Pitfalls in Link Prediction with Graph Neural Networks: Understanding the Impact of Target-link Inclusion & Better Practices

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

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

Vassilis N. Ioannidis
Amazon
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
Amazon AWS
xiangsx@amazon.com

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

ABSTRACT

While Graph Neural Networks (GNNs) are remarkably successful in a variety of high-impact applications, we demonstrate that, in link prediction, the common practices of including the edges being predicted in the graph at training and/or test have outsized impact on the performance of low-degree nodes. We theoretically and empirically investigate how these practices impact node-level performance across different degrees. Specifically, we explore three issues that arise: (I1) *overfitting*; (I2) *distribution shift*; and (I3) implicit *test leakage*. The former two issues lead to poor generalizability to the test data, while the latter leads to overestimation of the model's performance and directly impacts the deployment of GNNs. To address these issues in a systematic way, we introduce an effective and efficient GNN training framework, SpotTarget, which leverages our insight on low-degree nodes: (1) at training time, it excludes a (training) edge to be predicted if it is incident to at least one low-degree node; and (2) at test time, it excludes *all* test edges to be predicted (thus, mimicking real scenarios of using GNNs, where the test data is not included in the graph). SpotTarget helps researchers and practitioners adhere to best practices for learning from graph data, which are frequently overlooked even by the most widely-used frameworks. Our experiments on various real-world datasets show that SpotTarget makes GNNs up to 15× more accurate in sparse graphs, and significantly improves their performance for low-degree nodes in dense graphs.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

graph neural network; link prediction; shortcut learning

WSDM '24, March 4–8, 2024, Merida, Mexico

ACM ISBN 979-8-4007-0371-3/24/03…$15.00
https://doi.org/10.1145/3616855.3635786

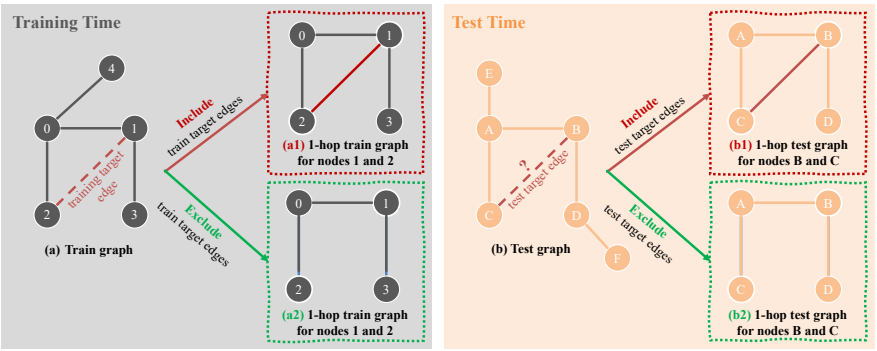
ACM Reference Format:
Jing Zhu*, Yuhang Zhou*, Vassilis N. Ioannidis, Shengyi Qian, Wei Ai, Xiang Song, and Danai Koutra. 2024. Pitfalls in Link Prediction with Graph Neural Networks: Understanding the Impact of Target-link Inclusion & Better Practices. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM '24), March 4–8, 2024, Merida, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3616855.3635786

1 INTRODUCTION

Graphs or networks are key representations for relational data that occur in many 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 many downstream applications such as drug repurposing, recommendation systems, and knowledge graph completion [1, 2, 17, 20, 22, 35]. It is also widely used as a pre-training method to produce high-quality entity representations that can be used in various business applications [12, 13]. Techniques to solve this task range from heuristics–e.g., predicting links based on the number of common neighbors between a pair of nodes–to graph neural network (GNN) models , which rely on message passing and leverage both the graph structure and node features. In recent years, GNN-based methods, which formulate the link prediction problem as a binary classification problem over node pairs, have led to state-of-the-art performance in many high-impact applications and have become the go-to approach both in industrial settings and academia [14, 16, 36, 38].

In this work, we focus on key pitfalls when training GNN models for the link prediction task, which we have found to cause significant disparities in node-level performance. Specifically, we investigate the common practices of including in the graph the *target links* (i.e., the edges for which the existence or absence is being predicted) at training and/or test time, and considering them during message passing [5, 36]. The inclusion of (training) target links at training time leads to two major issues, *overfitting* (I1) and *distribution shift* (I2), while the inclusion of target edges in the test graph data causes *implicit data leakage* (I3) through neighborhood aggregation. In turn, these issues lead to poor performance for GNN models and inability to effectively generalize to (truly) unobserved links at test time. We give an illustrative example of the issues in message passing in Fig. 1.

* equal contribution



Training Target Edges	Test Target Edges	Results	Issues
Include (a1)	-	✗	(I1) Overfitting
Include (a1)	Exclude (b2)	✗	(I2) Distribution shift
-	Include (b1)	✗	(I3) Leakage
Exclude (a2)	Exclude (b2)	✓	–

Figure 1: The pitfalls of including target links as message-passing edges during training or test time, and the issues that arise from these practices. [Left] Training time: Given a toy train graph and training target edge e_{12} in (a), we illustrate the impact of the inclusion (a1) and exclusion (a2) of e_{12} on the 1-hop induced train graph for nodes 1 and 2, which is used for message passing. [Right] Test time: We give the same illustration for a test graph and test target edge e_{BC} in (b). [Table] Overview of the three main issues and when they arise: (I1) When including train targets, GNNs *overfit* them instead of making predictions based on the graph structure and node features. (I2) When train target links are present but test target edges are absent, there is a *distribution shift* between training and testing. (I3) The presence of test target links causes *implicit test leakage.*

Illustrative Example. *In Fig. 1(a), e_{12} is a training target link 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* 1 *and* 2*, 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 B and C, where the two nodes are disconnected. This leads to distribution shift: there is discrepancy between the message-passing graphs used during training and testing despite the similarity between the target links.*

On the other hand, at test time, including the test target links in the test graph (e.g., edge e_{BC} in Fig. 1(b1)) causes data leakage. In our example, during neighborhood aggregation, the target node B would aggregate the messages from C and vice versa, resulting in a higher likelihood of predicting the existence of edge e_{BC} compared to the case where the link does not exist in the message-passing graph. However, in real-world applications, the goal is to predict future *links that are* not observed *in the data, so the inclusion of test target links corresponds to implicit data leakage.*

The pitfalls of including the target links in the graph at train and/or test time are commonplace in many GNN-based frameworks. For example, PyTorch Geometric (PyG) [8], a commonly-used repository, does not support excluding target edges when constructing the mini-batch graphs for training. Another popular library, DGL [30], for the first four years of its existence, did not include the function of excluding training target links in the official code examples that have been used by numerous researchers and practitioners. The majority of papers fail to reference the exclusion of target links as a consideration in their empirical analyses, and, anecdotally, multiple authors with both industry and academic experience have observed that these pitfalls often occur in practice. Although there have been efforts to deliberately eliminate the test-time pitfall in some popular benchmarks [11], it is still a commonly overlooked problem in applications that rely on proprietary data. Data contamination has also been a major issue in model evaluation, especially in the era of large language models[9, 26, 40, 41].

We demonstrate theoretically and empirically that low-degree nodes suffer more from the inclusion of target edges as it causes more significant relative degree changes for them compared to other nodes. Intuitively, for high-degree nodes, the target links that are erroneously considered have limited impact on the performance since they are only a small fraction of the edges considered during message passing. Thus, these practices significantly impact real-world applications, where the observed data are often incomplete and very sparse, with many low-degree nodes [7, 18, 19, 24]. To address the three issues (I1-I3), we introduce a GNN training framework, SpotTarget, to systematically and efficiently exclude the target links at training and test time, as well as check if target test edges are excluded for any user-defined dataset. Although excluding all training target links is an ideal solution, our analysis indicated that it significantly corrupts the mini-batch graph and renders learning with GNNs challenging. Our theoretical and empirical analysis shows that excluding the target links that are incident to at least one low-degree node achieves the best trade-off between avoiding the issues (I1, I2) and learning powerful node representations at training time. At test time, we argue that it is important to mimic real scenarios and avoid leakage for *all* target edges by excluding them from the test graph. Our key contributions are:

- **Systematic Analysis of the Target Link Inclusion Practices**: Focusing on link prediction, we perform the first thorough theoretical and empirical analysis on the effect of including target

edges as message-passing edges at training and test time. Our key insight is that low-degree nodes tend to suffer more from the issues that arise from these pitfalls.

- **Efficient Unified Framework**: We introduce the first unified GNN training framework, SpotTarget, which automatically tackles these issues at training and test time. During training, for efficiency, SpotTarget leverages our theoretical insight and excludes target links incident to at least one low-degree node. At test time, it excludes all target edges. These strategies ensure generalizable and robust model training without any data leakage issues. SpotTarget is also easy-to-use and scalable, and helps researchers and practitioners adhere to best practices, which are frequently overlooked even by the most widely-used GNN frameworks. We integrated it as a plug-and-play module in DGL.
- **Extensive Experiments**: To quantify the effect of including the target links as message-passing edges during training and test time, we conduct extensive experiments on various datasets, spanning from commonly-used link prediction benchmarks to real-world datasets. We show that SpotTarget makes GNN models up to 15× more accurate on sparse graphs, and significantly improves their performance for 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 GNNs 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 [4, 16, 29, 34, 42, 43]. Enclosing subgraph-based methods, including SEAL [36, 38], IGMC [37], GraIL [28], TCL-GNN [33], first extract an enclosing subgraph for the target edge, then apply GNNs to encode the representations of the nodes in enclosing subgraph, and finally aggregate the node representations by pooling methods. The learned subgraph features are fed into a classifier to predict the existence or absence of the target edge. Even though enclosing subgraph-based methods such as SEAL give more accurate predictions, GAE-based methods are typically orders of magnitude faster to compute and require fewer computation resources. In real-world applications, graphs are often massive with many millions of nodes or even billions of nodes, so typically GAE-based methods are employed [39].

Issues in Link Prediction using GNNs. Unlike node classification where edges are solely used as message-passing edges, edges in link prediction have two separate roles: (1) message passing and (2) prediction objectives. This distinction is often overlooked; GNNs designed for node classification tasks are often adapted for link prediction by simply stacking a decoder function, without explicitly handling the message passing and target links separately. The training pitfalls caused by the existence of target edges were initially identified by SEAL [31, 36], which made efforts to mitigate them through negative injection. Building upon it, FakeEdge [5] discussed the distribution shift issue that occurs due to the presence of target links during training and the absence of target links at test time. They further proposed to always add or remove the target links, or combine the strategies for subgraph-based methods like SEAL. Unlike these works, we focus on performing a thorough and systematic analysis of all the issues caused by including target links at training and/or test time, and characterizing the disparate impact of these practices on the performance of nodes of varying degrees. Moreover, unlike SEAL and FakeEdge that only apply to subgraph-based models, our SpotTarget aims to systematically and efficiently address the issues for more scalable GAE-based models, which are commonly-used in real-world applications (e.g., web-scale recommender systems). For example, our training framework is 10x faster than FakeEdge (2 hours for one epoch vs. 20 hours on Ogbl-Citation2).

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}

3 PRELIMINARIES

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

3.1 Definitions

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

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 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**. We adopt the widely-used train-validate-test setting, where only the epoch that achieves best performance on validation links is evaluated on test edges.

In this paper, 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 real-world applications).
(2) **Target links that are incident to at least one low-degree node**: Based on our theoretical insights in Sec. 5.1, our framework leverages target links 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}.

Graph Neural Networks. GNNs utilize a neighborhood aggregation scheme to learn a representation h_v for each node v. Node representation is 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. . 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 links, 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

More formally, we tackle the following problem: Given a graph G, a link prediction task, and a base GNN model in a mini-batch training setting, we seek to: (1) investigate the issues that arise from the common practices of including the target links T_{tr} and T_{tst} as message-passing edges at training and test time, respectively, and (2) propose an efficient, unified and easy-to-use solution that automatically addresses these issues.

4 ISSUES OF TARGET-LINK INCLUSION

In this section, we aim to explore the three issues that occur in link prediction with GNNs due to the practices of including the target links as message-passing edges at training and/or test time.

4.1 Issues during Training Time

The presence of the training target links in the train graph data and their use as message-passing edges cause overfitting as well as distribution shift.

(I1) 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 neighbors' 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. Since the training objective is to learn as high probability as possible for a link existing between node 1 and 2, GNNs would learn to overfit the training objective in order to predict the existence of the training target link. 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. When a target link is present in the enclosing subgraph as a message-passing edge, these models also suffer from overfitting issues. The overfitting issue leads to poor model generalizability to test data.

(I2) Distribution Shift. In typical GNN training processes for link prediction, the train target links T_{tr} are present and used during message passing, while the test target links T_{tst} are absent and never used during test. This practice poses a distribution shift problem. As an example, we consider the train graph in Fig. 1(a) along with e_{12} as the train target link, and the test graph with e_{BC} as the test target link in Fig. 1(b). As shown in Fig. 1(a1), at training time, when node 1 aggregates the messages from its neighbors, node 2 is among its direct neighbors, and the message from node 2 contributes to the computation of node 1's embeddings. In a realistic test scenario (Fig. 1(b2)), future links are not observed in the test data; so, when node B aggregates the messages from its neighbors, it does not include any message from node C as the latter is not a direct neighbor. This poses a distribution shift between training and testing, and also results in poor GNN model generalizability.

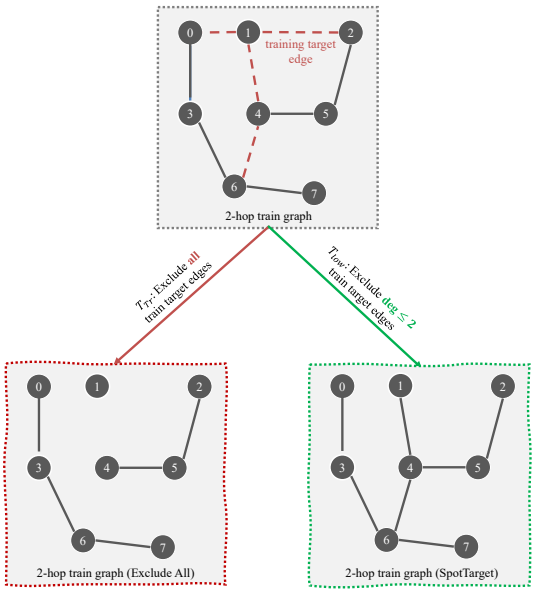


Figure 2: Example 2-hop message-passing graph for a mini-batch of size 4. Red lines are train target links and black lines correspond to other message-passing edges induced by the target edges. As shown on the left, excluding all target (red) links T_{Tr} during training results in three disconnected components. As shown on the right, if only edges incident to low-degree nodes are excluded T_{low} (e.g., deg ≤ 2), the graph connectivity is preserved. Our proposed solution avoids significant corruption of the graph structure while simultaneously avoiding issues (I1) and (I2).

4.2 Issues during Test Time

At test time, including the test targets links, T_{tst}, in message passing results in implicit data leakage.

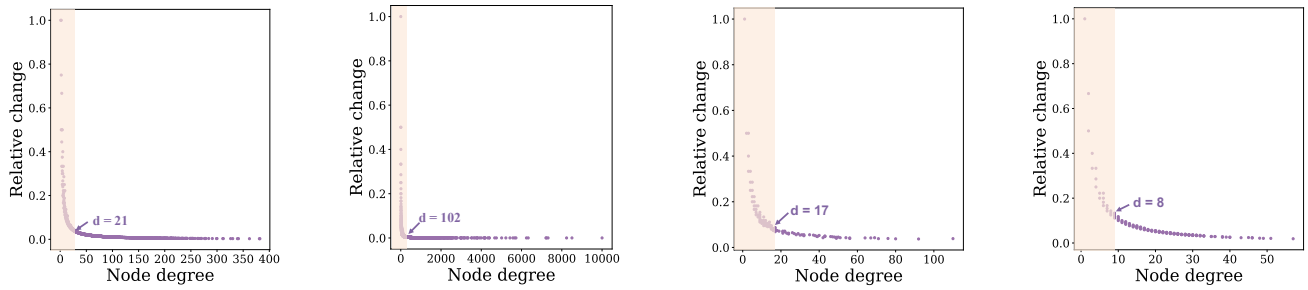
(I3) Data Leakage. As shown in Fig. 1(b1), when test target e_{BC} exists in the test message-passing graph, the target node B would aggregate messages from C and vice versa, which results in a higher likelihood of predicting a link between nodes B and C during inference, compared to the case when e_{BC} does not exist in the test graph in Fig. 1(b2). This leads to overestimation of the model's predictive performance and directly impacts the deployment of GNN models since *future* links that need to be predicted are never observed in real-world applications.

5 PROPOSED FRAMEWORK: SpotTarget

In this section, we present SpotTarget, the first framework that systematically resolves the issues arising from the presence of target links in the message-passing graph for link prediction. We propose separate solutions that are tailored to training and inference time.

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

As discussed in Sec. 4.1, the practice of including train target links T_{Tr} as message-passing edges causes overfitting and distribution



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

Figure 3: Average degree change for nodes when excluding training target links. The Y-axis corresponds to the relative change in degree before and after excluding all of the train target links in each mini-batch. Lower-degree nodes have higher relative degree change; for nodes with degree less than 5, the relative degree change is as high as 100%.

shift (I1-I2). One straightforward solution is to exclude all train target edges during training. However, this poses several challenges for both mini-batch and full-batch settings:

- First, for mini-batch training, an excluded link could be a message-passing edge of another target edge. For example, in Fig. 2, e_{14} is both a target edge and a message-passing edge for node 4 and node 1. The existence of e_{14} affects the message-passing graphs, and, in turn, the learning of target edges e_{46}, e_{01} and e_{12}. Excluding all the target edges causes significant corruption of the graph structure. In an extreme case, some nodes become isolated nodes, such as node 2 in Fig. 2. As a result, GNN models may fail to learn good representations when all target edges are excluded.
- Second, in full-batch training, if all edges are used as training target edges, then excluding all edges will result in a graph with only nodes and no edges, which is impractical. If only a portion of edges are used as training edges, the graph structure corruption caused by excluding all target edges still applies to full-batch settings. In full-batch training, it requires iterating over all edges in the graph to remove the target edges per training step, which is especially time-consuming. In practice, full batch training for link prediction on massive graphs is rare, since it is inefficient in terms of time and space complexity. As a result, we only consider mini-batch training in our proposed framework, SpotTarget.
- Third, although setting the batch size to 1 can solve the structure corruption, the mini-batch message-passing graph would become too small, causing inefficiency and instability for GNN training.

The question then becomes: *How can we achieve the best trade-off between avoiding issues (I1, I2) caused by the presence of train target links and preserving the graph structure in mini-batch training as much as possible?* The key insight to tackle this problem lies in identifying which nodes are mostly affected by issues (I1, I2), and only excluding target links incident to those nodes. At a high level, we show theoretically and empirically that low-degree nodes are impacted most by the inclusion of target link edges, as it causes more significant relative degree changes for them compared to other nodes. Excluding the target links incident to low-degree nodes achieves the best trade-off: since they have few neighbors, there is generally a small probability that the excluded target links that are incident to them are message-passing edges of another node in the mini-batch training. Next, we provide a theoretical and quantitative analysis to show that low-degree nodes are affected most by the issues, and target links in T_{low} should be excluded during training.

Theoretical Analysis. We begin by explaining from a theoretical perspective why primarily low-degree nodes suffer from the issues caused by the inclusion of train targets compared to high-degree nodes. Intuitively, we compare the change in influence that a random node v_k has on a high-degree node v_h and a low-degree node v_l before and after excluding an edge incident to v_h and v_l, respectively. We leverage the notion of influence/effect functions in statistics [27, 32] 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 degrees $d_h > d_l$, and 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 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 nodes v_h, v_k, respectively. Further we denote that effect of node v_k on node v_h after removing an incident edge to node v_h as $\frac{\partial \tilde{x}_h^\Lambda}{\partial x_k}$. We define the change in effect of v_k on v_h before and after removing an incident edge to v_h as distance function $D(k,h) = 1 - \mathbb{E}(\frac{\partial \tilde{x}_{h,s}^\Lambda}{\partial x_{k,t}} / \frac{\partial x_{h,s}^\Lambda}{\partial x_{k,t}})$ for any entry $1 \le s,t \le n$ of x_h and x_k. Similarly, we define the change in effect of node v_k on v_l as $D(k,l) = 1 - \mathbb{E}(\frac{\partial \tilde{x}_{l,s}^\Lambda}{\partial x_{k,t}} / \frac{\partial x_{l,s}^\Lambda}{\partial x_{k,t}})$ for any entry $1 \le s,t \le n$ of x_l and x_k. Then, $D(k,h) < D(k,l)$.*

Theorem 1 states that the change in influence of a random node v_k on another node v, caused by excluding a target link is higher on the low degree nodes v_l. This suggests that low-degree nodes benefit more by excluding target edges: when all target edges are present, low-degree nodes are more vulnerable to the issues brought by the inclusion of target edges. This statement holds for any GNN model relying on message passing. We provide detailed proofs here [1].

Quantitative Analysis: Average degree change. We further support our claim that low-degree nodes are affected more by providing a quantitative analysis on the relative degree changes before and after excluding the train target links. We analyze four datasets of various sparsity levels, as shown in Tab. 2. For each dataset, we sort its nodes by their degrees and report the average degree change before and after excluding the train targets for each mini-batch epoch. As shown in Fig. 3, for low-degree nodes, the relative change is near 100 %, while for high-degree nodes it is less than 10%.

[1] https://arxiv.org/abs/2306.00899

Algorithm 1 SpotTarget: Leakage Check(G)

Input: An input graph $\mathbf{G}$, edge splits $\mathbf{S}$, an argument $\mathbf{K}$ if validation target 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 needs
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 it 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 $\mathbf{G}_{\text{infer}}$

Proposed Solution: Exclude T_{low}. To achieve the best trade-off between avoiding issues (I1)-(I2) and minimizing the corruption of the graph structure in mini-batch training, SpotTarget excludes T_{low}, the train target edges where at least one incident node has degree lower than a degree threshold δ. Implementation-wise, to ensure the scalability and usability of our proposed solution in large-scale, real-world applications, we implemented it as a subclass of DGL's edge sampler, which is comparable to DGL's original edge sampler and can be readily combined with other DGL functions.

5.2 Test-time Best Practice: Exclude All Test Target Links

As we have discussed, including the test target links in the test message-passing graph causes test data leakage. This may occur inadvertently—for example, when adapting GNNs designed for node classification tasks for the link prediction task by simply stacking a decoder function—or when test target links are explicitly added into the graph to ensure that there is no distribution shift issue. We argue that under no circumstance should the test edges be used as message-passing edges. This would ensure more accurate estimation of GNN's predictive performance.

Proposed Solution. SpotTarget excludes all the test target links from the test message-passing graph. Moreover, it supports automatically checking for data leakage in user-specified data splits and rectifying the issues as needed (Alg. 1). In prior work [11], validation edges are sometimes used in the message-passing graphs to obtain more information, especially for data that is split into training/validation/test sets according to time. Including the validation target edges is typically not seen as data leakage. The decision of whether or not to use the validation edges as message-passing edges depends on the application of interest. SpotTarget requires the user to deliberately define this design choice, and generates the inference graph that complies with the user requirements.

6 EXPERIMENTS

Through our extensive empirical analysis, we aim to address the following research questions:

- **RQ1**: How well does SpotTarget address issues (I1) and (I2) on commonly-used graph benchmarks, which are dense?
- **RQ2**: How well does SpotTarget perform on sparse graphs with very skewed degree distributions?
- **RQ3**: How well does SpotTarget address issues (I1)-(I2) for edges incident to low-degree nodes on popular benchmarks?
- **RQ4**: At test time, how much is the performance of GNN models overestimated due to implicit data leakage (I3)?

Before presenting our results, we describe the experiment setup.

Table 2: Dataset statistics based on the training splits.

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 [25]	332	3,402	10.25	332
E-commerce [24]	346,439	238,818	1.38	768

Table 3: RQ1-Training Issues: Results on dense graphs. Test performance of different training frameworks across GNNs and datasets. SpotTarget has the best overall performance (lowest rank) across all datasets. *OOM = out of GPU memory.

Model	ExcludeNone(Tr)	ExcludeAll	ExcludeRandom	SpotTarget
	Ogbl-Collab (H@50 ↑)			
SAGE	48.57 ± 0.74	45.82 ± 0.41	45.74 ± 1.33	49.00 ± 0.65
MB-GCN	43.03 ± 0.50	37.75 ± 1.42	41.43 ± 2.25	39.58 ± 1.06
GATv2	45.61 ± 0.85	45.71 ± 0.87	45.87 ± 0.64	45.46 ± 0.19
SEAL	61.27 ± 0.28	64.11 ± 0.30	64.40 ± 0.57	64.57 ± 0.30
	Ogbl-Citation2 (MRR ↑)			
SAGE	82.06 ± 0.06	81.47 ± 0.17	82.06 ± 0.13	82.18 ± 0.18
MB-GCN	79.70 ± 0.25	79.06 ± 0.30	80.39 ± 0.15	79.88 ± 0.14
GATv2	OOM	OOM	OOM	OOM
SEAL	86.75 ± 0.20	86.74 ± 0.23	86.61 ± 0.39	86.93 ± 0.55
	USAir (AUC ↑)			
SAGE	95.97 ± 0.17	95.71 ± 0.12	96.42 ± 0.18	96.19 ± 0.53
MB-GCN	94.00 ± 0.14	94.09 ± 0.11	93.98 ± 0.06	94.28 ± 0.15
GATv2	95.05 ± 0.66	95.66 ± 0.24	95.80 ± 0.24	95.87 ± 0.46
SEAL	95.36 ± 0.24	95.94 ± 0.04	95.76 ± 0.24	96.39 ± 0.09
Rank ↓	2.81	3.09	2.45	1.64

Data. We evaluate our framework on four real-world datasets on the link prediction task and give their statistics in Tab. 2. Ogbl-Collab and Ogbl-Citation2 [11] are author collaboration and citation networks. USAir [25] is a network of US airlines. We note that these datasets are relatively dense, with average node degree of 8-20. In real-world applications, the observed data is typically incomplete and sparse, with skewed degree distributions and many low-degree nodes. For this reason, we also consider E-commerce [24], a sparse real-world dataset of queries and related products that are exact matches in Amazon Search.

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 [36]. For E-commerce, we choose to report MRR, Hits@10, and Hits@1, the three most commonly-used evaluation metrics for link prediction [11, 29, 38]. For all evaluation metrics, the higher the value is, the better.

GNN models. We select four GNN models to validate our proposed solutions. SAGE [10], MB-GCN [15] and GATv2 [3] are GAE-based models. MB-GCN [15] is a mini-batch GCN model and at each iteration, only a portion of the entire graph is seen. SEAL [36] is a subgraph-based model that extracts an enclosing subgraph for each target edge and predicts the link likelihood based on the

Table 4: RQ2-Training Issues: Results on the sparse E-commerce dataset. SpotTarget achieves consistently better performance than the baseline across metrics and models. 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

subgraph's embeddings. All GNNs are implemented in DGL [23, 30]. We conduct a hyperparameter tuning and choose the best.

Baselines. For training-time issues (I1, I2), we use ExcludeNone(Tr), ExcludeAll and ExcludeRandom as our baselines. ExcludeNone(Tr) does not exclude any training target links, while ExcludeAll excludes all target links T_{Tr}. Note that ExcludeAll on SEAL is essentially FakeEdge, which excludes all target edges on subgraph-based models. ExcludeRandom randomly excludes target edges during training, and the proportion of excluded targets is the same as our SpotTarget. For test-time issues (I3), our baseline is ExcludeNone(Tst), which uses the test target links in the inference graph. This approach corresponds to the case where data leakage occurs, which should always be avoided in real-world applications.

SpotTarget Variants. At training time, we consider two variants of SpotTarget that differ in the degree threshold δ that they use to exclude target links T_{low} for all datasets, and report the best-performing one: $\delta = 10$ or $\delta = 20$, which corresponds to the average degrees of the dense datasets we used. For the E-Commerce dataset, since 99.5% edges are incident to nodes with degree less than 5, SpotTarget excludes almost all target edges and achieves similar impact as ExcludeAll and ExcludeRandom. At test time, we consider two variants for SpotTarget: ExcludeValTst excludes both validation and test target edges from the test graph, while ExcludeTst only excludes the test target links. As shown in Alg. 1, whether to use ExcludeValTst or ExcludeTst depends on the user's input.

6.1 RQ1-Training Issues: Results on Dense Data

Setup. To evaluate SpotTarget's ability to address training issues (I1) and (I2) on dense graphs, we report the link prediction performance of four GNN models on three popular dense datasets over three trials. We report the recommended metrics for each dataset. For Ogbl-Collab and Ogbl-Citation2, we generate one negative per target edge during training and use the recommended negatives during evaluation. For USAir, we also generate one negative per target edge during training, while during evaluation, we treat all edges that do not appear in the train,test,validation as negative edges. In addition to the performance for each setting, we also report the average rank of our baselines and proposed framework SpotTarget. Our results are summarized in Tab. 3.

Results. SpotTarget achieves the best performance (lowest rank) across datasets and models. On Ogbl-Citation2 and USAir, our method almost achieves the best results across different types of models. This indicates that SpotTarget successfully addresses the train issues (I1, I2) while also avoiding significant corruption of the structure in the mini-batch graphs. Although the original implementation of SEAL uses ExcludeAll, we find that replacing that strategy with SpotTarget further helps improve SEAL's performance.

Moreover, comparing SpotTarget with ExcludeRandom, we can see that SpotTarget consistently gives better performance. This experimentally verifies Theorem 1 and show that specifically excluding the edges incident to low-degree nodes can benefit more.

We also observe that ExcludeAll typically results in slightly lower performance compared to ExcludeNone(Tr). As discussed in Sec. 5.1, this is mainly because excluding all target edges in one mini-batch causes a significant change on graph structure and even isolates some nodes. Thus, GNNs will not learn good node representations.

Observation 1. **(1)** *Across all datasets and models, SpotTarget achieves the best overall rank compared with ExcludeNone(Tr), ExcludeAll and ExcludeRandom. This indicates that it successfully addresses the issues (I1) and (I2).* **(2)** *In many cases (6/11), ExcludeAll leads to performance degradation because of currupting the structure of mini-batch graphs.*

6.2 RQ2-Training Issues: Results on Sparse Data

Setup. In the real-world E-commerce dataset, the graph is incomplete, sparse and full of low-degree nodes. Based on our theoretical analysis in Sec. 5.1, the low-degree nodes suffer more from training issues. To investigate the usefulness of SpotTarget in such settings, we repeat the previous experiments. Note that we do not report the results of ExcludeAll and ExcludeRandom because almost all edge is incident to nodes with degree less than 5, so SpotTarget excludes nearly every target edge. Furthermore, due to the high sparsity, we also do not report the results for SEAL since it is impractical to construct subgraphs for each node. The results are shown in Tab. 4.

Results. On sparse graphs like E-commerce, SpotTarget achieves *14.9* × better performance. Since many real-world graphs are very sparse (e.g. commonsense knowledge graphs and biochemical graphs have an average degree of 2 [6, 21]), SpotTarget can improve the performance of GNNs across numerous high-impact settings.

Observation 2. *SpotTarget achieves 14.9× better performance compared to ExcludeNone across models. This verifies empirically that low-degree nodes suffer more from issues (I1) and (I2), and excluding T_{low} works well especially for datasets with many low-degree nodes.*

6.3 RQ3-Training Issues: Results on Low-degree Nodes

Setup. To quantify how much low-degree nodes in dense datasets suffer from issues (I1) and (I2), we explore the predictive performance for edges adjacent to low-degree nodes. We report the performance of two different edge types: (1) edges that are incident to at least one low-degree node, i.e., $min(d_i, d_j) < \delta$ and; (2) edges that are only incident to low-degree nodes, i.e., $max(d_i, d_j) < \delta$. We report results on Ogbl-Citation2 for SAGE, and compare SpotTarget against three baselines. Results are shown in Tab. 5.

Table 5: RQ3-Training Issues: Results on low-degree nodes. We report MRR of SAGE on Ogbl-Citation2 on target edges incident to at least one low-degree nodes ($min(d_i, d_j)$) or only low-degree nodes ($max(d_i, d_j)$). SpotTarget achieves the best performance.

	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
	ExcludeRandom	76.11 ± 0.12	70.79 ± 0.53	79.41 ± 0.06	72.31 ± 0.04	55.21 ± 0.06	41.48 ± 0.42
	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: RQ4-Test Issue: Leakage quantification. We report the test results of four GNNs over three datasets. Note that ExcludeNone(Tst)'s good performance is due to data leakage; the test edges, never observed in real-world applications, are used during inference. Using test target links should be avoided; our framework, SpotTarget, can automatically check and/or enforce 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	✓	✓	✗

Results. For edges that are incident to low-degree nodes, ExcludeAll, ExcludeRandom and SpotTarget achieve significantly better performance than ExcludeNone(Tr). This corresponds to our theoretical analysis in Sec. 5.1 that highlights low-degree nodes are harmed by training issues (I1) and (I2) more, and excluding train target edges is more beneficial for low-degree nodes. Specifically, comparing ExcludeNone(Tr), ExcludeAll and ExcludeRandom, SpotTarget achieves better performance on various types of edges that are incident to low-degree nodes. This indicates that SpotTarget is better at maintaining the graph structure in mini-batch training.

Observation 3. *Better performance on edges adjacent to low-degree nodes in dense graphs indicates that SpotTarget successfully resolves (I1, I2) on low-degree nodes.*

6.4 RQ4-Test Issues: Leakage Quantification

Setup. Beyond the training issues (I1, I2), we aim to quantify the performance gap introduced by the data leakage at test time (I3). To achieve this, we report results on excluding different types of edges from the inference graph (validation, test edges). Although we are not evaluating in a deployed system, by excluding different types of edges, we are mimicking what would happen in a real application. All GNNs are trained using train edges only. ExcludeValTst excludes all validation and test target links during inference, and ExcludeTst only excludes validation edges. Both ExcludeValTst and ExcludeTst are variants of SpotTarget. ExcludeNone(Tst) keeps all validation and test target links during testing, resulting in data leakage (I3) and should be avoided in practice. The results are shown in Tab. 6.

Results. When validation target links are used as message-passing edges in inference graphs, we observe a slight performance boost, which matches findings in prior work [11]. However, the performance boost due to the inclusion of the test target links is undesired, as it can lead to overestimation of the models' predictive performance. In practice, test links cannot be observed and utilized. Specifically, when test targets are present during inference, SEAL seemingly achieves near-perfect results, which are not indicative of actual performance. SpotTarget successfully resolves issue (I3).

Observation 4. *Due to data leakage (I3), using test edges causes a fake performance boost across multiple datasets, especially for those with time-based splits (e.g., Ogbl-Collab). Increased performance verifies the necessity of SpotTarget, which always excludes the test target links from the inference graphs at test time. Since in real applications, future (test) links are never observed, if the model utilizes information from test target edges, its performance gets overestimated, i.e., a fake performance boost that will not be seen in practice is achieved.*

7 CONCLUSION

In this work, we focused on the pitfalls in link prediction with GNNs and systematically study the issues that arise from including the target links as message passing edges. We are the first to show (both theoretically and empirically) that low-degree nodes suffer more from these issues. Our proposed framework, SpotTarget, strikes the best balance between eliminating the issues from the target links, not significantly corrupting the structure of the mini-batch graphs, and being scalable and easy to use. SpotTarget can help researchers and practitioners adhere to best practices, which are frequently overlooked even by the widely-used GNN frameworks.

ACKNOWLEDGMENTS

We thank Yongyi Yang for providing constructive feedback on our theoretical proof. This material is based upon work supported by the National Science Foundation under IIS 2212143, CAREER Grant No. IIS 1845491, and AWS Cloud Credits for Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties.

ETHICAL DISCUSSION

In our work, the datasets we used are publicly available; we did not collect or release new datasets. During data pre-processing, we strictly followed ethical principles and did not attempt to infer sensitive attributes. As discussed in Sec. 6.3, our approach is able to improve performance for edges adjacent to low-degrees nodes, which can be used to mitigate the potential bias of current GNNs on marginalized nodes (e.g., individuals) that have few connections.

REFERENCES

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

[2] 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).

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

[4] 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).

[5] 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).

[6] 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).

[7] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On power-law relationships of the internet topology. *ACM SIGCOMM computer communication review* 29, 4 (1999), 251–262.

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

[9] Shahriar Golchin and Mihai Surdeanu. 2023. Time travel in llms: Tracing data contamination in large language models. *arXiv preprint arXiv:2308.08493* (2023).

[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] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1857–1867.

[14] 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).

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

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

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

[18] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 177–187.

[19] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of massive data sets*. Cambridge university press.

[20] 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.

[21] 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.

[22] 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.

[23] 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).

[24] 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

[25] 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.

[26] Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. 2023. NLP Evaluation in trouble: On the Need to Measure LLM Data Contamination for each Benchmark. *arXiv preprint arXiv:2310.18018* (2023).

[27] 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.

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

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

[30] 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).

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

[32] 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.

[33] 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.

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

[35] 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.

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

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

[38] 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.

[39] 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.

[40] Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. 2023. Don't Make Your LLM an Evaluation Benchmark Cheater. *arXiv preprint arXiv:2311.01964* (2023).

[41] Yuhang Zhou, Paiheng Xu, Xiaoyu Liu, Bang An, Wei Ai, and Furong Huang. 2023. Explore Spurious Correlations at the Concept Level in Language Models for Text Classification. *arXiv preprint arXiv:2311.08648* (2023).

[42] Jing Zhu, Xiang Song, Vassilis N Ioannidis, Danai Koutra, and Christos Faloutsos. 2023. TouchUp-G: Improving Feature Representation through Graph-Centric Finetuning. *arXiv preprint arXiv:2309.13885* (2023).

[43] 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.