LtrGCN: Large-Scale Graph Convolutional Networks-Based Learning to Rank for Web Search

Yuchen Li[1], Haoyi Xiong[2(✉)], Linghe Kong[1(✉)], Shuaiqiang Wang[2], Zeyi Sun[3], Hongyang Chen[3], Guihai Chen[1], and Dawei Yin[2]

[1] Shanghai Jiao Tong University, Shanghai, China
{yuchenli,linghe.kong}@sjtu.edu.cn, gchen@cs.sjtu.edu.cn
[2] Baidu Inc., Beijing, China
haoyi.xiong.fr@ieee.org, yindawei@acm.org
[3] Zhejiang Lab, Hangzhou, China
sunzeyi@zhejianglab.com, dr.h.chen@ieee.org

Abstract. While traditional Learning to Rank (LTR) models use query-webpage pairs to perform regression tasks to predict the ranking scores, they usually fail to capture the structure of interactions between queries and webpages over an extremely large bipartite graph. In recent years, Graph Convolutional Neural Networks (GCNs) have demonstrated their unique advantages in link prediction over bipartite graphs and have been successfully used for user-item recommendations. However, it is still difficult to scale-up GCNs for web search, due to the (1) extreme sparsity of links in query-webpage bipartite graphs caused by the expense of ranking scores annotation and (2) imbalance between queries (billions) and webpages (trillions) for web-scale search as well as the imbalance in annotations. In this work, we introduce the **Q**-subgraph and **W**-subgraph to represent every query and webpage with the structure of interaction preserved, and then propose **LtrGCN**—an LTR pipeline that samples **Q**-subgraphs and **W**-subgraphs from all query-webpage pairs, learns to extract features from **Q**-subgraphs and **W**-subgraphs, and predict ranking scores in an end-to-end manner. We carried out extensive experiments to evaluate **LtrGCN** using two real-world datasets and online experiments based on the A/B test at a large-scale search engine. The offline results show that **LtrGCN** could achieve Δ NDCG$_5$ = 2.89%–3.97% compared to baselines. We deploy **LtrGCN** with realistic traffic at a large-scale search engine, where we can still observe significant improvement. **LtrGCN** performs consistently in both offline and online experiments.

This work was supported in part by National Key R&D Program of China (No. 2021ZD0110303), NSFC grant 62141220, 61972253, U1908212, 62172276, 61972254, the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning, Shanghai Science and Technology Development Funds 23YF1420500, Open Research Projects of Zhejiang Lab No. 2022NL0AB01.

G. De Francisci Morales et al. (Eds.): ECML PKDD 2023, LNAI 14174, pp. 635–651, 2023.
https://doi.org/10.1007/978-3-031-43427-3_38

Keywords: Learning to Rank · Graph Convolutional Networks · Web Search

1 Introduction

Large-scale Learning to Rank (LTR) is a central part of real-world information retrieval problems, such as web search and content recommendations. Given a query of web search, the search engine first retrieves relevant webpages from the database and sorts the webpages by the ranking scores predicted by LTR models. While traditional LTR models transform ranking into regression problems of various types, they usually fail to capture the structural information over the interactions between billions of queries and trillions of webpages. These interactions indeed characterize how all these queries and webpages connect each other in a large bipartite graph of the web. To provide a better search experience, it is inevitable to incorporate such structural information in LTR.

On the other hand, in recent years, Graph Neural Networks, such as Graph Convolutional Neural Networks (GCN) [18], have been used for user-item recommendations and demonstrated their unique advantages in modeling the problem as link prediction over bipartite graphs [10]. Similar to LTR based on query-webpage pairs, the user-item recommendation also needs to rank items (e.g., products or contents) subject to the given profile of every user. However, it is difficult to directly adopt GCNs for LTR tasks at web-scale, due to the sparsity and imbalanced issues as follows. First of all, as shown in Fig. 1, *links are extremely sparse in the query-webpage bipartite graph* extracted from LTR training datasets, as labeling query-webpage pairs by professional annotators is expensive and time-consuming (difficult to scale-up). Furthermore, *from the webpages' perspectives, edges between queries and webpages are severely imbalanced*—it is quite common to link a query to dozens of webpages with both high and low relevant scores, while a webpage hardly links to any queries, especially to the queries that the webpage is less relevant or low-ranked, in the annotations. Apparently, either sparsity or imbalance issue would significantly downgrade the performance of GCN models [29]. Therefore, to tackle the above two challenges, there needs a non-trivial extension on the GCN-based model for handling LTR at web-scale.

In this work, we propose **LtrGCN** to tackle the above two issues and adopt GCNs for LTR tasks in a pipeline as follows. Given all query-webpage pairs in the LTR training dataset, **LtrGCN** first leverages two advanced sampling strategies to generate the **Q**-subgraph and **W**-subgraph for every query and webpage. Then, **LtrGCN** leverages GCNs to extract feature vectors from the **Q**-subgraph and **W**-subgraph as the representation of the query-webpage pair for ranking score prediction. The feature extraction and ranking score prediction are optimized in an end-to-end manner, so as to enable discriminative feature extraction while preserving structural information in the bipartite graph. As sparsity and imbalance issues are addressed, **LtrGCN** can work with the training datasets, where it is sufficient that only a small proportion of query-webpage pairs are labeled by experts. Furthermore, we conduct extensive experiments to evaluate

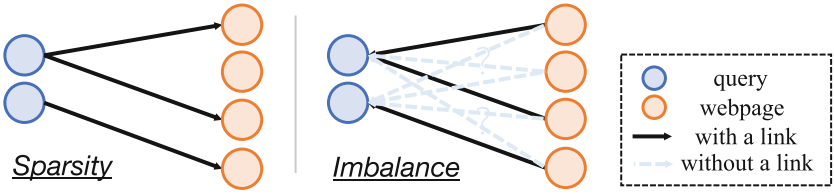


Fig. 1. Sparsity and Imbalance Issues in the Query-Webpage Bipartite Graph.

LtrGCN using two real-world datasets (offline), and launch online experiments based on an A/B test at a large-scale search engine. The offline results show that **LtrGCN** could achieve the best performance on both datasets compared to baselines. As for the online evaluation, we deploy **LtrGCN** with realistic traffic at a large-scale search engine, where we still observe significant improvement. **LtrGCN** performs consistently in both offline and online experiments. Our main contributions are summarized as follows:

- We study the problem of the extreme sparsity of links in query-webpage bipartite graphs caused by the expense of ranking score annotation and the imbalance between queries and webpages for web-scale search. To the best of our knowledge, this work is the first to investigate sparsity and imbalance in query-webpage bipartite graphs for large-scale industrial LTR tasks.
- We propose **LtrGCN** consisting of three steps: (1) *Q-subgraph Generation via Self-tuned Labeling* that annotates all unlabeled query-webpage pairs and assigns every query webpages with ranking scores to generate **Q**-subgraphs, (2) *W-subgraph Generation via Negative Sampling* that find irrelevant queries for every webpage to construct **W**-subgraphs, (3) *Learning to Rank based on GCN with Q-subgraphs and W-subgraphs* that learns the representations of query-webpage pairs from **Q**-subgraphs and **W**-subgraphs and predicts ranking scores in an end-to-end manner.
- We carry out extensive offline experiments on a public LTR dataset and a real-world dataset collected from a large-scale search engine. We also deploy **LtrGCN** at the search engine and implement a series of online evaluations. The experiment results show that, compared to the state-of-the-art in webpage ranking, **LtrGCN** could achieve the best performance on both offline datasets. Furthermore, **LtrGCN** obtains significant improvements in online evaluations under fair comparisons.

2 Methodology

2.1 Task Formulation

Given a set of search queries $\mathcal{Q} = \{q_1, q_2, \dots\}$ and all archived webpages $\mathcal{D} = \{d_1, d_2, \dots\}$, for each query $q_i \in \mathcal{Q}$, the search engine could retrieve a

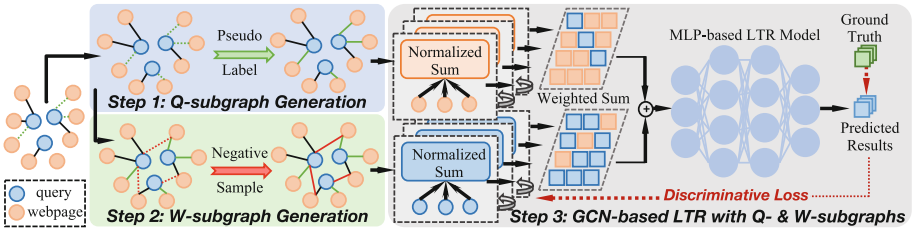


Fig. 2. The framework of **LtrGCN** consisting of three steps: (1) ***Q**-subgraph Generation via Self-tuned Labeling*, (2) ***W**-subgraph Generation via Negative Sampling*, and (3) *GCN-based LTR with **Q**-subgraphs and **W**-subgraphs.*

set of relevant webpages denoted as $D_i = \{d_j^i\}_{j=1}^{|D_i|} \subset \mathcal{D}$. Through professional labeling, a set of ranking scores $\boldsymbol{y}_i = \{y_j^i\}_{j=1}^{|D_i|}$ for q_i is established to characterize the relevance of the webpage $d_j^i \in D_i$ to the search query q_i. In this paper, we follow the settings in [26] and scale the relevant score from 0 to 4 to represent levels of relevance (i.e., {**bad-0, fair-1, good-2, excellent-3, perfect-4**}). We denote a set of query-webpage pairs with ranking score annotations as triples $\mathcal{S} = \{(q_1, D_1, \boldsymbol{y}_1),\ (q_2, D_2, \boldsymbol{y}_2), (q_3, D_3, \boldsymbol{y}_3), \dots\}$. We aim to learn an LTR scoring function $f : \mathcal{Q} \times \mathcal{D} \rightarrow [0, 4]$, which could be approximated through minimizing the following ranking loss:

$$\mathcal{L} = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \left(\frac{1}{|D_i|} \sum_{j=1}^{|D_i|} \ell(\boldsymbol{y}_j^i, f(q^i, d_j^i)) \right), \tag{1}$$

where ℓ represents the loss of the ranking prediction for query q_i with returned webpage d_j^i against the ground truth label $\boldsymbol{y}_j^i$. Note that, **LtrGCN** is flexible with standard loss functions (i.e., pointwise, pairwise, and listwise). As annotators can barely label a small number of query-webpage pairs due to the limited budgets, the key problem of LTR is thus to incorporate the unlabeled query-webpage pairs denoted as set $\mathcal{S}' = \{(q_1', D_1'),\ (q_2', D_2'), \dots\}$.

2.2 Overall Framework of LtrGCN

As illustrated in Fig. 2, **LtrGCN** consists of three steps: (1) *Q-subgraph Generation via Self-tuned Labeling*, (2) *W-subgraph Generation via Negative Sampling*, and (3) *Learning to Rank based on GCN with Q-subgraphs and W-subgraphs*. Specifically, in Step (1), **LtrGCN** first annotates all unlabeled query-webpage pairs with *pseudo* ranking scores and then assigns every query webpages with high ranking scores and also webpages with low scores to generate **Q**-subgraphs from the training set. Then, in Step (2), **LtrGCN** proposes a *negative sampling strategy* to find irrelevant queries for every webpage to construct **W**-subgraphs. Eventually, in Step (3), given **Q**-subgraphs and **W**-subgraphs for every high-ranked query-webpage pair, **LtrGCN** learns the representations of

query-webpage pairs using a Light Graph Convolution Network (LightGCN) [12] and enables LTR in an end-to-end manner.

2.3 Q-subgraph Generation via Self-tuned Labeling

As mentioned above, to leverage GCN for ranking, there needs to feed the model with **Q**-subgraphs and **W**-subgraphs. Given query-webpage pairs that are sparsely annotated with ranking scores in the training set, **LtrGCN** adopts a labeling approach [23] that first annotates every unlabeled query-webpage pair with a *pseudo* ranking score and then assigns every query webpages with high ranking scores and also webpages with low scores, so as to generate **Q**-subgraphs from the training set at full-scale. Thus, there needs the learning to predict *pseudo* ranking scores with labeled/unlabeled samples in the training set.

LtrGCN first gets every possible query-webpage pair from query and webpage datasets as (q_i, d_i^j) for $\forall q_i \in \mathcal{Q}$ and $\forall d_i^j \in D_i \subset \mathcal{D}$. For each query-webpage pair (q_i, d_i^j), **LtrGCN** further extracts an m-dimensional feature vector $\boldsymbol{x}_{i,j}$ representing the features of the j^{th} webpage under the i^{th} query. Then, the labeled and unlabeled sets of feature vectors can be presented as $\mathcal{M} = \{(\boldsymbol{x}_{i,j}, \boldsymbol{y}_j^i) | \forall (q_i, D_i, \boldsymbol{y}) \in \mathcal{S} \text{ and } \forall d_j^i \in D_i\}$ and $\mathcal{M}^{'} = \{\boldsymbol{x}_{i,j} | \forall (q_i, D_i) \in \mathcal{S}^{'}\}$. Given the labeled feature set $\mathcal{M}$ and the unlabeled feature set $\mathcal{M}^{'}$, **LtrGCN** further takes a two-step strategy to accomplish the pseudo-label generation via multi-loss learning as follows.

First, **LtrGCN** trains an LTR model with the listwise loss function as:

$$\mathcal{L}_{List} = -\frac{1}{|\mathcal{M}|} \sum_{i=1}^{|\mathcal{M}|} \left(\frac{1}{|D_i|} \sum_{j=1}^{|D_i|} \text{softmax}(\boldsymbol{y}_j^i) \times \log \left(\text{softmax}\, f(q^i, d_j^i) \right) \right). \quad (2)$$

The listwise-based LTR model is denoted as Rank^{List}. **LtrGCN** trains Rank^{List} using both $\mathcal{M}$ and $\mathcal{M}^{'}$ through self-training, where Rank^{List} is first trained using $\mathcal{M}$ through supervised learning. Then, Rank^{List} predicts the ranking score for each feature vector in $\mathcal{M}^{'}$ and pseudo-labels the feature vector with the prediction result. After that, **LtrGCN** combines $\mathcal{M}$ with pseudo-labeled data $\mathcal{M}^P$ and retrains Rank^{List} using the *combined data* $\mathcal{M}^C$.

Given the $\mathcal{M}^C$, $\mathcal{M}$ and $\mathcal{M}^{'}$, **LtrGCN** (1) trains an LTR model with the pointwise loss function as:

$$\mathcal{L}_{Point} = \frac{1}{|\mathcal{M}^C|} \sum_{i=1}^{|\mathcal{M}^C|} \left(\frac{1}{|D_i|} \sum_{j=1}^{|D_i|} |f(q^i, d_j^i) - \boldsymbol{y}_j^i|^2 \right). \quad (3)$$

The pointwise-based LTR model is denoted as Rank^{Point}. **LtrGCN** first trains Rank^{Point} with $\mathcal{M}^C$ and predicts pseudo-labels for each feature vector in $\mathcal{M}^{'}$ using trained Rank^{Point}. Then, **LtrGCN** updates $\mathcal{M}^P$ with the prediction results of Rank^{Point} and combines $\mathcal{M}$ with $\mathcal{M}^P$ to conduct $\mathcal{M}^C$. **LtrGCN** further retrains Rank^{List} using $\mathcal{M}^C$ and predicts ranking scores for each feature

vector in $\mathcal{M}^{'}$ using trained Rank^{List}. Finally, **LtrGCN** updates $\mathcal{M}^{P}$ with the prediction results of R^{List} and combines $\mathcal{M}$ with $\mathcal{M}^{P}$ to obtain $\mathcal{M}^{C}$. **LtrGCN** repeats the above steps with $\mathcal{T}$ rounds and returns $\mathcal{M}^{C}$.

With *pseudo* ranking scores predicted for all unlabeled samples, **LtrGCN** builds a **Q**-subgraph for every query with the (*pseudo*) ranking scores greater than **2 (good)**. Specifically, to build the **Q**-subgraph, **LtrGCN** randomly picks up a webpage that the query-webpage is with the ranking score lower than **1 (fair)**, and forms the three items (i.e., *the query, a highly-ranked webpage of the query, a low-ranked webpage of the query*) into a **Q**-subgraph.

2.4 W-subgraph Generation via Negative Sampling

Though **Q**-subgraph Generation step could generate ranking scores for every query-webpage pair in the training dataset, it is still difficult to construct **W**-subgraphs using predicted scores at full-scale. While every query connects to the webpages with high/low *pseudo* ranking scores, a webpage usually only connects to one or very limited highly-relevant queries and the number of webpages is much larger than that of effective queries from a webpages' perspective. Thus, there needs to find irrelevant queries for every webpage. To build **W**-subgraphs for a webpage, **LtrGCN** leverages a *negative sampling strategy*. Given a webpage, **LtrGCN** retrieves all query-webpage pairs, builds a **W**-subgraph for every query-webpage with the ranking scores higher than **2** (fair). Specifically, **LtrGCN** randomly picks up a query that does not connect to the webpage as the irrelevant query, then forms the three (i.e., *the webpage, a query where the webpage is highly ranked, and an irrelevant query*) into a **W**-subgraph. Specifically, for a query q^i, **LtrGCN** randomly chooses the webpage from the other query to conduct the negative samples and assigns the relevant score as 0 or 1 to represent poor relevance. Through this negative sampling method, **LtrGCN** could build **W**-subgraphs for a webpage.

2.5 GCN-Based LTR with Q-subgraphs and W-subgraphs

Given **Q**-subgraphs and **W**-subgraphs for every high-ranked query-webpage pair, in this step, **LtrGCN** learns the representations of query-webpage pairs with a GCN and enables learning to rank (LTR) in an end-to-end manner.

In the initial step, given the **Q**-subgraph and **W**-subgraph, **LtrGCN** extracts the feature vector of each query and webpage. Specifically, the feature of query q_i and webpage d_j^i is denoted as $\boldsymbol{z}_{q^i}^{(n=0)}$ and $\boldsymbol{z}_{d_j^i}^{(n=0)}$, where n indicates the feature output from the n^{th} GCN layer. Next, the GCN-based encoder utilizes the query-webpage interaction graph to propagate the representations as:

$$\begin{aligned}
\boldsymbol{z}_{q^i}^{(n+1)} &= \sum_{d_j^i \in \mathcal{N}_{q^i}} \frac{1}{\sqrt{\left|\mathcal{N}_{q^i}\right|}\sqrt{\left|\mathcal{N}_{d_{j,i}}\right|}} \boldsymbol{z}_{d_j^i}^{(n)}, \\
\boldsymbol{z}_{d_j^i}^{(n+1)} &= \sum_{q^i \in \mathcal{N}_{d_j^i}} \frac{1}{\sqrt{\left|\mathcal{N}_{q^i}\right|}\sqrt{\left|\mathcal{N}_{d_{j,i}}\right|}} \boldsymbol{z}_{q^i}^{(n)},
\end{aligned} \tag{4}$$

where $\mathcal{N}_{q^i}$ and $\mathcal{N}_{d_{j,i}}$ represent the set of webpages that are relevant to query q^i and the set of queries that are relevant to webpage d^i_j, respectively. Moreover, $\frac{1}{\sqrt{|\mathcal{N}_{q^i}|}\sqrt{|\mathcal{N}_{d_{j,i}}|}}$ is the normalization term used to prevent the scale of representations from increasing as a result of graph convolution operations. After N layers graph convolution operations, **LtrGCN** combines the representations generated from each layer to conduct the final representation of query q^i and webpage d^i_j as follows:

$$\boldsymbol{z}_{q^i} = \sum_{n=0}^{N} \beta_n \boldsymbol{z}^{(n)}_{q^i};\ \boldsymbol{z}_{d^i_j} = \sum_{n=0}^{N} \beta_n \boldsymbol{z}^{(n)}_{d^i_j}, \tag{5}$$

where $\beta_n \in [0, 1]$ is a hyper-parameter to balance the weight of each layer representation. Then, **LtrGCN** combines $\boldsymbol{z}_{q^i}$ and $\boldsymbol{z}_{d^i_j}$ to conduct the learned query-webpage pair representation as $\boldsymbol{z}_{i,j}$.

Given the learned vector $\boldsymbol{z}_{i,j}$, **LtrGCN** adopts a Multi-Layer Perception (MLP)-based model with a fully-connected layer to calculate the predicted score $\boldsymbol{s}_{i,j}$. The whole process can be formulated as: $\boldsymbol{s}_{i,j} = f_\theta(\boldsymbol{z}_{i,j})$, where θ is the set of discriminative parameters. Against the ground truth, **LtrGCN** leverages the discriminative loss function, which is defined as:

$$\mathcal{L}_{Disc} = \frac{1}{|\mathcal{Q}|} \sum_{i=1}^{|\mathcal{Q}|} \left(\frac{1}{|D_i|} \sum_{j=1}^{|D_i|} \ell_{LTR} \left(\boldsymbol{y}^i_j, f_\theta(\boldsymbol{z}_{i,j}) \right) \right), \tag{6}$$

where ℓ_{LTR} represents the standard LTR loss function (i.e., pointwise, pairwise and listwise).

3 Deployment of LtrGCN

In this section, we introduce the deployment settings of **LtrGCN** at a large-scale industrial search engine. As illustrated in Fig. 3, we present the overall workflow of the real-world deployment and the three-stage design of the search engine as follows: (1) *Webpage Collection*, (2) *Webpage Storage and Indexing*, and (3) *Retrieval and Ranking*.

Webpage Collection. To efficiently navigate the vast expanse of webpages available on the internet, the search engine employs high-performance crawlers known as Web Crawlers. These crawlers play a vital role in collecting and downloading webpages. The Web Crawler operates by systematically scanning a comprehensive list of links, and actively searching for new webpages and updates to existing ones. It selects and stores valid links containing the desired content, creating a downloading list. Utilizing real-time web traffic data, the Web Crawler initiates the downloading process, ensuring timely retrieval of information.

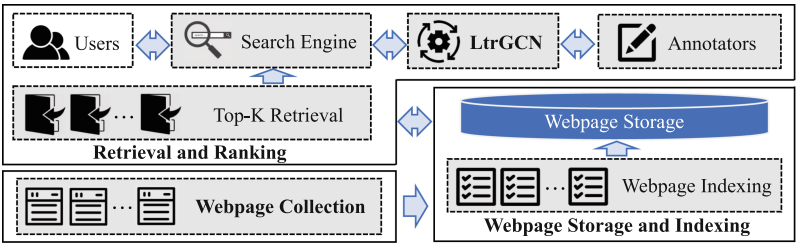


Fig. 3. The overview of the large-scale search engine with **LtrGCN** deployed.

Webpage Storage and Indexing. The search engine stores downloaded webpages in distributed archival storage systems and create efficient indices for high-performance search. These storage systems utilize elastic resources across multiple regional data centers, reducing storage costs. The indexing system balances indexing workloads and achieves superb I/O efficiency through novel key-value operations and in-memory computation. This combination allows search engines to effectively manage large volumes of web content and provide fast and accurate search results.

Retrieval and Ranking. Given a search query, the search engine first retrieves all relevant webpages from the dataset and sorts top-K relevant webpages. Specifically, the search engine adopts a pre-trained language model based semantic retrieval algorithms to enhance the conventional retrieval approach. Then, the search engine pairs each webpage with the query to conduct a query-webpage pair and uses **LtrGCN** to accomplish ranking tasks. To ensure **LtrGCN** can satisfy the rapid shift of internet interest, the search engine periodically picks up new queries and relevant webpages, hires people to annotate scores and re-trains **LtrGCN** with labeled data.

4 Experiments

In this section, we first detail experimental settings. Then, we introduce the results of the offline experiments. Finally, extensive online experiments further demonstrate the effectiveness of **LtrGCN** at a real-world search engine.

4.1 Experimental Settings

Datasets. To demonstrate the effectiveness of our proposed model, we present extensive experiments on a common-used public dataset, MSLR-Web30K [26] and a real-world dataset collected from a large-scale commercial web search engine. **MSLR-Web30K** contains about 30,000 queries and 3,771,125 query-webpage pairs. Each query-webpage pair is represented as a 136-dimensional real-valued feature vector associated with a relevance label with a scale from 0

(irrelevant) to 4 (perfectly relevant). In our experiments, we perform the five-fold cross-validation [26] and report the average results across five folds.

Collected Dataset contains 15,000 queries and over 770,000 query-webpage pairs from a large-scale industrial search engine. In our dataset, each query-webpage pair is also represented as a 120-dimensional real-valued feature vector associated with a relevant score. We randomly split the real-world dataset into a training set (9,000 queries), a validation set (3,000 queries), and a test set (3,000 queries). All features are standardized before being fed into the ranking models in our experiments.

Evaluation Metric. To evaluate the performance of **LtrGCN**, we utilize **Normalized Discounted Cumulative Gain (NDCG)** [15], which is widely adopted in LTR tasks. The NDCG score for the query could be computed as follows:

$$\mathrm{NDCG}_N = \frac{1}{Z} \sum_{i=1}^{N} \frac{2^{y_i} - 1}{\log_2(1+i)}, \tag{7}$$

where Z is a normalization factor that is the ideal order of Discounted Cumulative Gain [14], and y_i is the ranking score of the i^{th} webpage. Additionally, the value of NDCG ranges between $[0, 1]$, and a higher NDCG_N indicates a better LTR model. *In our experiments, we consider the NDCG of the top 5 and 10 results (i.e., $NDCG_5$ and $NDCG_{10}$) for research and business purposes.*

Interleaving [9] is a widely used metric for evaluating the performance of an industrial search engine. In interleaved comparison, two results generated from different systems are delivered to users whose click-through actions would be attributed to the system that delivers the corresponding results.

Good vs. Same vs. Bad (GSB) [41] is an online pairwise metric evaluated by professional annotators. In manual comparison, two results produced by the new system and the legacy system are provided to human experts that are required to judge which result is better.

Loss Functions and Competitor Systems. To evaluate the effectiveness of our proposed model comprehensively, we adopt different state-of-the-art ranking loss functions as follows: **Root Mean Square Error (RMSE)** is a widely used *pointwise* loss. **RankNet** [6] and **LambdaRank** [5] are two popular *pairwise* losses for neural LTR tasks both in research and industry. More particular, LambdaRank multiplies actual gradients with the change in NDCG by swapping the rank positions of the two candidates. **ListNet** [7] and **ListMLE** [36] are two *listwise* losses, which calculate the probability of the ideal permutation based on the ground truth. **ApproxNDCG** [27] and **NeuralNDCG** [25] are two *listwise* loss functions that directly optimize the metric.

For offline experiments, we compare **LtrGCN** with the state-of-the-art ranking models to conduct comprehensive comparisons as follows: **MLP** is a commonly used ranking model. **Context-Aware Ranker (CAR)** [24] is a Trans-

Table 1. Performance on MSLR-Web30k under various ratios of labeled data.

Methods	5%		10%		15%		20%	
	$NDCG_5$	$NDCG_{10}$	$NDCG_5$	$NDCG_{10}$	$NDCG_5$	$NDCG_{10}$	$NDCG_5$	$NDCG_{10}$
RMSE	35.26	38.02	39.12	41.95	43.31	45.65	46.04	48.86
RankNet	34.51	37.43	38.52	41.32	42.54	45.08	45.32	47.89
LambdaRank	35.84	38.50	39.65	42.47	43.69	46.23	46.57	49.56
ListNet	34.90	37.94	38.71	41.76	42.85	45.40	45.63	48.42
ListMLE	33.85	36.95	37.90	40.84	41.88	44.43	44.72	47.26
ApproxNDCG	34.29	37.20	38.32	41.01	42.37	44.70	45.26	47.50
NeuralNDCG	35.36	38.26	39.50	42.10	43.60	45.97	46.39	49.20
CAR_{RMSE}	35.89	38.82	40.24	43.02	44.16	46.51	46.96	49.78
$CAR_{RankNet}$	36.04	38.94	40.46	43.27	44.32	46.62	47.03	49.84
$CAR_{LambdaRank}$	35.83	38.79	40.05	42.84	44.03	46.39	46.83	49.62
$CAR_{ListNet}$	35.52	38.54	39.80	42.60	43.84	46.19	46.59	49.38
$CAR_{ListMLE}$	36.17	39.03	40.61	43.45	44.45	46.81	47.12	49.90
$CAR_{ApproxNDCG}$	35.48	38.47	39.68	42.48	43.72	46.04	46.45	49.26
$CAR_{NeuralNDCG}$	35.66	38.64	39.87	42.70	43.90	46.25	46.72	49.53
XGBoost	33.63	36.94	37.68	40.81	41.67	44.46	44.53	47.28
LightGBM	35.14	38.12	39.63	42.32	43.38	45.98	46.05	49.39
+RMSE	35.62	38.64	39.49	42.29	43.64	46.05	46.42	49.21
+RankNet	35.64	38.67	39.22	42.48	43.98	46.48	46.73	49.24
+LambdaRank	35.96	38.72	40.24	42.75	44.01	46.62	46.95	49.92
+ListNet	36.13	38.87	40.47	43.03	44.26	46.73	47.02	50.14
+ListMLE	36.05	38.91	40.35	42.86	44.15	46.58	46.71	50.03
+ApproxNDCG	36.33	**39.08**	40.94	43.36	44.60	47.01	47.28	50.43
+NeuralNDCG	**36.52**	39.07	**41.16**	**43.62**	**44.72**	**47.35**	**47.49**	**50.76**

former [31]-based ranking model. **XGBoost** [8] and **LightGBM** [17] are two tree-based ranking models with pairwise and listwise loss, respectively.

Considering the high expense of deploying ranking models and the prior experience, we only compare our proposed model with the aforementioned models without including more previous ranking models [2,3,16]. *For online experiments, we only report the improvement between* ***LtrGCN*** *and the* ***legacy system***.

4.2 Offline Experimental Results

Comparative Results. Tables 1 and 2 illustrate offline experimental results of **LtrGCN** compared with baselines on MSLR-Web30K and Collected Dataset on $NDCG_5$ and $NDCG_{10}$. *We use the name of each loss to present MLP with the loss and "+" to represent "****LtrGCN****+".* Intuitively, we could observe that **LtrGCN** outperforms all baselines on two datasets. Specifically, **LtrGCN**+ApproxNDCG obtains the best performance on $NDCG_{10}$ with 5% labeled data on MSLR-Web30K, which gains 1.88% improvements compared with the base model with

Table 2. Performance on Collected Dataset under various ratios of labeled data.

Methods	5%		10%		15%		20%	
	$NDCG_5$	$NDCG_{10}$	$NDCG_5$	$NDCG_{10}$	$NDCG_5$	$NDCG_{10}$	$NDCG_5$	$NDCG_{10}$
RMSE	50.12	53.42	54.45	57.86	57.62	61.34	59.64	64.76
RankNet	49.76	53.07	54.08	57.37	57.41	60.92	59.38	64.25
LambdaRank	51.19	54.24	55.38	58.62	58.38	62.05	61.30	65.28
ListNet	50.48	53.61	54.91	58.04	58.05	61.41	59.92	64.82
ListMLE	49.24	52.46	53.42	56.70	56.61	60.25	58.67	63.68
ApproxNDCG	49.50	52.75	53.73	57.02	57.08	60.61	59.05	64.01
NeuralNDCG	51.05	53.89	55.19	58.31	58.24	61.82	61.21	64.97
CAR_{RMSE}	51.24	53.71	55.42	58.78	58.16	62.08	61.43	65.42
$CAR_{RankNet}$	51.36	53.82	55.49	58.81	58.33	62.15	61.49	65.58
$CAR_{LambdaRank}$	51.60	54.08	55.76	59.13	58.73	62.19	61.62	65.89
$CAR_{ListNet}$	51.68	54.14	55.85	59.24	58.84	62.27	61.75	65.92
$CAR_{ListMLE}$	51.47	53.96	55.52	58.90	58.50	62.12	61.56	65.70
$CAR_{ApproxNDCG}$	51.72	54.17	55.93	59.32	59.02	62.32	61.91	66.08
$CAR_{NeuralNDCG}$	51.98	54.38	56.02	59.43	59.17	62.39	62.04	66.12
XGBoost	50.70	53.19	54.91	58.36	58.16	61.75	61.43	64.75
LightGBM	51.53	53.94	55.74	59.05	58.87	62.28	62.15	65.98
+RMSE	50.68	53.86	54.66	58.35	57.94	61.65	60.13	65.10
+RankNet	50.83	53.92	54.92	58.42	58.23	61.69	60.56	65.19
+LambdaRank	51.34	54.47	55.82	59.06	58.87	62.27	61.62	65.61
+ListNet	51.62	54.60	55.95	59.23	59.06	62.36	61.87	65.88
+ListMLE	51.32	54.23	55.70	59.04	58.62	61.82	61.50	65.43
+ApproxNDCG	52.05	54.57	56.17	59.60	59.39	62.42	62.23	66.17
+NeuralNDCG	**52.16**	**54.79**	**56.36**	**59.78**	**59.62**	**62.53**	**62.64**	**66.29**

ApproxNDCG. **LtrGCN** with NeuralNDCG achieves the best performance on MSLR-Web30K with the other settings. Moreover, **LtrGCN** with NeuralNDCG obtains the best performance against all competitors. Specifically, **LtrGCN**+ NeuralNDCG achieves the improvement with 1.11%, 1.17%, 1.38% and 1.43% that MLP with NeuralNDCG on $NDCG_5$ on Collected Dataset. The performance of our model improves consistently with the label ratio increasing. We also compared **LtrGCN** with LightGCN [12], which cannot be trained at all with "Out of Memory" flagged, due to the sparsity issue.

Ablation Studies. In this study, we conduct a series of ablation studies to investigate the effectiveness of the three steps of **LtrGCN**. Specifically, **LtrGCN** w/o ***Q**-subgraph Generation via Self-tuned Labeling* (*QGSL*) is the model that replaces *QGSL* with a pointwise-based self-trained LightGBM to pseudo data. As for **LtrGCN** w/o ***W**-subgraph Generation via Self-tuned Labeling*, it fails to train the model due to the sparsity issue. **LtrGCN** w/o *GCN-based LTR with **Q**-subgraphs and **W**-subgraphs* (*GLQW*) is the proposed model that directly utilizes the MLP-based LTR model on the combined data. **LtrGCN**

Table 3. Ablation studies of **LtrGCN**+NeuralNDCG on $NDCG_5$ under various ratios of labeled data on two datasets.

Methods	MSLR-Web30K				Collected Dataset			
	5%	10%	15%	20%	5%	10%	15%	20%
+NeuralNDCG	**36.52**	**41.16**	**44.72**	**47.49**	**52.16**	**56.36**	**59.62**	**62.64**
+NeuralNDCG w/o *QGSL*	35.71	40.34	43.87	46.65	51.29	55.48	59.12	61.78
+NeuralNDCG w/o *GLQW*	35.48	40.19	43.70	46.54	51.50	55.65	59.35	61.91
+NeuralNDCG w/o *MLM*	35.92	40.43	44.12	46.90	51.47	55.74	59.48	62.12

Table 4. Ablation studies of **LtrGCN**+ApproxNDCG on $NDCG_{10}$ under various ratios of labeled data on two datasets.

Methods	MSLR-Web30K				Collected Dataset			
	5%	10%	15%	20%	5%	10%	15%	20%
+ApproxNDCG	**39.08**	**43.36**	**47.01**	**50.43**	**54.57**	**59.60**	**62.42**	**66.17**
+ApproxNDCG w/o *QGSL*	37.84	42.15	45.76	49.24	53.21	58.23	61.01	64.74
+ApproxNDCG w/o *GLQW*	38.12	42.40	46.15	49.36	53.39	58.57	61.28	65.62
+ApproxNDCG w/o *MLM*	38.05	42.37	45.98	49.39	53.40	57.92	61.17	64.93

w/o *MLP-based LTR Model* (*MLM*) is the proposed model that utilizes an MLP model with two layers following *GLQW*.

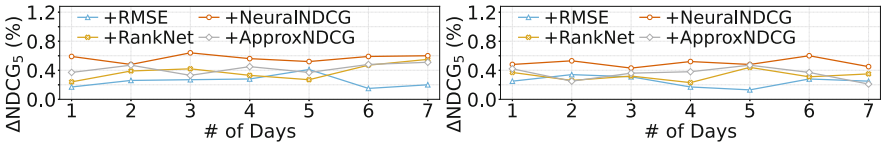
As shown in Tables 3 and 4, we sample the ablation study results of **LtrGCN** with NeuralNDCG on $NDCG_5$ and **LtrGCN** with ApproxNDCG on $NDCG_{10}$ under four ratios of labeled data. Intuitively, we could observe that the three steps contribute to positive improvements for **LtrGCN** under all settings. Specifically, *GLQW* gains the improvement with 1.04%, 0.97%, 1.02% and 0.95% on $NDCG_5$ for **LtrGCN**+NeuralNDCG on MSLR-Web30K. Similarly, *QGSL* improves the performance of **LtrGCN** with ApproxNDCG on $NDCG_{10}$ with 1.36%, 1.37%, 1.41% and 1.43% on Collected Dataset. All results of ablation studies demonstrate the effectiveness of the three steps for **LtrGCN**.

4.3 Online Experimental Results

Interleaving and Manual Evaluation. Table 5 illustrates performance improvements on Δ_{AB} and ΔGSB. We first find that **LtrGCN** trained under 20% labeled data achieves substantial improvements for the online system on two metrics, which demonstrates the practicability and effectiveness of our proposed model. Specifically, the proposed model achieves the most significant improvement with 0.26% and 3.00% on Δ_{AB} and ΔGSB for random queries, respectively. Also, we observe that the proposed model outperforms the *legacy system* for long-tail queries whose search frequencies are lower than 10 per week. Particularly, the largest advantages of Δ_{AB} and ΔGSB are 0.41% and 6.50%.

Table 5. Performance improvements of online evaluation.

Model	Δ_{AB}		ΔGSB	
	Random	Long-Tail	Random	Long-Tail
The Legacy System	–	–	–	–
LtrGCN+ApproxNDCG	0.14%	0.35%	2.50%	5.00%
LtrGCN+NeuralNDCG	**0.26%**	**0.41%**	**3.00%**	**6.50%**



(a) 15% ratio of labeled data (b) 20% ratio of labeled data

Fig. 4. Online comparative performance (ΔNCDG$_5$) of **LtrGCN** for 7 days (t-test with $p < 0.05$ over the baseline).

Online A/B Test. To further verify the effectiveness of **LtrGCN**, we conduct a series of online A/B test with real-world web traffic and compare it with the *legacy system* at a large-scale search engine. According to offline experimental results, we deploy the trained **LtrGCN** under four ratios of labeled data with 5% real-world web traffic, which contains millions of queries per day. The online A/B tests last for 7 days. Due to the page limit, we only report the performance of trained models under 15% and 20% labeled data. Figure 4 illustrates the comparison of **LtrGCN** with the *legacy system* on ΔNCDG$_5$. **LtrGCN** could boost the performance compared with the online *legacy system* all day, which demonstrates that **LtrGCN** is practical for improving the performance of the large-scale search engine. Moreover, we could observe that the trained **LtrGCN** with NeuralNDCG under 15% and 20% labeled data achieves the most significant improvement with 0.64% and 0.60%. The improvement reveals the effectiveness of **LtrGCN**. Eventually, it could be observed that **LtrGCN** performs stably on all days. Online performance is consistent with offline experiment results.

5 Related Work

Learning-to-rank (LTR) techniques generally pertain to machine learning methods that are utilized to solve ranking problems, which are crucial in various applications, such as search engine and recommendation system. Based on the loss function, LTR models could be divided into three types: pointwise [20], pairwise [5,16] and listwise [7,25,27,36]. The pointwise loss formulates the LTR problem into a regression task. The pairwise loss converts two documents into a document pair to treat LTR tasks as binary classification problems. The listwise loss treats the whole document list as a sample and directly optimizes the evaluation metrics [28], such as NDCG [15]. Recently, deep models have been widely

employed in LTR tasks, achieved by minimizing various ranking loss functions in an end-to-end manner [4,22,32,39]. However, deep techniques have led to the study of learning to rank using implicit user feedback, but biases cause unsatisfied performance, so unbiased learning to rank has been proposed to mitigate these biases [30,34,38]. *In this work, we focus on solving practical LTR problems in the industrial scenario.*

In recent years, the modeling graph structure is highlighted by the developed Graph Convolutional Networks (GCN). Existing GCN methods could be categorized into two families [35,40]: spectral GCN and spatial GCN. Spectral GCN leverages graph spectral representations to define graph convolutions, such as SGCN [33], JK-Net [37] and MixHop [1]. Spatial GCN models suggest mini-batch graph training on spatially connected neighbours [42]. Many works have studied the problem of node representation and re-defined graph convolution in the spatial domain, such as GraphSage [11] and ASGCN [13]. It is important to note that several recent attempts offer comprehensive insights on GNNs [19,21]. Moreover, some outstanding works pay more attention to avoiding the unnecessary complexity of GCN, such as SGCN [33] and LightGCN [12]. *In this work, we leverage a GCN-based encoder to learn the representations of query-webpage pairs for the downstream LTR task.*

6 Conclusion

In this work, we design, implement and deploy a GCN-based LTR model **LtrGCN** at a large-scale industrial search engine to address the problem of extreme sparsity of links in query-webpage bipartite graphs and imbalance between queries and webpages for web-scale search. Specifically, **LtrGCN** utilizes two advanced sampling strategies to generate the **Q**-subgraphs and **W**-subgraphs from all query-webpage pairs in the first two steps. Then, **LtrGCN** leverages GCNs to extract feature vectors from **Q**-subgraphs and **W**-subgraphs for LTR as the representation of the query-webpage pair or ranking score prediction. The feature extraction and ranking scores prediction are optimized in an end-to-end manner, so as to enable discriminative feature extraction while preserving structural information in the bipartite graph. To demonstrate the effectiveness of **LtrGCN**, we conduct extensive offline and online experiments compared with a large number of baseline methods. Offline experiment results show that **LtrGCN** could achieve significant performance compared with other competitors. Furthermore, **LtrGCN** significantly boosts the online ranking performance at the industrial search engine, which is consistent with offline results.

Ethical Statement. The authors declare that they have listed all conflicts of interest. This article does not contain any studies with human participants or animals performed by any of the authors. All research and analysis presented in this paper will adhere to ethical principles of honesty, integrity, and respect for human dignity. Sources of information will be cited accurately and fully, and any potential conflicts of interest will be disclosed. Informed consent will be obtained from human subjects involved in

the research, and any sensitive or confidential information will be handled with the utmost discretion. Data they used, the data processing and inference phases do not contain any user personal information. This work does not have the potential to be used for policing or the military. The rights and welfare of all individuals involved in this research project will be respected, and no harm or discomfort will be inflicted upon them. This paper strives to maintain high ethical standards and promote the advancement of knowledge in an ethical and responsible manner.

References

1. Abu-El-Haija, S., et al.: Mixhop: higher-order graph convolutional architectures via sparsified neighborhood mixing. In: Proceedings of the 36th International Conference on Machine Learning, ICML, pp. 21–29 (2019)
2. Ai, Q., Bi, K., Guo, J., Croft, W.B.: Learning a deep listwise context model for ranking refinement. In: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, pp. 135–144 (2018)
3. Ai, Q., Wang, X., Bruch, S., Golbandi, N., Bendersky, M., Najork, M.: Learning groupwise multivariate scoring functions using deep neural networks. In: Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval, SIGIR, pp. 85–92 (2019)
4. Bruch, S., Zoghi, M., Bendersky, M., Najork, M.: Revisiting approximate metric optimization in the age of deep neural networks. In: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, pp. 1241–1244 (2019)
5. Burges, C.J.C., Ragno, R., Le, Q.V.: Learning to rank with nonsmooth cost functions. In: Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, NeurIPS, pp. 193–200 (2006)
6. Burges, C.J.C., et al.: Learning to rank using gradient descent. In: Machine Learning, Proceedings of the Twenty-Second International Conference, ICML, pp. 89–96 (2005)
7. Cao, Z., Qin, T., Liu, T., Tsai, M., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: Machine Learning, Proceedings of the Twenty-Fourth International Conference, ICML, pp. 129–136 (2007)
8. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, SIGKDD, pp. 785–794 (2016)
9. Chuklin, A., Schuth, A., Zhou, K., Rijke, M.D.: A comparative analysis of interleaving methods for aggregated search. ACM Trans. Inf. Syst. (TOIS) **33**(2), 1–38 (2015)
10. Gao, C., Wang, X., He, X., Li, Y.: Graph neural networks for recommender system. In: The Fifteenth ACM International Conference on Web Search and Data Mining, WSDM, pp. 1623–1625 (2022)
11. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS, pp. 1024–1034 (2017)
12. He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., Wang, M.: 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, SIGIR, pp. 639–648 (2020)

13. Huang, W., Zhang, T., Rong, Y., Huang, J.: Adaptive sampling towards fast graph representation learning. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS, pp. 4563–4572 (2018)
14. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (2002)
15. Järvelin, K., Kekäläinen, J.: IR evaluation methods for retrieving highly relevant documents. SIGIR Forum **51**(2), 243–250 (2017)
16. Joachims, T.: Training linear SVMs in linear time. In: Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, SIGKDD, pp. 217–226 (2006)
17. Ke, G., et al.: LightGBM: a highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS, pp. 3146–3154 (2017)
18. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR (2017)
19. Klicpera, J., Bojchevski, A., Günnemann, S.: Predict then propagate: graph neural networks meet personalized pagerank. In: 7th International Conference on Learning Representations, ICLR (2019)
20. Li, P., Burges, C.J.C., Wu, Q.: Mcrank: learning to rank using multiple classification and gradient boosting. In: Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, NeurIPS, pp. 897–904 (2007)
21. Li, Q., Han, Z., Wu, X.: Deeper insights into graph convolutional networks for semi-supervised learning. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI, pp. 3538–3545 (2018)
22. Li, Y., Xiong, H., Kong, L., Zhang, R., Dou, D., Chen, G.: Meta hierarchical reinforced learning to rank for recommendation: a comprehensive study in moocs. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD, pp. 302–317 (2022)
23. Li, Y., et al.: Coltr: semi-supervised learning to rank with co-training and over-parameterization for web search. IEEE Trans. Knowl. Data Eng. (2023)
24. Pobrotyn, P., Bartczak, T., Synowiec, M., Białobrzeski, R., Bojar, J.: Context-aware learning to rank with self-attention. arXiv preprint arXiv:2005.10084 (2020)
25. Pobrotyn, P., Białobrzeski, R.: NeuralNDCG: direct optimisation of a ranking metric via differentiable relaxation of sorting. arXiv preprint arXiv:2102.07831 (2021)
26. Qin, T., Liu, T.Y.: Introducing letor 4.0 datasets. arXiv preprint arXiv:1306.2597 (2013)
27. Qin, T., Liu, T., Li, H.: A general approximation framework for direct optimization of information retrieval measures. Inf. Retr. **13**(4), 375–397 (2010)
28. Qiu, Z., Hu, Q., Zhong, Y., Zhang, L., Yang, T.: Large-scale stochastic optimization of NDCG surrogates for deep learning with provable convergence. In: International Conference on Machine Learning, ICML, pp. 18122–18152 (2022)
29. Shi, M., Tang, Y., Zhu, X., Wilson, D.A., Liu, J.: Multi-class imbalanced graph convolutional network learning. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI, pp. 2879–2885 (2020)
30. Vardasbi, A., de Rijke, M., Markov, I.: Cascade model-based propensity estimation for counterfactual learning to rank. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, pp. 2089–2092 (2020)

31. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS, pp. 5998–6008 (2017)
32. Wang, R., et al.: DCN V2: improved deep & cross network and practical lessons for web-scale learning to rank systems. In: WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, WWW, pp. 1785–1797 (2021)
33. Wu, F., Jr, A.H.S., Zhang, T., Fifty, C., Yu, T., Weinberger, K.Q.: Simplifying graph convolutional networks. In: Proceedings of the 36th International Conference on Machine Learning, ICML, pp. 6861–6871 (2019)
34. Wu, X., Chen, H., Zhao, J., He, L., Yin, D., Chang, Y.: Unbiased learning to rank in feeds recommendation. In: The Fourteenth ACM International Conference on Web Search and Data Mining, WSDM, pp. 490–498 (2021)
35. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst. **32**(1), 4–24 (2021)
36. Xia, F., Liu, T., Wang, J., Zhang, W., Li, H.: Listwise approach to learning to rank: theory and algorithm. In: Machine Learning, Proceedings of the Twenty-Fifth International Conference, ICML, pp. 1192–1199 (2008)
37. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: Proceedings of the 35th International Conference on Machine Learning, ICML, pp. 5449–5458 (2018)
38. Yan, L., Qin, Z., Zhuang, H., Wang, X., Bendersky, M., Najork, M.: Revisiting two tower models for unbiased learning to rank. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, pp. 2410–2414 (2022)
39. Yang, T., Ying, Y.: AUC maximization in the era of big data and AI: a survey. ACM Comput. Surv. **55**(8), 172:1–172:37 (2023)
40. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: a survey. IEEE Trans. Knowl. Data Eng. **34**(1), 249–270 (2022)
41. Zhao, S., Wang, H., Li, C., Liu, T., Guan, Y.: Automatically generating questions from queries for community-based question answering. In: Fifth International Joint Conference on Natural Language Processing, IJCNLP, pp. 929–937 (2011)
42. Zhou, J., et al.: Graph neural networks: a review of methods and applications. AI Open **1**, 57–81 (2020)