the target edge. Despite the fact that enclosing subgraph-based methods, such as SEAL, yield more accurate predictions, GAE-based methods are typically significantly faster to compute and require fewer computational resources. In industrial applications, graphs often exhibit a massive scale, encompassing millions or even billions of nodes. Consequently, GAE-based methods are commonly preferred [35].

Table 1: Major symbols and their definitions.

Symbols	Definitions
G	Graph
d_i	Degree of node i
e_{ij}	The target edge between nodes i, j to be predicted
T_{tr}	The set of train target edges
T_{tst}	The set of test target edges
T_{low}	Set of target edges incident to at least one low-degree node
δ	Degree threshold to filter edges in T_{low}

Issues in Link Prediction using GNNs. In contrast to node classification, where edges serve solely as message passing edges, the edges in the link prediction task play dual roles: (1) message passing edges and (2) prediction objectives. Consequently, it is crucial to pay special attention to dataset construction and loading to mitigate potential pitfalls that can negatively impact the performance of deployed systems. However, this distinction is frequently overlooked, particularly in GAE-based methods, which often directly adapt GNN models designed for node classification to generate node embeddings. This oversight leads to problems in training GNN models for link prediction. The training pitfalls caused by the existence of target edges were initially identified by SEAL [27, 32], which made efforts to mitigate them through negative injection. Expanding on this work, FakeEdge [7] delves into the distribution shift issues that arise from the presence of target edges during training and their absence during testing. They further propose frameworks that involve the addition, removal, or a combination of both of the target edges from the induced subgraph in subgraph-based methods like SEAL. In contrast to SEAL or FakeEdge, our work aims to conduct a comprehensive analysis of all the pitfalls arising from the presence of target edges. Through an extensive degree-based analysis, we demonstrate which types of nodes are more susceptible to these pitfalls and present strategies to strike an optimal balance between avoiding training pitfalls and maintaining graph structure based on node degrees. In comparison to SEAL and FakeEdge, our proposed framework specifically addresses this concern within GAE-based methods. This holds greater practical significance as GAE-based methods are frequently employed in industry due to their high efficiency and their ability to leverage nodes' natural features for link prediction. Additionally, our proposed framework exhibits greater scalability compared to FakeEdge. For instance, while FakeEdge requires over 20 hours to complete a single epoch on Ogbl-Citation2, SpotTarget achieves the same task in just 2 hours.

3 PRELIMINARIES

In this section, we formally define key notions as well as the problem that we seek to solve. The major symbols we use is defined in Tab. 1.

3.1 Definitions

3.1.1 Graphs. We consider a **graph** $G = (V, E, \mathbf{X})$, where V is the set of vertices, E is the set of edges, and $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ represents the d-dimensional input node features. We denote the k-**hop neighbors** of node u—i.e., the set of nodes at a distance less than or equal

to k from u—as $N_k(u)$. The **degree** d_u of node u is defined as the number of its 1-hop neighbors or adjacent nodes, i.e., $d_u = |N_1(u)|$.

3.1.2 Link Prediction. Given a graph $G = (V, E, \mathbf{X})$, the link prediction task aims to determine whether there is or will be a link e_{ij} between a pair of nodes i and j, where $i, j \in V$ and $e_{ij} \notin E$. We refer to e_{ij}, the edge for which we want to predict the existence or absence, as **target edge or link**.

In this work, we distinguish different types of target links: (1) **training vs. test target links**: the training target edges, T_{tr}, are used to train a supervised link prediction model, while the test target links, T_{tst}, are the links for which we want to predict the existence or absence at test time (e.g., when evaluating the test performance or making predictions in a deployed system). In our real experiments, (2) **target links that are incident to at least one low-degree node vs. not**: based on our theoretical insights in Sec. 5.1, our framework leverages target links that are incident to at least one low-degree node (i.e., edges e_{uv} for which $\min(d_u, d_v)$ is small), denoted as T_{low}. Throughout the paper, our discussion of the inclusion of test target links is done in the setting of evaluating test performance instead of deployed system, where there exist ground truth test links as labels. We primarily adopt the widely used train, validate, test setting, where only the epoch that achieves best performance on validation edges will be evaluated by test edges.

3.1.3 Graph Neural Networks. GNN models utilize a neighborhood aggregation scheme to learn a representation vector h_v for each node v. In general, the node representation of node v can be formulated as a k-round neighborhood aggregation schema: $h_v^{(k)} = \text{COMBINE}^{(k)}(\{h_v^{(k-1)}, \text{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in N_k(v)\})\})$, where AGGREGATE(.) is typically mean or max pooling, and COMBINE(.) can be a sum/concatenation/attention on nodes' ego- and neighbor-embeddings. GNN models utilize a neighborhood aggregation scheme to learn a representation vector h_v for each node v. For link prediction tasks, GNNs are typically trained using cross-entropy loss to differentiate positive links versus negative links. Given a set of target links, we define the k-**hop message passing graph** of a GNN model as the induced subgraph that contains all the endpoint nodes of the target edges, their k-hop neighbors, and the edges of the original graph that connect these nodes. Examples of (train and test) 1-hop message passing graphs are given in Fig. 1.

3.2 Problem Statement

Given a graph G, a link prediction task, and a base GNN model in a mini-batch training setting, we seek to: (1) explain the pitfalls of including the target links T_{tr} and T_{tst} as message passing edges during training and test time, respectively, and (2) propose solutions to best avoid the pitfalls.

4 PITFALLS IN MESSAGE PASSING FOR LINK PREDICTION

In this section, we aim to explain the pitfalls of including target edges as message passing edges during training and test time.

4.1 Pitfalls during Training Time

When the train target edges are presented and used as message passing edges, it causes overfitting and distribution shifts.

(P1) Overfitting. Suppose that we have an original train graph G, as shown in Fig. 1(a). GAE based methods first generate node 1 and node 2's embeddings by aggregating their 1-hop neighbor's information and decode the likelihood of node 1 and node 2 forming an edge using a dot product decoder. When the target edge e_{12} is present, node 1's embedding aggregates node 2's features, and vice versa. And since the training objective is to predict y_{12} as high as possible. The GNNs would learn that as long as it tries to overfit on the train objective, it would achieve its train target. Similarly, subgraph-based models first find an enclosing subgraph for target edges T_{tr} and then apply GNNs upon the enclosing subgraph to predict the link existence also suffer from overfitting issues. And this overfitting issue causes GNNs to generalize poorly to test data.

(P2) Distribution Shift. In typical GNN trainings for link prediction, the train target edges T_{tr} are usually present and used during message passing, while the test target edges T_{tst} are absent and never used during test, and this poses a distribution shift problem. As shown in Fig. 1, suppose that (a) is the train graph with e_{12} as the train target edge and (b) is the test graph with e_{BC} as the test target edge. At training time, when node 1 aggregates the message from its neighbors, node 1 sees that node 2 is its direct neighbors, and the messages from node 2 are aggregated and used to compute node 1's embeddings Fig. 1(a1). In the realistic test scenario, when node B aggregates the message from its neighbors, it never treats node C as its direct neighbor because future links are not observed in the data. And this poses a distribution shift between training and testing time and results in poor generalizability.

4.2 Pitfalls during Test Time

(P3) Data Leakage. During the test of the GNN models, popular GNN frameworks do not provide explicit functions to remove the testing edges from message passing. One may overlook the fact that the test target edges are included by default in message passing. We found that when test targets edges T_{tst} are included in message passing, it results in implicit data leakages, as shown in Fig. 1(b1). When test target e_{12} exists in the test message passing graph, in order to do the inference, the target node v_1 would aggregate messages from v_2 and vice versa, resulting in a higher likelihood of predicting v_1 and v_2 forming an edge, compared with the case when e_{12} does not exist in the message passing graph. And this is actually a test leakage because it would result in near-perfect performance for expressive GNN frameworks even though GNNs never learn on the test target edges. In real-world deployment applications, future links that we want to predict upon are never observed in the data.

5 PROPOSED FRAMEWORK: SpotTarget

In this section, we detail SpotTarget, the first framework systematically resolving the issues of the presence of target edges in the message passing graph for link prediction. We propose separate solutions for training and inference time, respectively.

5.1 Training-time Solution: Exclude Target Links Adjacent to Low-degree Nodes

As discussed in Sec. 4.1, when train target edges T_{Tr} are used as message passing edges, it causes overfitting and distribution shifts.

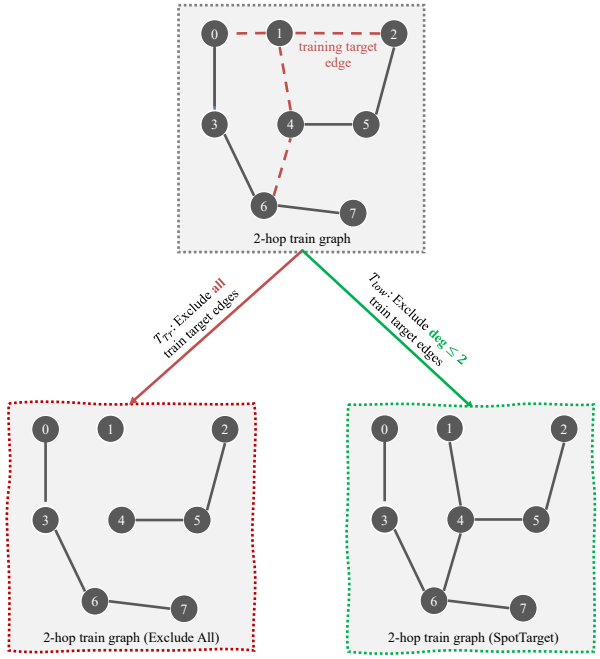


Figure 2: Example 2-hop message passing graph for a mini-batch of size 4. Red lines are train target edges and black lines are other message passing edges induced by the target edges. [Left] if all target (red) edges T_{Tr} are excluded during training, it results in three disconnected components. [Right] If only low-degree train targets edges are excluded T_{low}, the graph is still connected. There is limited corruption to the graph structure while avoiding pitfalls P1 and P2.

One intuitive solution is to exclude all train target edges during training. However, this exposes several challenges.

- First, when excluding all target edges in the batch, one excluded target edge could be the message passing edge of another target edge. For example, in Fig. 2, e_{14} is both a target edge and a message passing edge of node 4 and node 1. The existence of e_{14} can also influence the learning of target edges e_{46}, e_{01} and e_{12}. Excluding all target edges causes significant corruption of the graph structure. The extreme case is that some nodes become isolated nodes after excluding all the target edges like node 2 in Fig. 2. As a result, GNNs fail to learn good representations when all target edges are excluded.
- Second, the corruption of graph structure resulting from the exclusion of all target edges is applicable to both mini-batch and full-batch settings. Moreover, in full-batch training, each training step requires iterating over all edges in the graph to remove the target edges, resulting in time-consuming operations. In industrial settings, performing full-batch training for link prediction on large graphs (consisting of millions of nodes) is uncommon due to its inefficiency in terms of both time and space complexity.
- Third, although setting the batch size to 1 can resolve the issue of structure corruption, it leads to a significantly small mini-batch graph, resulting in inefficiency and instability during training.

The question at hand is: How can we attain the optimal trade-off between mitigating the pitfalls (P1, P2) resulting from the presence of train target edges and preserving the graph structure as much as possible during mini-batch training?

The key to addressing this problem lies in analyzing which nodes are most affected by P1 and P2 and selectively excluding only the target edges incident to those highly affected nodes. We argue that by exclusively excluding the target edges incident to low-degree nodes, we can achieve the optimal trade-off between avoiding training pitfalls and minimizing graph corruption. Since low-degree nodes have fewer neighbors compared to high-degree nodes, there is a small probability that the excluded target edges adjacent to low-degree nodes serve as message-passing edges for other nodes. Next, we present a theoretical analysis demonstrating that low-degree nodes are the most affected by the pitfalls. Based on this analysis, we propose that target links in T_{low} incident to at least one low-degree node should be excluded during mini-batch training.

Theoretical Analysis. In this section, we explain why low-degree nodes suffer more from the pitfalls caused by the inclusion of train targets than high-degree nodes from a theoretical perspective. Intuitively, we focus on comparing the distance a random node v_k's effect on node v_h, v_l before and after excluding one neighboring edge of v_h, v_l respectively. We leverage the notion of influence/effect functions in statistics [22, 28] to measure the relative influence of a node on another node through a specific train edge.

Theorem 1. *Let v_h and v_l be two nodes in a graph with $d_h > d_l$, let node v_k be an arbitrary node in the graph. Assume that ReLU is the activation function, the Λ-layer GNN is untrained, and all random walk paths from v_h to v_l have a return probability of 0. We denote the effect of node v_k on node v_h after Λ-th layer GNN as $\frac{\partial x_h^\Lambda}{\partial x_k}$ where x_h, x_k are $n-$dimensional vectors indicating the embeddings for node v_h, v_k. Further we denote that effect of node v_k on node v_h after removing one neighboring edge of node v_h as $\frac{\partial \tilde{x}_h^\Lambda}{\partial x_k}$. Define the change of effect of v_k on v_h before and after removing one neigboring edge of v_h as a distance function $D(k,h)$, where $D(k,h) = 1 - \mathbb{E}(\frac{\partial \tilde{x}_{h,s}^\Lambda}{\partial x_{k,t}} / \frac{\partial x_{h,s}^\Lambda}{\partial x_{k,t}})$ for any entry $1 \leq s,t \leq n$ of x_h and x_k. And similarly, for node v_l, we also have $D(k,l) = 1 - \mathbb{E}(\frac{\partial \tilde{x}_{l,s}^\Lambda}{\partial x_{k,t}} / \frac{\partial x_{l,s}^\Lambda}{\partial x_{k,t}})$ for any entry $1 \leq s,t \leq n$ of x_l and x_k. Then $D(k,h) < D(k,l)$.*

We provide the proof in Appendix B. From Theorem 1, we prove that the change of effect of a random node v_k on v_h, v_l brought by excluding target edges is higher for v_l than v_h. This shows that low-degree nodes benefit more by excluding target edges. As a result, when all target edges are present, low-degree nodes are more vulnerable to the pitfalls brought by the inclusion of target edges. And this holds for any message passing GNNs.

Besides the theoretical analysis, we also provide a quantitative analysis of average degree change in Appendix A.3 and conclude that when excluding training targets, the relative degree change for nodes with low degrees is much larger than high-degree nodes.

Solution: Exclude edges incident to at least one low-degree node lower than δ. To achieve the best trade-off between these, we propose to exclude the train target edges where at least one incident node is lower than a degree threshold δ. We denote the set of these target edges as T_{low}. We claim that excluding train targets in T_{low} achieves the best trade-off between avoiding pitfalls brought

Algorithm 1 SpotTarget: Leakage Check(G)

Input: An input graph $\mathbf{G}$, edge splits $\mathbf{S}$, an argument $\mathbf{K}$ if valid edges are used as inference inputs, $\mathbf{K} = \{T, F\}$
Output: The desired inference graph $\mathbf{G}_{\text{infer}}$
/* STEP 1. Check if the input graph contains validation and test edges */
C_{valid} = Check Existence($\mathbf{G}, \mathbf{S}_{\text{valid}}$)
C_{test} = Check Existence($\mathbf{G}, \mathbf{S}_{\text{test}}$)
/* STEP 2. Delete Test and Validation Edges According to User Requirement*/
if C_{test} is True **then**
    $\mathbf{G}_{\text{infer}}$ = RemoveEdge($\mathbf{G}, \mathbf{S}_{\text{test}}$)
else
    $\mathbf{G}_{\text{infer}} = \mathbf{G}$
/*If Validation Edges Exist in the Inference Graph and is not Desired*/
if C_{valid} is True and $\mathbf{K}$ is False **then**
    $\mathbf{G}_{\text{infer}}$ = RemoveEdge($\mathbf{G}_{\text{infer}}, \mathbf{S}_{\text{valid}}$)
return $\mathbf{G}_{\text{infer}}$

Table 2: Statistics of the datasets used for link prediction experiments. All numbers are counted in the training dataset without valid and test edges.

Dataset	# Nodes	# Edges	Node deg.	Attr. dim.
ogbl-collab [11]	235,868	2,358,104	8.20	128
ogbl-citation2 [11]	2,927,963	30,387,995	20.73	128
USAir [21]	332	3,402	10.25	332
E-commerce [20]	346,439	238,818	1.38	768

by the inclusion of train targets and graph structure distraction. Implementation-wise, to ensure the scalability of our sampler and the usability of our sampler in real-world deployed systems, we implemented it as a subclass of DGL's edge sampler. This is equivalent to the DGL's original edge sampler and can be combined with other functions in DGL smoothly.

5.2 Test-time Right Practices: Exclude All Test Target Links

As discussed in Sec. 4.2, one may overlook the inclusion of test target edges in message passing or to add them into the graph to maintain no distribution shifts, but this practice can result in test leakage. We firmly assert that under no circumstance should test edges be utilized as message passing edges. In real-world deployed systems, the future links between nodes are unknown, making it imperative to avoid any utilization of test edges for message passing.

We propose that test target edges should always be excluded from the test message passing graph. To facilitate this, we provide a module that automatically checks for the presence of test target edges in the inference graph and removes them if necessary. The algorithms outlining this process are presented in Alg. 1. In previous literature [11], it was sometimes common to include validation edges in the message passing graphs to gather additional information, particularly for time-split data. The inclusion of validation edges in the message passing graph is not considered as data leakage. However, the decision of whether to use validation edges as message passing edges relies heavily on the user's preference. In our algorithm, we prompt users to deliberately specify their preference and return the desired inference graph accordingly.

Table 3: Train-Pitfall Results: Test performance of different training frameworks across GNN models and datasets. Per dataset, we report the recommended evaluation metric in the literature. *OOM = out of GPU memory

Model	ExcludeNone(Tr)	ExcludeAll	SpotTarget
	Ogbl-Collab (H@50 ↑)		
SAGE	48.57 ± 0.74	45.82 ± 0.41	49.00 ± 0.65
MB-GCN	43.03 ± 0.50	37.75 ± 1.42	39.58 ± 1.06
GATv2	45.61 ± 0.85	45.71 ± 0.87	45.46 ± 0.19
SEAL	61.27 ± 0.28	64.11 ± 0.30	64.57 ± 0.30
	Ogbl-Citation2 (MRR ↑)		
SAGE	82.06 ± 0.06	81.47 ± 0.17	82.18 ± 0.18
MB-GCN	79.70 ± 0.25	79.06 ± 0.30	79.88 ± 0.14
GATv2	OOM	OOM	OOM
SEAL	86.75 ± 0.20	86.74 ± 0.23	86.93 ± 0.55
	USAir (AUC ↑)		
SAGE	95.97 ± 0.17	95.71 ± 0.12	96.19 ± 0.53
MB-GCN	94.00 ± 0.14	94.09 ± 0.11	94.28 ± 0.15
GATv2	95.05 ± 0.66	95.66 ± 0.24	95.87 ± 0.46
SEAL	95.36 ± 0.24	95.94 ± 0.04	96.39 ± 0.09
Rank ↓	2.27	2.45	1.27

6 EXPERIMENTS

We aim to address the following questions through experiments.

- How well does SpotTarget address P1 and P2 on commonly benchmarked datasets?
- What is its performance in the e-commerce setting, where the most nodes have a very low degree?
- How well does SpotTarget address P1-P2 on edges incident to low-degree nodes specifically on commonly-used benchmarks?
- For test-related pitfall P3, what is the "fake" performance increase that is introduced by implicit data leakage?

We also provide analysis on the degree threshold δ in App. A.4 and show that SpotTarget has comparable computational complexity with baseline models in App. A.5.

6.1 Experimental Setup

Data. We evaluate our framework on four real-world datasets from multiple domains. Ogbl-Collab and Ogbl-Citation2 [11] are the author collaboration and citation networks. USAir [21] is a network of US Air lines. We notice that these datasets are relatively dense, with average node degree of 8-20. In real-world deployed systems, the observed data is typically more sparse, with many low-degree nodes. Hence, we adapt a sparse graph dataset from E-commerce [20]. E-commerce is a real-world dataset constructed according to shopping queries and related products in Amazon Search, which captures whether the queries are exact matches of the products. We give the details of dataset construction in Appendix A.1. Detailed statistics of all datasets are shown in Tab. 2 .

Table 4: Train-Pitfall Results: Test performance on the sparse E-commerce dataset. SpotTarget achieves a consistently better performance boost on E-commerce across metrics and datasets. For SAGE and GATv2, SpotTarget is up to 15× more accurate.

	SAGE		MB-GCN		GATv2	
Metrics	**ExcludeNone(Tr)**	**SpotTarget**	**ExcludeNone(Tr)**	**SpotTarget**	**ExcludeNone(Tr)**	**SpotTarget**
MRR ↑	4.40 ± 0.31	65.85 ± 0.31	17.07 ± 7.38	69.67 ± 0.52	5.98 ± 0.56	69.44 ± 0.55
H@10 ↑	6.55 ± 0.37	89.67 ± 0.19	28.35 ± 7.47	89.79 ± 0.25	9.64 ± 1.10	90.52 ± 0.26
H@1 ↑	3.04 ± 0.31	52.84 ± 0.46	10.83 ± 5.21	57.63 ± 0.57	3.94 ± 0.81	57.11 ± 1.03

Metrics. Following prior works, we use Mean Reciprocal Rank (MRR) on Ogbl-Citation2 and Hits@50 on Ogbl-Collab [11]. Area Under the Curve (AUC) is used for USAir [32]. For E-commerce, we choose to report MRR, Hits@10, and Hits@1, the three most commonly-used evaluation metrics for link prediction [11, 24, 34]. For all evaluation metrics, the higher the number is, the better.

GNN models. We select four GNN models to validate our method. SAGE [10], MB-GCN [13] and GATv2 [4] are GAE-based models. MB-GCN [13] is a mini-batch GCN model that follows the same message passing functions as GCN. At each iteration, only a portion of the entire graph is seen instead of the entire graph. We call it as MB-GCN, namely mini-batch GCN. SEAL [32] is a subgraph-based model that extracts an enclosing subgraph for each target edge and predicts the link likelihood based on the subgraph's embeddings.

Implementation Details. All GNN models are implemented in DGL [19, 26]. We conduct a comprehensive hyperparameter tuning and choose the best ones. More details are in Appendix A.2.

Baselines. For train pitfalls P1 and P2, we use ExcludeNone(Tr), ExcludeAll as our baselines, which indicates two different ways of including/excluding target edges. ExcludeNone(Tr) does not exclude any training target edges, and ExcludeAll exclude all train target edges T_{Tr}. Note that ExcludeAll on SEAL is essentially FakeEdge, which excludes all target edges on subgraph-based models.

For test pitfalls P3, our baseline is ExcludeNone(Tst), which uses the test target edges in the inference graph. Note that ExcludeNone(Tst) is the case where data leakage happens, which should be avoided at all time in real-world deployed systems.

SpotTarget Variants. At training time, our proposed framework SpotTarget excludes the set of target edges T_{low} incident to at least one low-degree node, where the degree threshold is denoted as δ. Experimentally, we choose δ to be 10 and 20, which corresponds to the average degrees of the datasets used, and report the best performing ones. More discussions of δ is in Appendix A.4.

At test time, our SpotTarget includes two variants ExcludeValTst and ExcludeTst. ExcludeValTst excludes both validation and test target edges in the test graph, while ExcludeTst only excludes test target edges in the test graph. Whether to use ExcludeValTst or ExcludeTst depends on the user's preference.

6.2 Train Pitfalls: Results on Dense Datasets

Setup. To evaluate SpotTarget's ability on addressing train pitfalls P1 and P2 on common benchmarks, we report the link prediction performance of four GNN models on three dense datasets and three trials for each setting. For Ogbl-Collab and Ogbl-Citation2, we generate one negative per edge during training and use the recommended negatives during evaluation. For USAir, we also generate one negative per edge during training, and during evaluation, we treat all edges that do not appear in the train,test,validation as negative edges. Besides the performance for each setting, we also report the average rank of our baselines ExcludeNone(Tr), ExcludeAll and proposed method SpotTarget. Results are summarized in Tab. 3.

Results. SpotTarget consistently achieves the best performance (lowest rank) across different datasets and models. This indicates SpotTarget's successful resolution of pitfalls P1 and P2 without corrupting the mini-batch graphs. Although ExcludeAll is used in the original implementation of SEAL, replacing ExcludeAll with SpotTarget leads to further performance improvement. It is worth noting that ExcludeAll generally leads to slightly lower performance compared to ExcludeNone(Tr). This is because excluding all target edges in a single mini-batch leads to a significant alteration of the graph structure, including the isolation of certain nodes. Hence, GNNs struggle to learn effective node representations.

Observation 1. **(1)** *SpotTarget outperforms both ExcludeNone(Tr) and ExcludeAll, achieving the best average results and the lowest rank. This suggests SpotTarget successfully resolving the pitfalls P1 and P2.* **(2)** *In some cases (6/11), ExcludeAll in the mini-batch graphs results in performance degradation due to graph structure corruption.*

6.3 Train Pitfalls: Results on Sparse Datasets

Setup. We repeat our experiments on E-commerce dataset with the similar setup in Section 6.2. Since more than 99.5% of the edges in E-commerce have a degree less than 5, ExcludeAll is the same as SpotTarget, so we only report ExcludeNone(Tr) and SpotTarget. We do not report SEAL for E-commerce because for a dataset with average degree of 1, it is impractical to construct subgraphs for each node. The results are shown in Table 4.

Results. SpotTarget achieves a *14.9* × performance boost across models on the sparse E-commerce graph. This observation demonstrates that SpotTarget is well-suited for training GNNs on the task of link prediction for sparse datasets, which are commonly encountered in real-world deployed systems. For example, commonsense knowledge graphs and biochemical graphs, with an average degree of 2 [8, 17], exemplify the prevalence of sparse graphs.

Observation 2. *SpotTarget has 14.9 × better performance compared to ExcludeNone across models. This empirical evidence confirms that low-degree nodes are more affected by P1 and P2, and excluding T_{low} proves to be effective, particularly for datasets with a significant number of low-degree nodes.*

6.4 Train Pitfalls: Results on Low-degree Nodes

Setup. To quantify how much low-degree nodes in dense datasets suffer from P1 and P2, we are interested in analyzing the performance of edges adjacent to low-degree nodes in dense graphs. We

Table 5: Train-Pitfall Results: We report the test MRR of SAGE on Ogbl-Citation2 on target edges which have at least one low-degree nodes. In addition to T_{low}, we also report cases where target edges are only incident to low-degree nodes: $max(d_i, d_j) < 10, 5$. SpotTarget achieves the best performance for T_{low} and target edges that are incident to both low-degree nodes.

	Exclusion	$max(d_i, d_j) < 10$	$max(d_i, d_j) < 5$	$min(d_i, d_j) < 10$	$min(d_i, d_j) < 5$	$min(d_i, d_j) = 2$	$min(d_i, d_j) = 1$
MRR ↑	ExcludeNone(Tr)	73.11 ± 0.25	62.15 ± 0.84	78.78 ± 0.12	69.54 ± 0.37	47.02 ± 0.56	27.54 ± 0.88
	ExcludeAll	77.45 ± 0.41	75.39 ± 1.42	79.17 ± 0.12	73.86 ± 0.33	60.05 ± 1.11	48.60 ± 1.11
	SpotTarget	78.08 ± 0.06	76.23 ± 0.56	79.30 ± 0.18	73.87 ± 0.18	61.48 ± 0.51	51.47 ± 2.51

Table 6: Test-Pitfall Results: We report the test results of four GNN models over three datasets. Note that despite ExcludeNone(Tst)'s good performance, it is essentially a case of data leakage because test target edges are never observed in real-world scenarios. We should avoid using test edges in the inference graph and SpotTarget helps you to automatically check this. *OOM = out of GPU memory.

Models	SpotTarget		Baseline
	ExcludeValTst	ExcludeTst	ExcludeNone(Tst)
	Ogbl-Collab (H@50 ↑)		
SAGE	48.57 ± 0.74	57.61 ± 0.88	83.82 ± 0.59
MB-GCN	43.03 ± 0.50	50.53 ± 1.10	75.41 ± 0.43
GATv2	45.61 ± 0.85	54.94 ± 0.19	84.16 ± 2.62
SEAL	57.50 ± 0.31	55.16 ± 1.94	99.91 ± 0.05
	Ogbl-Citation2 (MRR ↑)		
SAGE	82.06 ± 0.06	82.28 ± 0.11	89.22 ± 0.10
MB-GCN	79.70 ± 0.25	81.25 ± 0.22	88.32 ± 0.14
GATv2	OOM	OOM	OOM
SEAL	86.75 ± 0.20	87.01 ± 0.39	97.14 ± 0.18
	USAir (AUC ↑)		
SAGE	95.97 ± 0.17	95.51 ± 0.53	99.15 ± 0.59
MB-GCN	94.00 ± 0.14	94.11 ± 0.13	98.66 ± 0.22
GATv2	95.05 ± 0.66	94.07 ± 0.21	98.96 ± 0.11
SEAL	95.36 ± 0.24	95.10 ± 0.76	97.20 ± 0.78
No Leakage?	✓	✓	✗
Deployment	✓	✓	✗

report the performance of two different edge types: (1) edges that are adjacent to at least one low-degree node, i.e., $min(d_i, d_j) < \delta$ and; (2) edges that are only adjacent to low-degree nodes, i.e., $max(d_i, d_j) < \delta$. Our analysis is mainly done on Ogbl-Citation2 for SAGE. We compare SpotTarget against two baselines: ExcludeNone(Tr) and ExcludeAll. Results are shown in Tab. 5.

Results. In general, both ExcludeAll and SpotTarget achieve significantly better performance than ExcludeNone(Tr) for edges that are incident to low-degree nodes. This finding aligns with our earlier discussion in Sec. 5.1, which highlights that low-degree nodes are more adversely affected by train pitfalls P1 and P2 and excluding train target edges proves to be more advantageous for low-degree nodes. Specifically, when comparing SpotTarget and ExcludeAll, we observe that SpotTarget, which only excludes T_{low} instead of T_{Tr}, achieves superior performance on edges incident to low-degree nodes. This observation suggests that SpotTarget excels in preserving the graph structures during mini-batch training.

Observation 3. *The performance improvement observed on edges adjacent to low-degree nodes in dense graphs demonstrates the successful resolution of P1 and P2 for low-degree nodes in dense graphs by our SpotTarget. Furthermore, SpotTarget can be applied to mitigate performance bias in individuals on the periphery of the community.*

6.5 Test Pitfalls: Leakage Quantification

Setup. Besides training pitfalls, we are interested in quantifying the performance gap introduced by the data leakage from the test pitfall (P3). We report results on excluding different types of edge in the inference graph (validation, test edges). Although here we are not evaluating in a deployed system, by excluding different types of edge, we are mimicking what would happen in a deployed system. All GNN models are trained using train edges only. ExcludeValTst excludes all validation and test target edges during inference, and ExcludeTst only excludes validation edges. Both ExcludeValTst and ExcludeTst are variants of SpotTarget. ExcludeNone(Tst) keeps all validation and test target edges during testing, resulting in data leakage(P3) and should be avoided in deployed systems.

Results. Utilizing validation targets as message passing edges in inference graphs typically leads to a marginal performance improvement, consistent with previous findings [11]. In the case of Ogbl-Collab, a dataset split by time, utilizing validation targets as message passing edges results in a 10% performance boost. However, when test targets are employed as message passing edges in the inference graph, it yields a performance boost of up to 20%. However, the performance improvement resulting from the incorporation of test targets is undesirable. In real-world deployed systems, test links are unobserved and cannot be utilized for message passing. Particularly, in the case of SEAL, the presence of test targets in the inference phase leads to near-perfect results, which is what we aim to prevent. SpotTarget effectively addresses P3.

Observation 4. *Utilizing test edges results in a fake performance boost across multiple datasets, which is caused by data leakage (P3). The performance boost is particularly significant in time-split datasets such as Ogbl-Collab. The increased performance validates the importance of SpotTarget, which consistently excludes the test target edges from the inference graphs during testing. In real-world deployed systems, future (test) links are always unobserved. If the model utilizes information from test target edges, it undeniably generates an artificial performance improvement that does not reflect real-life scenarios.*

Conclusion. Our study systematically examines the challenges of using target edges as message passing edges in GNNs. We empirically and theoretically establish that low-degree nodes are more susceptible to these issues and propose SpotTarget. SpotTarget strikes a balance between eliminating the challenges introduced by target edges and preserving the integrity of mini-batch graphs.

Acknowledgements We thank Yongyi Yang for discussing and proofreading our theoretical proof.

REFERENCES

[1] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.

[2] Derrick Blakely, Jack Lanchantin, and Yanjun Qi. 2021. Time and Space Complexity of Graph Convolutional Networks. *Accessed on: Dec* 31 (2021).

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013).

[4] Shaked Brody, Uri Alon, and Eran Yahav. 2021. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491* (2021).

[5] Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak. 2018. Hyperspherical variational auto-encoders. *arXiv preprint arXiv:1804.00891* (2018).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Kaiwen Dong, Yijun Tian, Zhichun Guo, Yang Yang, and Nitesh V Chawla. 2022. FakeEdge: Alleviate Dataset Shift in Link Prediction. *arXiv preprint arXiv:2211.15899* (2022).

[8] Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. 2022. Long range graph benchmark. *arXiv preprint arXiv:2206.08164* (2022).

[9] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[11] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[12] Vassilis N Ioannidis, Xiang Song, Da Zheng, Houyu Zhang, Jun Ma, Yi Xu, Belinda Zeng, Trishul Chilimbi, and George Karypis. 2022. Efficient and effective training of language and graph neural network models. *arXiv preprint arXiv:2206.10781* (2022).

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

[14] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[15] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[16] David Liben-Nowell and Jon Kleinberg. 2003. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*. 556–559.

[17] Chaitanya Malaviya, Chandra Bhagavatula, Antoine Bosselut, and Yejin Choi. 2020. Commonsense knowledge base completion with structural and semantic context. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 2925–2933.

[18] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. 2016. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)* 49, 4 (2016), 1–33.

[19] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[20] Chandan K. Reddy, Lluís Màrquez, Fran Valero, Nikhil Rao, Hugo Zaragoza, Sambaran Bandyopadhyay, Arnab Biswas, Anlu Xing, and Karthik Subbian. 2022. Shopping Queries Dataset: A Large-Scale ESCI Benchmark for Improving Product Search. (2022). arXiv:2206.06588

[21] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 385–394.

[22] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Investigating and mitigating degree-related biases in graph convoltuional networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1435–1444.

[23] Komal Teru, Etienne Denis, and Will Hamilton. 2020. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*. PMLR, 9448–9457.

[24] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2019. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082* (2019).

[25] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[26] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[27] Xiyuan Wang, Haotong Yang, and Muhan Zhang. 2023. Neural Common Neighbor with Completion for Link Prediction. *arXiv preprint arXiv:2302.00890* (2023).

[28] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*. PMLR, 5453–5462.

[29] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, and Chao Chen. 2021. Link prediction with persistent homology: An interactive view. In *International Conference on Machine Learning*. PMLR, 11659–11669.

[30] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. In *International conference on machine learning*. PMLR, 7134–7143.

[31] Xiangxiang Zeng, Xiang Song, Tengfei Ma, Xiaoqin Pan, Yadi Zhou, Yuan Hou, Zheng Zhang, Kenli Li, George Karypis, and Feixiong Cheng. 2020. Repurpose open data to discover therapeutics for COVID-19 using deep learning. *Journal of proteome research* 19, 11 (2020), 4624–4636.

[32] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in neural information processing systems* 31 (2018).

[33] Muhan Zhang and Yixin Chen. 2019. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058* (2019).

[34] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2021. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems* 34 (2021), 9061–9073.

[35] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 36–44.

[36] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems* 34 (2021), 29476–29490.

A APPENDIX

A.1 Dataset Construction

For E-commerce, over 94% of the nodes have a degree less than 5 and more than 99.5% of the edges in E-commerce are adjacent to a node with a degree less than 5. The E-commerce dataset is constructed as follows. We first adopt the ESCI dataset which categorizes the relationship between between shopping queries and products as "exact", "substitute", "complement", and "irrelevant". We extract the shopping queries and their "exact" matching products only, and randomly split them to train, validation, and test splits according to a 70%/10%/20% ratio. We also remove all isolated nodes in the dataset. We use the pretrained BERT embeddings [6] as text features.

A.2 Hyperparameter Tuning

We conduct extensive hyperparameter tuning using grid search. We search on the learning rates = {1e-1, 1e-2, 1e-3, 1e-4, 5e-4, 5e-5} and the number of layers = $\{1, 2, 3\}$, hidden dimension = $\{128, 256, 512, 1024\}$. We report the best performing hyperparameters for each setting.

Throughout the experiments, we use the numbers of training batch size for ogbl-collab, ogbl-citation2, USAir and E-commerce is set to 256, 1024, 64 and 512, respectively. For SAGE models, the learning rate is 5e-4, the dimension of hidden layers is 256, and the number of hidden layers is 3. For GCN, the number of hidden layers is 1 and others remain the same as the SAGE. For GATv2, the number of heads is 8, the learning rate is 5e-4, the dimension of hidden layers is 64. For SEAL, the learning rate is 1e-4, the number of hops is 1, the training percent is 15%, 2% and 100% and the

dimension of hidden layers is is 256, 256 and 32 for ogbl-collab, ogbl-citation2, and USAir respectively.

Note that our result on FakeEdge is a bit lower than what they report because (1) we use a different split of USAir than theirs because there is no public splits for USAir. (2) For their reported FakeEdge results, they set number of hops to be 2 and hidden channel to be 128, as well as using all training sets. We found this to be extremely computation intensive and cannot be run on larger datasets like Ogbl-Citation2. So in our reported results, we follow the hyperparameters that SEAL uses, namely setting number of hops to be 1 and hidden channel to be 32.

We used a single Nvidia A40 GPU to train the model and repeat our experiments with three different random seeds. The test results are reported on the epoch with the best performance in the validation set.

A.3 Quantitative Analysis of Average Degree Change

We further support our claim that low-degree nodes are more affected by providing a quantitative analysis on the relative changes of degree before and after excluding the train target edges T_{Tr}. The analysis is done in four datasets of various scales and sparsity. For each dataset, we sort its nodes according to their degrees and report the average degree change before and after excluding the train target edges for each mini-batch epoch. As shown in Fig. 3, for nodes with lower degrees, the relative degree change can go as high as 100 % while for high-degree nodes, the relative degree change is pretty small.

A.4 Ablation: Which Degree to Use?

Setup. For our SpotTarget at training time, we only exclude edges with adjacent nodes smaller than a degree threshold δ. But what will be a good threshold δ to use? In order to answer this question, we conduct experiments on USAir with varying degree thresholds. The results are shown in Fig. 4.

Results. When $\delta = 0$, we do not exclude any target edges, and when $\delta = +\infty$, we exclude all train target edges. As shown in Fig. 4, we see that as we exclude target edges with a higher degree threshold (exclude more target edges), the performance of the model will first go up and then go down, forming a U-shape curve. This indicates that we need to strike a balance between eliminating the issues brought by the target edges as well as not corrupting the mini-batch graphs too much. We found that the best choice of δ is typically the average degree of the dataset.

Observation 5. *To get the best effects of our SpotTarget, we need to conduct the sensitivity check to discover the optimal degree threshold δ. The optimal threshold may be positively correlated with the average node degrees of the graph.*

A.5 Time Complexity Analysis

The additional computational complexity of SpotTarget is due to the target edge exclusion part. For each training iteration, we need to iterate over the edges in the mini-batch to examine whether they are incident to low-degree nodes. The time complexity of excluding target edges is $O(|B|)$, where $|B|$ is the number of edges in the message passing graph. The time complexity of training in the ExcludeNone(Tr), ExcludeAll and SpotTarget frameworks is similar since the time of additional edge exclusion is much smaller compared with the model training time. The difference of number of edges in the message passing graph only make marginal changes of GNNs training time [2].

B EXTENDED THEORETICAL ANALYSIS

We first prove theorem 1 on GCN and then extend the proof into general message-passing GNN models.

B.1 GCN

Proof. Here, we want to prove the when a neighboring edge of a node is removed in order to eliminate the train pitfalls: overfitting (P1) and distribution shift (P2)., the changes on high degree nodes is smaller than the change on low degree nodes.

We first define the overall influence of node v_k on node v_h after L-th layer GCN as $\frac{\partial x_h^\Lambda}{\partial x_k}$ [22, 28].

According to [22], we have that the partial derivative of x_h to x_k for an Λ-th layer untrained GCN is

$$\frac{\partial x_{h,s}^\Lambda}{\partial x_{k,t}} = \sqrt{d_h d_k} \sum_{p=1}^{\Psi} \prod_{\lambda=\Lambda}^{0} \frac{1}{d_{p^\lambda}} \text{diag}(\mathbb{1}_{\sigma_\lambda})_{s,s} \mathbf{W}_{s,t}^\lambda \tag{1}$$

for all $1 \le s, t \le n$. Here $\text{diag}(\mathbb{1}_{\sigma_\Lambda})$ is a diagonal mask matrix representing the activation result, Ψ is the set of all $(\Lambda+1)$-length random-walk paths on the graph from node v_h to v_k , and p^Λ represents the Λ-th node on a specific path p (p^0 and p^Λ denote node i and k accordingly).

Excluding one neighboring edges of node v_h would bring two changes: (1) the degree of node v_h will decrease to $d_h - 1$ as one of its neighbors is removed, and (2) There will be less random walk paths from node v_h to v_k, $|\tilde{\Psi}| < |\Psi|$.

Thus we have

$$1 - \mathbb{E}\left(\frac{\partial \tilde{x}_{h,s}^\Lambda}{\partial x_{k,t}} \Big/ \frac{\partial x_{h,s}^\Lambda}{\partial x_{k,t}}\right) = 1 - \frac{\sqrt{(d_h-1)d_k} \sum_{p=1}^{\tilde{\Psi}} \mathbb{E}\left(\prod_{\lambda=\Lambda}^{0} \frac{1}{d_{p^\lambda}} \text{diag}(\mathbb{1}_{\sigma_\lambda})_{s,s} \mathbf{W}_{s,t}^\lambda\right)}{\sqrt{d_h d_k} \sum_{p=1}^{\Psi} \mathbb{E}\left(\prod_{\lambda=\Lambda}^{0} \frac{1}{d_{p^\lambda}} \text{diag}(\mathbb{1}_{\sigma_\lambda})_{s,s} \mathbf{W}_{s,t}^\lambda\right)} \tag{2}$$

From [22], we have $\sum_{p=1}^{\Psi} \mathbb{E}\left(\prod_{\lambda=\Lambda-1}^{0} \frac{1}{d_p^\lambda} \text{diag}(\mathbb{1}_{\sigma_\lambda})_{s,s} \mathbf{W}_{s,t}^\lambda\right) = v$ is a constant.

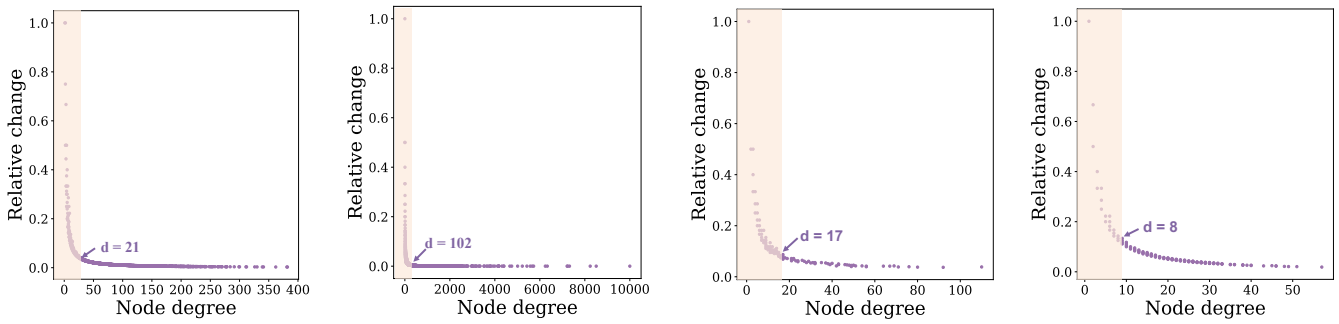
Eq. 2 can rewritten as

$$1 - \mathbb{E}\left(\frac{\partial \tilde{x}_{h,s}^\Lambda}{\partial x_{k,t}} \Big/ \frac{\partial x_{h,s}^\Lambda}{\partial x_{k,t}}\right) = 1 - \sqrt{\frac{d_h - 1}{d_h}} \frac{1/(d_h-1)\text{diag}(\mathbb{1}_{\sigma_\Lambda})_{s,s} \mathbf{W}_{s,t}^\Lambda \sum_{v_n \in \tilde{N}(h)} v}{1/(d_h)\text{diag}(\mathbb{1}_{\sigma_\Lambda})_{s,s} \mathbf{W}_{s,t}^\Lambda \sum_{v_n \in N(h)} v} \tag{3}$$

Then we have

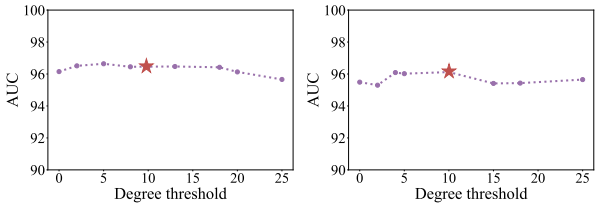
$$\begin{aligned} 1 - \mathbb{E}\left(\frac{\partial \tilde{x}_{h,s}^\Lambda}{\partial x_{k,t}} \Big/ \frac{\partial x_{h,s}^\Lambda}{\partial x_{k,t}}\right) &= 1 - \sqrt{\frac{d_h - 1}{d_h}} \frac{1/(d_h-1)(d_h-1)v}{1/(d_h)d_h v} \\ &= 1 - \sqrt{1 - \frac{1}{d_h}} \end{aligned} \tag{4}$$

Since if $d_h > d_l$, we can deduce $\sqrt{1 - \frac{1}{d_h}} > \sqrt{1 - \frac{1}{d_l}}$, thus $1 - \mathbb{E}\left(\frac{\partial \tilde{x}_{h,s}^\Lambda}{\partial x_{k,t}} \Big/ \frac{\partial x_{h,s}^\Lambda}{\partial x_{k,t}}\right) < 1 - \mathbb{E}\left(\frac{\partial \tilde{x}_{l,s}^\Lambda}{\partial x_{k,t}} \Big/ \frac{\partial x_{j,s}^\Lambda}{\partial x_{k,t}}\right)$ and $D(k,h) < D(k,l)$ hold. □



(a) Ogbl-Collab (b) Ogbl-Citation2 (c) USAir (d) E-commerce

Figure 3: The average degree change of nodes when excluding training targets. The X axis the degree of the nodes in the sorts, sorted from lowest to highest and the Y axis is the relative changes in degree before and after excluding all of the train target edges in each mini-batch. Lower degree nodes get higher relative degree change, for nodes with degree less than 5, the relative degree change can go as high as 100%.



(a) USAir with SAGE model (b) USAir with GATv2 model

Figure 4: The performance with varying degree thresholds. We see a U-shape effect, which works best if you only exclude the train target edges T_{low}. The red star indicates the average degree of the dataset USAir. Empirically, we find empirically selecting the threshold δ to be the average degree of the dataset works well.

B.2 GNNs

Tn the previous section, we have proved that excluding a target edge can cause a larger influence on low-degree nodes than high-degree nodes with the GCN model. In this part, we will prove that Theorem 1 can be extended to general GNN models.

Proof. For general GNNs, the output node features of the Λ-th layer are generated as follows:

$$x_h^{\Lambda+1} = \sigma(W^{\Lambda} \sum_{v_a \in N(h)} \alpha_{a,h} x_a^{\Lambda}) \tag{5}$$

where α could be the constant or pre-defined parameters related with graph attributes such as node degrees or the parameters will be learned, such as attention scores. The effect of node v_k on node v_h can be expressed as follows:

$$\mathbb{E}(\frac{\partial x_h^{\Lambda}}{\partial x_k}) = v d_k \sum_{v_n \in N(h)} \alpha_{n,h} \cdot \text{diag}(\mathbb{1}_{\sigma_\Lambda}) \cdot \mathbf{W}^{\Lambda} \tag{6}$$

For the effect of node v_k after excluding one target edge, the cardinality of the set of v_h neighbor nodes decreases from N to $N-1$ and the value of α may also change to $\tilde{\alpha}$.

$$\mathbb{E}(\frac{\partial \tilde{x_h}^{\Lambda}}{\partial x_k}) = v d_k \sum_{v_n \in N(\tilde{h})} \tilde{\alpha_{n,h}} \cdot \text{diag}(\mathbb{1}_{\sigma_\Lambda}) \cdot \mathbf{W}^{\Lambda} \tag{7}$$

We have the effect ratio is

$$\begin{aligned} \mathbb{E}(\frac{\partial \tilde{x_{h,s}}^{\Lambda}}{\partial x_{k,t}} / \frac{\partial x_{h,s}^{\Lambda}}{\partial x_{k,t}}) &= \frac{d_k(\sum_{v_n \in N(\tilde{h})} \tilde{\alpha_{n,h}} \text{diag}(\mathbb{1}_{\sigma_\Lambda})_{s,s} \mathbf{W}_{s,t}^{\Lambda} v)}{d_k(\sum_{v_n \in N(h)} \alpha_{n,h} \text{diag}(\mathbb{1}_{\sigma_\Lambda})_{s,s} \mathbf{W}_{s,t}^{\Lambda} v)} \\ &= \frac{\sum_{v_n \in N(\tilde{h})} \tilde{\alpha_{n,h}}}{\sum_{v_n \in N(h)} \alpha_{n,h}} \end{aligned} \tag{8}$$

If the value of α is unrelated with the degree of v_h, then the theorem holds since $\alpha = \tilde{\alpha}$ the expectation of the ratio is $\frac{d_h - 1}{d_h}$ and $D(k,h) < D(k,l)$. If the value of $\alpha \propto (d_h)^m$ then we have the raio is $\frac{(d_h-1)(d_h-1)^m}{d_h(d_h)^m}$. If $m >= -1$, we still have the theorem holds. To our best knowledge, we do not find the existing GNNs formula with $\alpha \propto (d_h)^m$ as well as $m < -1$, so our theorem should hold for general GNNs. □