EFLEC: Efficient Feature-LEakage Correction in GNN based Recommendation Systems

Ishaan Kumar*
ishaan.kumar@huawei.com
Huawei Technologies Canada
Montreal, Quebec, Canada

Yaochen Hu*
yaochen.hu@huawei.com
Huawei Technologies Canada
Montreal, Quebec, Canada

Yingxue Zhang
yingxue.zhang@huawei.com
Huawei Technologies Canada
Montreal, Quebec, Canada

ABSTRACT

Graph Convolutional Neural Networks (GNN) based recommender systems are state-of-the-art since they can capture the high order collaborative signals between users and items. However, they suffer from the feature leakage problem since label information determined by edges can be leaked into node embeddings through the GNN aggregation procedure guided by the same set of edges, leading to poor generalization. We propose the accurate removal algorithm to generate the final embedding. For each edge, the embeddings of the two end nodes are evaluated on a graph with that edge removed. We devise an algebraic trick to efficiently compute this procedure without explicitly constructing separate graphs for the LightGCN model. Experiments on four datasets demonstrate that our algorithm can perform better on datasets with sparse interactions, while the training time is significantly reduced.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Collaborative filtering*; • **Computing methodologies** → *Learning to rank*.

KEYWORDS

Feature leakage correction; recommendation systems; graph neural networks

ACM Reference Format:
Ishaan Kumar, Yaochen Hu, and Yingxue Zhang. 2022. EFLEC: Efficient Feature-LEakage Correction in GNN based Recommendation Systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3477495.3531770

1 INTRODUCTION

Recommender systems are ubiquitous in the world overloaded with information. The primary task of a recommender system is to provide relevant items to users based on past user behaviour, such as download, purchase, liking. To improve user satisfaction, collaborative filtering (CF) based methods have been extensively adopted for recommendation systems [5, 9, 10, 13, 14, 16, 17, 19], which assumes that users with similar interactions on a subset of items are more likely to have similar preferences on a new set of items. Latent embeddings of users and items are learned according to historical data to predict the ranking of items for a given user. Neural network models are proposed [5, 9, 16, 17, 19] to capture non-linear similarity relations between users and items to boost the performance.

Recently, Graph Convolutional Neural Networks (GNN) [2, 6, 12] demonstrate extraordinary capacity on learning representation for graph information, and increasing amount of recommender systems are designed and proposed with GNN [1, 8, 22, 24, 27, 30–32]. A bipartite graph can be naturally formulated in recommender systems, where users and items are nodes, and the user-item interactions define the edges. GNN based recommender systems can learn high order collaborative signals and are among the state-of-the-art models. The basic GNN models have been developed and evolved towards two main directions: increasing the quality of representations [23, 24, 28, 29, 32] and improving the efficiency of training and inferencing [15, 25, 26, 31]. Interestingly, [8] proposes a simplified GNN model that yields state-of-the-art performance while is efficient to train and inference. They find that the non-linear transformation and learnable projection matrices are unnecessary for GNN based recommender systems. Due to its excellent performance and efficiency, we derive our results based on the LightGCN model.

Despite that vast progress on GNN based recommender systems, few works have studied the *feature leakage problem*. The *feature leakage problem* occurs in a model or training algorithm if the information such as a duplicated label, a proxy for the label, or the label itself is used as a feature [3]. Moreover, such a feature is not always available during the inferencing phase, resulting in poor generalization. Unfortunately, most GNN based recommender systems suffer from the *feature leakage problem*. Edges are the target training labels, while they also function as the signal to aggregate the embeddings from neighbouring nodes. The training labels (edges) information is partially integrated into the node embeddings after the GNN layers. [33] is one of the earliest works that solves the feature leakage problem in GNN based models for the link prediction problem. Although recommender systems can be regarded as a link prediction task, [33] relies on constructing different subgraphs for different links, which is not scalable for recommender systems. [32] is the most relevant to our work. They remove all the related edges in the labels from the graph in each mini-batch. However, this aggressive approach might leave too much information behind, leading to sub-optimal performance, especially for sparse graphs.

*Both authors contributed equally to this research.

SIGIR '22, July 11–15, 2022, Madrid, Spain

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3531770

In this work, we propose EFLEC, an Efficient Feature LEakage Correction algorithm for GNN based recommender systems. We consider an *accurate removal* approach, which evaluates the embedding of nodes for each edge over a separate graph with that specific edge removed. Compared with [32], this approach can accurately remove the problematical edge during GNN aggregation for its two end nodes without sacrificing potentially valuable edges within the same mini-batch. However, a direct implementation for *accurate removal* is computationally infeasible. We algebraically derive the relation between accurate removal embeddings and the original embeddings and propose a dynamic programming algorithm to efficiently compute the accurate removal embeddings based on the original embeddings (which can be efficiently computed). We conduct extensive experiments to verify the effectiveness of our algorithm.

2 PRELIMINARIES

2.1 Recommender Systems with GNN Encoder

Recommender Systems with Embedding Learning. One common design for recommendation systems learns user and item embeddings $\mathbf{e}_u$ and $\mathbf{e}_i$ for each user u and item i in a common latent space based on their historical interactions. Predominantly, the inner product of these embeddings $\hat{y}_{ui} = \mathbf{e}_u^\intercal \cdot \mathbf{e}_i$ is used to depict similarity between a given user-item pair. During inference, we pick the top-k items with the highest predicted similarity for a given user.

GNN Encoder. LightGCN [8] is the state-of-the-art GNN based model to learn informative user and item embeddings. The main idea is that we can naturally treat the historical interactions between users and items as a graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ is the set of nodes contains user nodes $\mathcal{U}$ and item nodes $\mathcal{I}$, and $\mathcal{E}$ is the set of edges where edge $(u, i) \in \mathcal{E}$ if and only if user u has interacted with item i. Let $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ be the adjacent matrix of the graph $\mathcal{G}$, the normalized adjacent matrix $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the diagonal matrix with $\mathrm{D}_{ii} = \sum_j \mathrm{A}_{ij}$. Without confusion, we abuse the notation $\mathbf{A}$ to represent $\tilde{\mathbf{A}}$ in the remaining paper. Graph convolution operations are iteratively conducted to aggregate the features of neighbour nodes to inject the graph structure information into the final embeddings. Specifically, for each node (user or item) j, the embedding at layer $k+1$ is evaluated as

$$\mathbf{e}_j^{(k+1)} = \mathbf{A}_{j,*}\mathbf{E}^{(k)}, \tag{1}$$

where the row vector $\mathbf{e}_j^{(k+1)}$ denotes the embedding of node j at layer $k+1$, $\mathbf{A}_{j,*}$ denotes the j^{th} row of $\mathbf{A}$ and $\mathbf{E}^{(k)}$ is the matrix containing all user nodes and item nodes embeddings at layer k as row vectors. $\mathbf{e}_j^{(0)}$ is a randomly initialized learnable embedding for node j. The final embedding of a node j with a K layer model are defined as

$$\mathbf{e}_j = \frac{1}{K+1}\sum_{k=0}^{K}\mathbf{e}_j^{(k)} = \frac{1}{K+1}\sum_{k=0}^{K}\mathbf{A}_{j,*}^{k}\mathbf{E}^{(0)}. \tag{2}$$

Model Training. To train a ranking model, we use the Bayesian Personalized Ranking (BPR) [20] loss, which is defined on the set of triples $\mathcal{D} = \{(u, t, n)|u \in \mathcal{U}, t, n \in \mathcal{I}, (u, t) \in \mathcal{E}, (u, n) \notin \mathcal{E}\}$, and $L_{\text{BPR}} = -\sum_{(u,t,n)\in\mathcal{D}} \sigma(\hat{y}_{ut} - \hat{y}_{un})$, where σ is the sigmoid function. In practice, we adopt mini-batch stochastic gradient decent (SGD) method to boost the training efficiency. To evaluate the loss on a mini-batch of triples in $\mathcal{D}$, we only need to evaluate the user and item embeddings that appears in the mini-batch.

2.2 Feature Leakage Problem

A training algorithm suffers from the feature leakage problem if the information such as a duplicated label, a proxy for the label, or the label itself is used as a feature [3]. Feature leakage problem leads to poor generalization since the label information is usually absent during inferencing.

Unfortunately, the proxy of labels appears in the training procedure in the system we introduced in Sec. 2.1. The main reason is that each edge in $\mathcal{E}$ performs dual roles in the training procedure. For each triple $(u, t, n) \in \mathcal{D}$, essentially a label in BPR loss, u and t is determined by the edge $(u, t) \in \mathcal{E}$, so edge (u, t) serves a role to set the training label. On the other hand, edge (u, t) provides a signal for the GNN aggregation in (1). Therefore for each triple (u, t, n), the GNN aggregation will reinforce the similarity between the embeddings e_u and e_t, but not for e_u and e_n. However, this reinforcement never appears in the inferencing phase, leading to degraded generalization.

2.3 Accurate Removal and Sample-and-remove Methods

To alleviate the feature leakage problem, one straightforward idea is to remove the related edge in the message passing procedure. Specifically, for each triple (u, t, n), we evaluate a modified version of the embeddings of node u and t based on the graph with edge (u, t) removed, i.e., $\mathcal{G}_{-(u,t)} = \{\mathcal{V}, \mathcal{E}\backslash\{(u, t)\}\}$. We call this approach *accurate removal* method. However, the accurate removal method is computationally infeasible in practice since for each triple (u, t, n) in a mini-batch, we have to evaluate the embeddings on a new graph $\mathcal{G}_{-(u,t)}$.

[32] proposes a slightly different variant. Instead of generating a new graph for each triple, they construct a single graph for each mini-batch, simply removing all the edges (u, t) in the given mini-batch of triples. We call it the sample-and-remove (*S&R*) method. Although *S&R* is shown to be effective and efficient in solving the feature leakage problem [32], the aggressive removing ignores a substantial amount of information, especially when the original graph is sparse.

Is it possible to use the graph information as an accurate removal method while keeping an acceptable computation complexity as the *S&R* method? The answer is definitely yes.

3 PROPOSED ALGORITHM

Our major goal is to conduct the accurate removal method with similar computation complexity to the original LightGCN model. The main idea is to seek the relation of the node embeddings from the vanilla LightGCN and those after accurate removal method. In the end, we should find a quick transformation algorithm based on the embeddings $\mathbf{E}^{(k)}, \forall k$ generated by vanilla method. Specifically, for each triple (u, t, n), we need to evaluate the final embedding based on the graph $\mathcal{G}_{-(u,t)}$. We use $\hat{}$ over the existing variables to denote the version under the graph $\mathcal{G}_{-(u,t)}$. The problem is to

compute $\hat{\mathbf{e}}_z = \frac{1}{K+1}\sum_{k=0}^{K}\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)}$, for $z \in \{u,t\}$. As for the node n, we directly use the original final embedding $\mathbf{e}_n$ from LightGCN. Essentially, we need to evaluate $\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)}$ for $\forall k$ based on $\mathbf{A}^k_{z,*}\mathbf{E}^{(0)}$ for $\forall k$.

3.1 Analysis

We derive the relation between $\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)}$ and $\mathbf{A}^k_{z,*}\mathbf{E}^{(0)}$. Note that for vanilla version, we have

$$\mathbf{A}^k_{z,*}\mathbf{E}^{(0)} = \mathbf{A}_{z,*}\mathbf{A}^{k-1}\mathbf{E}^{(0)} = \sum_{i\in\mathcal{N}(z)} \mathrm{A}_{z,i}\mathbf{A}^{k-1}_{i,*}\mathbf{E}^{(0)} \tag{3}$$

$$= \sum_{i\in\mathcal{N}(z)} \mathrm{A}_{zi} \sum_{j\in\mathcal{N}(i)} \mathrm{A}_{ij}\mathbf{A}^{k-2}_{j,*}\mathbf{E}^{(0)}, \tag{4}$$

where (4) is achieved by applying (3) twice. And similarly for accurate removal version, we have

$$\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)} = \sum_{i\in\hat{\mathcal{N}}(z)} \hat{\mathrm{A}}_{zi} \sum_{j\in\hat{\mathcal{N}}(i)} \hat{\mathrm{A}}_{ij}\hat{\mathbf{A}}^{k-2}_{j,*}\mathbf{E}^{(0)}. \tag{5}$$

Another fact is the relation between $\mathbf{A}$ and $\hat{\mathbf{A}}$,

$$\hat{\mathrm{A}}_{ij} = \begin{cases} \mathrm{A}_{ij}, & \text{if } i \neq u, j \neq t, \\ |\mathcal{N}(u)|/|\hat{\mathcal{N}}(u)|\mathrm{A}_{ij}, & \text{if } i = u, j \neq t, \\ |\mathcal{N}(t)|/|\hat{\mathcal{N}}(t)|\mathrm{A}_{ij}, & \text{if } i \neq u, j = t, \\ 0, & \text{if} i = u, j = t. \end{cases} \tag{6}$$

Combining (4) (5) and (6) and subtracting a weighted (5) from (4), we have

$$\mathbf{A}^k_{z,*}\mathbf{E}^{(0)} - |\hat{\mathcal{N}}(z)|/|\mathcal{N}(z)|\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)} = \mathrm{A}_{z\bar{z}}\mathbf{A}^{k-1}_{z,*}\mathbf{E}^{(0)} + \Delta^k_z, \tag{7}$$

where

$$\Delta^k_z = \sum_{i\in\hat{\mathcal{N}}(z)} \mathrm{A}_{zi}\Bigg(\underbrace{\sum_{j\in\hat{\mathcal{N}}(i)} \mathrm{A}_{ij}(\mathbf{A}^{k-2}_{j,*} - \hat{\mathbf{A}}^{k-2}_{j,*})}_{P_1} + \mathrm{A}_{iz}\mathbf{A}^{k-2}_{z,*} - \hat{\mathrm{A}_{iz}}\hat{\mathbf{A}}^{k-2}_{z,*}\Bigg)\mathbf{E}^{(0)}, \tag{8}$$

and $\bar{z}$ denotes the other end points on the edge (u,t) from z. To simply the computation, we adopt an approximated version of $\tilde{\Delta}^k_z$ by ignoring P_1 in (8), and eventually get

$$\tilde{\Delta}^k_z = \mathbf{A}^{k-2}_{z,*}\mathbf{E}^{(0)} \cdot \sum_{i\in\hat{\mathcal{N}}(z)} \mathrm{A}^2_{zi} - \hat{\mathbf{A}}^{k-2}_{z,*}\mathbf{E}^{(0)} \cdot |\mathcal{N}(z)|/|\mathcal{N}(z)| \sum_{i\in\hat{\mathcal{N}}(z)} \hat{\mathrm{A}}^2_{zi}. \tag{9}$$

We can verify that when $k = 0, 1$, $\Delta^k_z = 0$, so we define $\tilde{\Delta}^k_z = 0$ for $k = 0, 1$. By (7) and (9), we get a way to represent $\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)}$ by $\mathbf{A}^k_{z,*}\mathbf{E}^{(0)}$ and $\hat{\mathbf{A}}^{k-2}_{z,*}\mathbf{E}^{(0)}$, we are ready to build an efficient algorithm to compute $\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)}$ for $\forall k$ via dynamic programming.

Due to the approximation in (9), the results from our algorithm are equivalent to the accurate removal method only under the condition that the number of the model layers $K \leq 2$. For more layers, we could further reduce the error by expanding (4) and (5) to more layers and derive the relations with a similar procedure. However, more layers will introduce more computation complexity. We adopt the current design to strike a balance between efficiency and accuracy.

Algorithm 1: EFLEC for a K-layer LightGCN model.

Input : The target node z, the triple (u,t,n), the original embeddings from LightGCN $\mathbf{A}^k\mathbf{E}^{(0)}$ for $k = 0, 1, \ldots, K$, normalized adjacent matrix $\mathbf{A}$.
Output : Leakage corrected embedding $\hat{\mathbf{e}}_z$.

$T \leftarrow \{\mathbf{e}^{(0)}_z\}$;
for $k \leftarrow 1$ ***to*** K **do**
  Compute $\tilde{\Delta}^k_z$ according to (9);
  According to (7), compute $\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)} = |\mathcal{N}(z)|/|\hat{\mathcal{N}}(z)|\left(\mathbf{A}^k_{z,*}\mathbf{E}^{(0)} - \left(\mathrm{A}_{z\bar{z}}\mathbf{A}^{k-1}_{z,*}\mathbf{E}^{(0)} + \tilde{\Delta}^k_z\right)\right)$;
  $T \leftarrow T \cup \{\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)}\}$;
Compute $\hat{\mathbf{e}}_z$ by taking the mean of all the elements in T;
return $\hat{\boldsymbol{e}}_z$;

Table 1: Dataset statistics.

Dataset	#User	#Items	#Interactions	Density
Instant	63,884	10,664	174,527	2e-4
Instrument	54,272	33,030	161,105	9.8e-5
Yelp	31,668	38,048	1,561,406	1.3e-3
Gowalla	29,858	40,981	1,027,370	8.4e-4

3.2 Main Algorithm

Algorithm 1 depicts the procedure to efficiently evaluate the embeddings for $z \in \{u,t\}$ in each triple (u,t,n) . We can repeatedly apply the same procedure efficiently for all the triples in a mini-batch since they can reuse the same input. Although (9) is dependent on $\hat{\mathbf{A}}^{k-2}_{z,*}\mathbf{E}^{(0)}$, we should already have it in earlier iteration at the time of evaluating $\hat{\mathbf{A}}^k_{z,*}\mathbf{E}^{(0)}$.

4 EXPERIMENT

4.1 Experimental Settings

4.1.1 Dataset. We use datasets with two levels of graph density. Statistics for all datasets are reported in Table 1. To demonstrate the effectiveness of our technique on datasets with low average node degree, we use Amazon Instant Video (Instant) and Amazon Musical Instrument datasets (Instrument) [7, 18][1]. We filter out interactions with a rating of less than four and follow a 2-core setting for these datasets, i.e. we only retain user and item nodes with at least two interactions. Under this setting, we can have at least one training and one testing interaction for node. We split the interactions into train/valid/test sets with a 0.7/0.1/0.2 split ratio. We ensure that our train and test split have at least one interaction for each node.

To test the applicability of our technique on the widely used 10-core setting in literature, we use Yelp2018 (Yelp) and Gowalla datasets provided by authors of LightGCN[2]. As the authors do not provide a validation set, we further split the training data into train/valid sets with a 0.9/0.1 split ratio.

[1] http://jmcauley.ucsd.edu/data/amazon/links.html
[2] https://github.com/kuandeng/LightGCN/tree/master/Data

Table 2: Mean results of recall@20, nDCG@20, and time per epoch (T) in seconds. Bold represent the best and underline represents the second best. Vanila is not considered in the ranking for time.

	Method	Instant			Instrument			Yelp			Gowalla		
		Recall	nDCG	T(s)	Recall	nDCG	T(s)	Recall	nDCG	T(s)	Recall	nDCG	T(s)
2 Layers	Vanilla	0.1698	0.0805	2.96	0.0392	0.0187	2.75	0.0577	0.0467	110.13	0.1623	0.1375	82.65
	DropEdge	0.1656	0.0776	4.43	0.0465	0.0216	4.57	0.0581	**0.0469**	259.93	0.1622	0.1375	130.01
	S&R	**0.2202**	**0.1047**	5.23	**0.0541**	**0.0257**	4.62	0.0576	0.0466	444.94	0.1628	**0.1379**	259.49
	EFLEC	**0.2207**	**0.1029**	**3.11**	**0.0546**	**0.0260**	**3.06**	**0.0583**	**0.0469**	**122.75**	0.1630	**0.1382**	**72.88**
3 Layers	Vanilla	0.1776	0.0874	4.52	0.0471	0.0216	3.55	**0.0604**	**0.0489**	136.23	0.1677	0.1414	67.15
	DropEdge	0.1806	0.0825	**3.65**	0.0521	0.0241	**3.70**	**0.0603**	**0.0487**	219.73	**0.1690**	**0.1422**	105.28
	S&R	**0.2160**	**0.1059**	5.72	**0.0574**	**0.0270**	5.18	0.0600	0.0485	465.03	**0.1687**	**0.1420**	190.42
	EFLEC	**0.2155**	**0.1046**	4.56	**0.0573**	**0.0271**	4.15	0.0602	0.0485	**145.11**	**0.1689**	**0.1422**	**71.06**

Table 3: Mini-batch size studies on Instant dataset. R=Recall, N=nDCG, FB=Full-batch.

Batch size	Method	R@20	R@10	N@20	N@10
FB/4	Vanilla	0.1546	0.1132	0.0729	0.0624
	S&R	0.2047	0.1513	0.0963	0.0827
	EFLEC	0.2126	0.1529	0.0963	0.0811
FB/2	Vanilla	0.1553	0.1104	0.0720	0.0606
	S&R	0.1962	0.1425	0.0930	0.0793
	EFLEC	0.2028	0.1448	0.0948	0.0799
FB	Vanilla	0.1437	0.1006	0.0652	0.0542
	S&R	0.1116	0.0740	0.0475	0.0379
	EFLEC	0.1941	0.1351	0.0871	0.0720

4.1.2 Hyper-parameters. We follow the exact setting as LightGCN. Embedding size is set to 64, all the parameters are initialized with Xavier initialization [4] and we use Adam [11] optimizer. We use symmetric normalization without self-edges.

For all datasets we use 1e-3 learning rate and weight decay factor $\lambda = 1e-4$. For Instant and Instrument, we use a mini-batch size of 2048. For Yelp and Gowalla, we use a mini-batch size of 1024. We run experiments for 500 epochs and use the checkpoint with the best validation performance for computing test results.

4.1.3 Baselines. We compare our method to following baselines:

- Vanilla. LigthGCN model without any modification.
- DropEdge [21]. Originally proposed for node classification, this method randomly drops edges from the training graph at the beginning of each training epoch. After edge dropping, the adjacency matrix is re-normalized. We set the dropout probability to 0.5.
- Sample-and-remove (*S&R*). All the edges that appear in the mini-batch of triples are removed from the graph before executing the mini-batch. After removing the edges, the modified training adjacency matrix is re-normalized.

4.2 Main Results

Table 2 compares baseline methods with EFLEC on all the datasets for two-layer and three-layer LightGCN models and presents the mean of 5 trials. We can observe that all the methods addressing feature leakage problems demonstrate an equal or better performance than the vanilla counterpart. For Instant and Instrument, the data sets with lower average node degrees, we observe that *S&R* and EFLEC significantly improve over Vanilla. Specifically, a two-layer model achieves 30%/39% gain with Recall@20 metric. For Yelp and Gowalla, *S&R* and EFLEC get only marginal or no improvement over Vanilla. The high average node degrees make the feature leakage problem less severe since the proxy label information occupies much less weight in the final embeddings. Overall, our EFLEC algorithm achieves a slight but stable improvement over *S&R* on the two-layer models since it fully utilizes the graph information. On the three-layer models, their performance is similar to each other.

All the leakage correction algorithms run slower than the vanilla. *S&R* is the most time-consuming due to the time consuming procedure of reconstructing the adjacent matrix and normalizing it on every mini-batch. Our EFLEC is much more efficient than *S&R* since it mainly relies on the original node embedding from LightGCN, and it does not need to reconstruct the adjacent matrix. DropEdge is efficient, but its performance on recall is not as good as *S&R* and EFLEC, especially on Instant and Instrument data sets.

4.3 Extended Studies

Although *S&R* achieves a slightly worse but comparable model performance against our EFLEC under the current setting, we argue that EFLEC has more advantage when training the model under larger mini-batch sizes. We train models with mini-batch of size {Full-batch, Full-batch/2, Full-batch/4}. Other hyper-parameters are kept the same as our main results. As observed in Table 3, the performance of the *S&R* method is halved in the extreme Full-batch setting. Whereas EFLEC is less affected as mini-batch size increases since EFLEC always utilizes the complete graph information while the aggressive *S&R* method drops too much information under large mini-batch sizes.

5 CONCLUSION

This work studies the feature leakage problem on GNN based recommendation systems. We propose EFLEC, an efficient algorithm that corrects the feature leakage problem. Empirical results demonstrate that our algorithm can improve the performance on sparse

datasets while the computation time is close to the vanilla algorithm without correction. Although we developed our technique for LightGCN, the algebraic idea can be extended to a wide variety of GNN based models, and we would like to discuss it in future works.

REFERENCES

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. In *Proc. Int. Conf. Learning Representations*.

[3] Soumen Chakrabarti, Earl Cox, Eibe Frank, Ralf Hartmut Güting, Jiawei Han, Xia Jiang, Micheline Kamber, Sam S Lightstone, Thomas P Nadeau, Richard E Neapolitan, et al. 2008. *Data mining: know it all.* Morgan Kaufmann.

[4] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 249–256.

[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proc. Int. Joint Conf. Artificial Intelligence*.

[6] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, Vol. 2017-Decem. 1025–1035. arXiv:1706.02216

[7] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proceedings of the 25th International Conference on World Wide Web* (Montréal, Québec, Canada) *(WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 507–517. https://doi.org/10.1145/2872427.2883037

[8] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[10] Thomas Hofmann. 2004. Latent semantic models for collaborative filtering. *ACM Trans. Information System* (2004).

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. Int. Conf. Learning Representations*.

[13] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*.

[14] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* (2009).

[15] Chong Li, Kunyang Jia, Dan Shen, C.J. Richard Shi, and Hongxia Yang. 2019. Hierarchical Representation Learning for Bipartite Graphs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 2873–2879. https://doi.org/10.24963/ijcai.2019/398

[16] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*.

[17] Bin Liu, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang. 2019. Feature Generation by Convolutional Neural Network for Click-Through Rate Prediction. In *Proc. World Wide Web Conference*.

[18] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Santiago, Chile) *(SIGIR '15)*. Association for Computing Machinery, New York, NY, USA, 43–52. https://doi.org/10.1145/2766462.2767755

[19] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-Based Neural Networks for User Response Prediction over Multi-Field Categorical Data. *ACM Trans. Inf. Syst.* 37, 1, Article 5 (Oct. 2018), 35 pages. https://doi.org/10.1145/3233770

[20] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proc. Conf. Uncertainty in Artificial Intelligence*.

[21] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hkx1qkrKPr

[22] Jianing Sun, Wei Guo, Dengcheng Zhang, Yingxue Zhang, Florence Regol, Yaochen Hu, Huifeng Guo, Ruiming Tang, Han Yuan, Xiuqiang He, et al. 2020. A framework for recommending accurate and diverse items using bayesian graph convolutional neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2030–2039.

[23] Jianing Sun, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, Xiuqiang He, Chen Ma, and Mark Coates. 2020. Neighbor Interaction Aware Graph Convolution Networks for Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 1289–1298. https://doi.org/10.1145/3397271.3401123

[24] Jianing Sun, Yingxue Zhang, Chen Ma, Mark Coates, Huifeng Guo, Ruiming Tang, and Xiuqiang He. 2019. Multi-graph convolution collaborative filtering. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1306–1311.

[25] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to Hash with Graph Neural Networks for Recommender Systems. In *Proceedings of The Web Conference 2020 (WWW '20)*. Association for Computing Machinery, New York, NY, USA, 1988–1998. https://doi.org/10.1145/3366423.3380266

[26] Haoyu Wang, Defu Lian, and Yong Ge. 2019. Binarized Collaborative Filtering with Distilling Graph Convolutional Network. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 4802–4808. https://doi.org/10.24963/ijcai.2019/667

[27] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat Seng Chua. 2019. Neural graph collaborative filtering. *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2019), 165–174.

[28] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled Graph Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 1001–1010. https://doi.org/10.1145/3397271.3401137

[29] Xiao Wang, Ruijia Wang, Chuan Shi, Guojie Song, and Qingyong Li. 2020. Multi-component graph convolutional collaborative filtering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 6267–6274.

[30] Shiwen Wu, Wentao Zhang, Fei Sun, and Bin Cui. 2020. Graph Neural Networks in Recommender Systems: A Survey. 37, 4 (2020). http://arxiv.org/abs/2011.02260

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

[32] Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. 2019. STAR-GCN: Stacked and reconstructed graph convolutional networks for recommender systems. *IJCAI International Joint Conference on Artificial Intelligence* 2019-Augus (2019), 4264–4270. https://doi.org/10.24963/ijcai.2019/592

[33] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/53f0d7c537d99b3824f0f99d62ea2428-Paper.pdf