Leveraging Graphs to Click-Through Rate Prediction through Test-Time Augmentation

Zhongyu Ouyang
Dartmouth College
Hanover NH, US
zhongyu.ouyang.gr@dartmouth.edu

Mingxuan Ju
Snap Inc.
Seattle WA, US
mju@snap.com

Yanfang Ye
University of Notre Dame
South Bend IN, US
yye7@nd.edu

***Abstract*—Integrating graph knowledge to collaborative filtering (CF) has been proven effective in improving recommendation performance. However, how to leverage graphs to click-through rate (CTR) prediction, where contextual (e.g., user/item/interaction) features are available, remains far less explored. To bridge this gap, we first demonstrate that incorporating graphs to train CTR methods improves the recommendation as well, evaluated by both the predictive and ranking-based metrics. Notably, while the improvement is evident, the substantial computational overheads entailed by graphs are prohibitively expensive for real-world recommendations. In light of this, we propose TagCTR, a test-time augmentation strategy that utilizes graphs only once during testing time to improve the performance of CTR models. The key philosophy is to utilize structural knowledge hidden in graphs on-the-fly during testing while circumventing the notorious computational overheads of message passing during training. TagCTR can be used as a plug-and-play module and easily employed to enhance the ranking ability of various CTR methods with significantly reduced computational overhead. We conduct comprehensive experiments across four benchmark datasets with varying levels of sparsity. Across all datasets, we demonstrate that TagCTR yields noticeable improvements (i.e., 8.1% on average) during testing time with little to no additional computational overheads (i.e., 0.5% on average).**

***Index Terms*—Recommender Systems, Efficient Machine Learning Systems, Test-Time Augmentation**

I. Introduction

Recommender systems are ubiquitous in online applications and have redefined user experiences in product recommendation on e-Commerce platforms [1], [2], personalized advertising [3], [4], and friend recommendation on social media platforms [5]–[7]. Collaborative filtering (CF) is a technique that leverages user-item interactions or preferences (e.g., click, ratings, etc.) to suggest items of interest based on similar user behaviors [8]–[10]. Most of the CF methods learn the ranking prediction task, aiming to learn the user-item similarities so that more relevant items will be ranked at the top of the list. Traditional CF methods are mainly based on matrix factorization [8], [9]. These methods usually assign learnable ID embeddings to all users and items and utilize them to reconstruct interactions between. In comparison with CF, click-through rate (CTR) models leverage additional contextual features including user features (e.g., demographic information), item features (e.g., item descriptions), or interaction features (e.g., transaction timestamp) [11]–[13] to predict the interaction probability between the users and items. These methods pay extra attention to the variability of user preferences for items across different contexts in which they interact with the system and are commonly evaluated by predictive metrics such as AUC and logloss.

Recently, graph neural networks (GNNs) have demonstrated competitive performance for graph-structured data and further inspired a series of research to conduct modeling over the bipartite user-item graphs that broadly exist in recommender systems [14]–[16]. Specifically, GNNs complement CF methods with the message-passing mechanism to capture high-order relational patterns among users and items. Graph-enhanced CF methods can significantly improve the recommendation performance by up to $\sim$40%, compared with traditional CF methods without graphs [14]. However, the exploitation of graphs to CTR prediction where contextual features are incorporated, remains far less explored.

To incorporate graphs into CTR prediction, a recent work [17], denoted as graph convolution machine (GCM), directly utilizes a vanilla GNN to conduct message passing over the user-item bipartite graph, where each user/item node is attached with its corresponding features and each edge is attached with its contextual features of the interaction. Although GCM significantly improves the recommendation performance, it entails several issues that hinder it from being applied to industrial applications. Firstly, GCM is **prohibitively expensive to train** in distributed machine learning infrastructures that most real-world industrial applications utilize. In these scenarios, billions of users and items co-exist and their corresponding ID embeddings are usually distributed across multiple machines. GCM requires repetitive message passings over the user-item bipartite graph during the model training. Hence, training over a single pair of user and item entails multiple queries (quadratic to the training batch size) of their neighbors' embeddings, which incur tremendous overhead due to the limited communication bandwidth between distributed machines. This quadratic computational complexity can be further aggravated by the fact that such an expensive operation is repeatedly executed at each training step. Furthermore, applying GCM into existing context-aware recommender systems **requires tremendous engineering efforts** to support large-scale graph machine learning. Because existing industrial pipelines mostly explore deep models (e.g., DCN [13]) that take i.i.d. tabular data as input, which is intrinsically different from non-i.i.d. graph data. It would be desirable if there exists a strategy that effectively injects graph knowledge yet can be easily extended to recommender systems that are well-established in existing industrial pipelines. To leverage these aforementioned challenges, we aim to answer:

How can we efficiently yet effectively incorporate graph knowledge into CTR methods?

To bridge the above research gap, our study begins by first validating the beneficial synergy between contextual features and graph knowledge in an existing prevalent CTR method. For proof-of-concept purposes, we first design a toy example by substituting vanilla embedding tables for users and items with a simple graph-based encoder which conducts message passing over the user-item bipartite graph. In addition to the traditional predictive metric (i.e., AUC), we also evaluate the toy example's ranking ability via the ranking-based metrics (i.e., NDCG and recall). Experimental results show that the resultant toy example brings noticeable improvements in the recommendation performance over four benchmark datasets with significantly increased computational resource utilization. This is because the toy example requires repetitive message passings over the bipartite graph during the model training. Besides, while the toy example is applicable to a broad range of methods, it requires model re-training and infrastructural modifications, which also hinders its applicability.

To address these issues, our solution diverges from the conventional practice of incorporating graphs during the training phase. We propose TagCTR, a test-time augmentation strategy that utilizes graphs only once during testing time to significantly improve the ranking abilities of CTR models. We emphasize that this improvement is essential, as prior CTR models are primarily evaluated under binary classification. However, the fundamental objective for a human-centered experience in recommendation is to perform a *ranking* task: *in practice, CTR models are extensively used as ranking mechanisms to rank items that have been recalled, aligning closely with their core operational use.* Since our proposed TagCTR is applied only at testing time, it can be employed to enhance various CTR methods with fractional extra computational overheads. Specifically, given a well-trained target CTR method, to predict the interaction between a user and an item, TagCTR first acquires a set of candidate users and items, following our graph-based heuristics curated. Then, TagCTR constructs a new set of user-item pairs based on the candidates and queries the target CTR method to obtain the predicted results over the new set. Lastly, through a simple weighted aggregation, TagCTR combines the predicted results as the final prediction. Through this simple yet effective scheme, TagCTR can significantly improve the ranking ability of the target CTR method while introducing minimal additional overheads to the overall pipeline. Our main contributions can be summarized as:

- We incorporate a graph encoder to a typical CTR method during training, to empirically validate the beneficial synergy between contextual features and graphs from both the predictive and ranking perspectives.
- We propose TagCTR, a test-time augmentation strategy that uses graphs only once during the testing time to improve the ranking abilities of the CTR models. TagCTR effectively utilizes graph knowledge while circumventing notorious computational overheads of message passing during the training. It can be used as a plug-and-play module and easily applied to any CTR methods with extremely simple implementation.
- Comprehensive experiments across four benchmark datasets with varying levels of sparsity are conducted. Across all datasets, we demonstrate that TagCTR yields noticeable improvements on ranking (i.e., **8.1%** on average) during testing time with little to no additional computational overheads (i.e., **0.5%** on average). Furthermore, TagCTR can be applied to a broad range of CTR methods (i.e., seven state-of-the-art CTR methods in our experiments) and consistently improve their ranking capabilities.

II. Related Work

Collaborative Filtering. As a prevalent technique that is widely employed in modern recommender systems, collaborative filtering (CF) makes recommendations based on the idea that similar users tend to have similar preferences [18]. Traditional CF methods aim to reconstruct user-item interactions with parameterized user and item embeddings. They model the reconstruction as a matrix factorization process with the user and item ID embeddings [8], [19], [20]. Besides, some other methods maintain the ID embeddings and adopt neural networks to enhance the interaction modeling in between [11], [21]. Apart from improving the interaction modeling, other recent works focus on refining other aspects such as the objectives and learning paradigms for performance enhancement [9], [22]–[24]. For example, ENMF [22] introduces an efficient methodology for optimizing the mean squared error (MSE) loss across the entire dataset. DirectAU [9] measures representation quality in CF from the perspective of alignment and uniformity. They optimize the two corresponding losses to enforce the properties and improve the performance.

Graph-based Collaborative Filtering. Apart from signals from direct interactions, high-order CF signals in the user-item bipartite graph are crucial for personalized recommendation as well. These signals can be captured by the graph convolution operation in most graph neural networks (GNNs) [25]–[27]. Prior efforts adopt GCN [25] to the user-item interaction graph [14], [28], [29] to capture CF signals in the neighborhood. Later on, LightGCN [15] simplifies the graph convolution in GCN by preserving only linear neighborhood aggregation. In addition to improving the structure of GNNs, recent works [16], [30]–[33] enforce contrastive learning constraints in training the models for improved performance. For example, SGL [30] performs classical graph augmentation to the original bipartite graph to reinforce node representation learning via self-discrimination. SimGCL [16] refines the graph augmentation strategy in SGL with the perturbation of uniform noises and contrasts between the two perturbed graph views. NCL [31] explicitly incorporates potential neighbors into constructing contrastive pairs, and defines a structure-contrastive objective to optimize.

Click-through Rate Prediction. The problem of click-through rate (CTR) prediction is defined as predicting the interaction likelihood between a user and an item given the user ID, item ID, and optional context features as input. The incorporated contextual information includes user demographic features, item description, interaction timestamps, etc. These features additionally consider the variability of user preferences for items across different contexts in which they interact with the system. Early works in CTR seek efficient interactions between the interaction and the contextual information. They preserve low-order feature interactions through the prominent Factorization Machines [34] and seek high-order feature interactions through deep neural networks (DNNs) [11], [12], [35]. Later works modify the layer design within a deep neural network to automatically learn bounded-degree feature interactions [1], [13]. Some recent studies project the features to other predefined hyperspace [36], [37] for more efficient and complex feature interactions.

Our strategy follows the the paradigm for CTR prediction methods and exploits graph knowledge to the models for performance enhancement. Unlike graph-based CF methods which incorporate graph knowledge in training, our strategy simply modifies the inference process during test time. This utilization of graph connectivity differs from graph-based CF methods in completely excluding the user-item interaction graph from the training phase.

III. Enhancing CTR Prediction with Graphs

We first introduce CTR models without conventional graph usage and demonstrate a graph-based encoder that enhances their performance during training, from both the predictive and ranking aspects. However, the significant computational overhead (up to ∼1000% increase) limits its practicality in industry. To address this, we propose TagCTR, a test-time augmentation strategy that utilizes graphs only during testing, effectively leveraging graph knowledge while avoiding the training's computational overhead. We detail our proposed TagCTR in Section III-D.

A. The Paradigm of CTR Prediction

Formally, we denote the IDs of user i and item j as x_i and x_j respectively, and the context features of the interaction between as $\mathbf{c}_{ij} \in \mathbb{R}^{d^c}$, where d^c refers to the dimension of the contextual feature. Let the encoder of user/item IDs be $f(\cdot): \mathbb{R} \to \mathbb{R}^d$, where d refers to the latent dimension of the encoded ID embeddings, and the contextual feature encoder be $h(\cdot): \mathbb{R}^{d^c} \to \mathbb{R}^{d'}$, where d' is the dimension of encoded contextual embeddings. The paradigm of performing CTR prediction is outlined in Figure 1 (c). To predict the probability of user i and item j having an interaction, the input $\mathbf{z}_{ij} \in \mathbb{R}^{2d+d'}$ to a CTR model is defined as:

$$\mathbf{z}_i = f(x_i),\ \mathbf{z}_j = f(x_j),\ \mathbf{z}^c_{ij} = h(\mathbf{c}_{ij}), \tag{1}$$

$$\mathbf{z}_{ij} = \left[\mathbf{z}_i \parallel \mathbf{z}_j \parallel \mathbf{z}^c_{ij}\right], \tag{2}$$

where $\parallel$ refers to the concatenation operation, $\mathbf{z}_i$ and $\mathbf{z}_j$ refer to the latent ID embeddings of user i and item j generated by the encoder respectively, and $\mathbf{z}^c_{ij}$ is the encoded contextual features. We depict this table embedding mechanism in Figure 1 (a). The three components of the input are then concatenated as the final input to the model, as depicted in Figure 1(c).

The traditional encoder $f(\cdot)$ in CTR models is a look-up table encoder, where $f(x_i)$ is the x_i-th entry in a matrix $E \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|)\times d}$, $\mathcal{U}$ is the user set and $\mathcal{I}$ is the item set. We depict the table encoder in Figure 1(a). With the encoded embeddings, each CTR model than adopt a corresponding rating function $r(\cdot): \mathbb{R}^{2d+d'} \to \mathbb{R}$ to predict the interaction likehood. Specifically, we let $p_{ij} = r(\mathbf{z}_{ij})$, where p_{ij} represents how likely user i would click/interact with item j.

To showcase the learning scheme of CTR prediction, we here demonstrate the modeling process of DCN [1], one of the most typical CTR models broadly utilized in both industrial and academic settings. Specifically, DCN comprises two main components: the cross network and the deep network. The cross network, defined by its cross layers, is designed to model explicit feature interactions. Each cross layer is formulated as:

$$\mathbf{z}^{(l+1)} = \mathbf{z}^{(0)}\mathbf{z}^{(l)\mathsf{T}}\mathbf{w}^{(l)} + \mathbf{b}^{(l)} + \mathbf{z}^{(l)}, \tag{3}$$

where $\mathbf{z}^{(l+1)}, \mathbf{z}^{(l)} \in \mathbb{R}^d$ are the input and output column vectors from the l-th cross layer, and $\mathbf{w}^{(l)}, \mathbf{b}^{(l)} \in \mathbb{R}^d$ are the trainable parameters in the l-th cross layer. On the other hand, the deep network is a series of fully connected layers designed to capture complex and non-linear interactions between the input features:

$$\mathbf{h}^{(l+1)} = \text{ReLU}(\mathbf{W}^{(l+1)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}), \tag{4}$$

where $\mathbf{h}^{(l)} \in \mathbb{R}^{d_l}, \mathbf{h}^{(l+1)} \in \mathbb{R}^{d_{l+1}}$ are the input and output of the l-th hidden layer, respectively, and $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{l+1}\times d_l}$ are the learnable parameters in the l-th layer.

For a specific user i and item j, $\mathbf{z}^{(0)} = \mathbf{h}^{(0)} = \mathbf{z}_{ij}$. The outputs of both networks are then concatenated, and fed to a two-class logits layer to generate the final prediction:

$$p_{ij} = \sigma\left([\mathbf{z}^{(L_1)} \parallel \mathbf{h}^{(L_2)}]\mathbf{w}_{\text{logits}}\right), \tag{5}$$

where $\mathbf{z}^{(L_1)} \in \mathbb{R}^{d_1}, \mathbf{h}^{(L_2)} \in \mathbb{R}^{d_2}$ are the L_1-th and L_2-th layer outputs from the cross and deep network respectively, $\mathbf{w}_{\text{logits}} \in \mathbb{R}^{d_1+d_2}$ is the weight vector in the logits layer, and $\sigma(x) = 1/(1+e^{-x})$. The DCN model is then trained with the binary cross entropy loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{\{i,j\}\in T_r} y_{ij}\log(p_{ij}) + (1-y_{ij})\log(1-p_{ij}) + \lambda\sum_{l}\|\mathbf{w}^{(l)}\|^2, \tag{6}$$

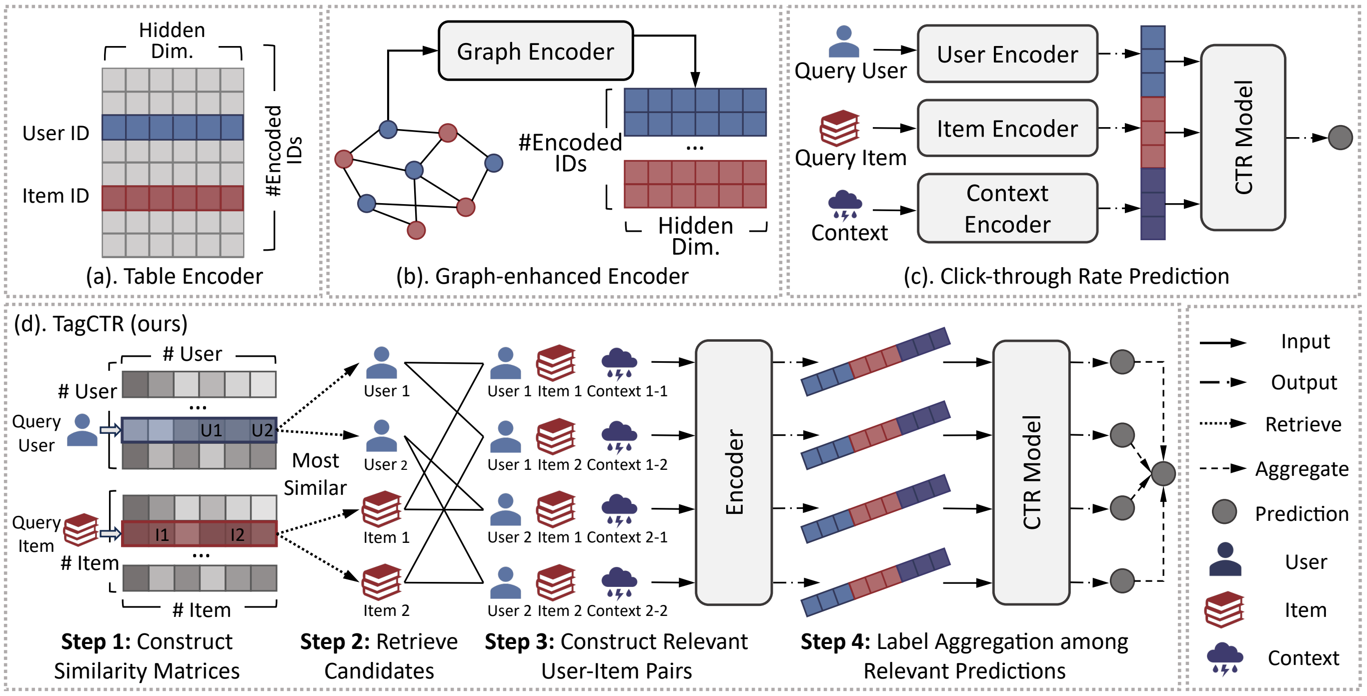

Fig. 1. (a) The table encoder for user/item ID embeddings; (b) The graph-enhanced user/item ID encoder; (c) The paradigm of performing CTR prediction; (d) The overall framework of our strategy explained in steps.

where T_r denotes the training set of positive and negative pairs, p_{ij} is the predicted interaction probability between user i and item j, y_{ij} refer to the binary label (1 for positive and 0 for negative pairs), and λ is the L_2 regularization coefficient.

B. A Naive Graph-based Framework to Enhance Existing Methods on CTR Prediction

In this section, we first introduce the definition of the user-item bipartite graph that depicts rich topological relationships such as co-purchase or shared interests, and then describe how it can be naively incorporated to existing CTR methods in their training processes. Formally, we denote the interaction matrix $\mathbf{M} \in \{0,1\}^{|\mathcal{U}|\times|\mathcal{I}|}$, where $m_{ij} = 1$ represents an observed positive interaction between user i and item j, and $m_{ij} = 0$ otherwise. The interaction graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ is the set of nodes, and $\mathcal{E} = \{(i,j)|\forall i \in \mathcal{U}, \forall j \in \mathcal{I}, m_{ij} = 1\}$ is the set of edges.

A graph-based encoder utilizes the interaction bipartite graph to enhance the ID embedding quality. Unlike a table encoder which independently encodes the users and items, when generating the user and item embeddings, a graph encoder additionally leverages their graph relationships (e.g., co-purchase, shared interests, etc). We depict the graph encoder in Figure 1 (b).

Most existing graph-enhanced CF methods focus on scenarios without the incorporation of contextual features, such as NGCF [14] and LightGCN [15]. However, whether or not their effectiveness can be transferred to CTR methods needs further investigation. Since the message passing mechanism [14], [15], [25] in graph-enhanced CF methods is the key to extracting graph knowledge, a natural way to extend CTR methods with graph knowledge is to include this mechanism in their paradigms. Following this path, we adapt a well-studied linear message passing mechanism [15] to existing CTR methods.

Specifically, let $f_g(\cdot,\cdot): \mathcal{G} \times x \rightarrow \mathbb{R}^d$ be the graph encoder. For user i and item j, $f_g(\cdot,\cdot)$ conducts message passing in each layer to propagate and aggregate information from the neighborhood. The graph-encoded embedding for node i (user or item) is formulated as follows:

$$\mathbf{z}_i = f_g(\mathcal{G}, x_i) = \sum_{l=0}^{L} a_l \mathbf{z}_i^{(l)},$$

$$\text{where} \quad \mathbf{z}_i^{(l)} = \sum_{v \in N_i} \frac{1}{\sqrt{|N_i|}\sqrt{|N_j|}} \mathbf{z}_j^{(l-1)} \quad \text{and} \quad \mathbf{z}_j^{(0)} = f(x_j), \tag{7}$$

In Equation (7), $\mathbf{z}_i^{(l)}$ is the embedding for node i in layer l, N_i is the set of neighbors for node i in $\mathcal{G}$, and a_l is the readout coefficient for each layer-l's embeddings. With the obtained graph-based user and item ID embeddings, we can construct the input features following Equation (2). These input features can further be fed into any CTR method (e.g., DCN) to predict the interaction probabilities.

C. The Benefit of Graphs to CTR Methods

In comparison with user and item ID embeddings obtained from a table encoder as described in Section III-A, those obtained from a graph encoder as introduced in Section III-B possess additional graph topological knowledge. To empirically verify the benefit of such graph knowledge to CTR methods, we design an experiment where all comparison models are identical except their encoders. Specifically, we compare the recommendation performance of a DCNV2 [1] with a table encoder and that of a DCNV2 with a graph encoder. Since the graph encoder additionally incorporates graph knowledge to ID embeddings, and these two frameworks only differ in the encoding, the recommendation performance gap can indicate how integrating graph knowledge affects the CTR method.

In this experiment, we train all models on four benchmark datasets, Yelp2018 [11], Amazon-Books [30], MovieLens-1M [38] and Anime [39]. We evaluate the models by both the predictive metric (i.e., AUC) and the ranking-based metrics (i.e., recall and NDCG). Their results are the averaged performance under five random seeds and are shown in Table I, where the **Tab.** columns represent results of the DCNV2 with a table encoder, and **Graph** columns refer to those of the DCNV2 with a graph encoder.

Compared with DCNV2 equipped with a table encoder, we observe that DCNV2 equipped with a graph-based encoder consistently surpasses its counterpart across the four datasets and all metrics. Adopting a graph-based encoder not only yields higher AUC and lower logloss, but also leads to better ranking abilities demonstrated by higher values in recall and NDCG. The enhancement suggests that the additional graph knowledge incorporated in the graph-based ID embeddings helps improve both the predictive and ranking abilities of the target CTR methods. We emphasize that the enhanced ranking abilities are essential, as CTR models primarily focus on binary classification, yet the fundamental objective in recommendation is to rank the items. In practice, CTR models are extensively used as ranking mechanisms to rank recalled items, aligning closely with their core operational use. A CTR model yielding higher ranking-based metrics suggests its stronger ability to rank more relevant items to the top of the lists, which thus leads to higher-qualified recommendations.

However, this integration incurs a noticeable increase in computational overhead. It incurs on average $\sim$**480%** more overheads for the total training time, and $\sim$**605%** more for the total testing time. These excess computational overheads arise from the message passing operations described in Equation (7) – to acquire the ID embedding of a user/item, the model is required to query representations of all nodes within the 2-hop neighborhood of the node to conduct the further aggregation in between. Moreover, this phenomenon can be further aggravated on dense and large graphs where the average number of neighbors per node is large. For example, it encounters $\sim$**1049%** more time in training and $\sim$**1261%** more in testing in the Anime dataset. Therefore, in industrial applications where billions of users and items construct a massive graph, simply substituting the table encoder with a graph-based encoder is prohibitively expensive and hence impractical.

TABLE I
COMPARATIVE PREDICTIVE AND RANKING RESULTS OF DCNV2 [1] WITH A TABLE ENCODER (DENOTED UNDER THE TAB. COLUMNS), AND THOSE OF DCNV2 WITH A GRAPH ENCODER (DENOTED UNDER THE GRAPH COLUMNS).

Metric	**AUC** $\uparrow$			**Logloss** $\downarrow$		
Dataset	**Tab.**	**Graph**	**%Δ**	**Tab.**	**Graph**	**%Δ**
ML-1M	81.97	82.21	0.30	51.34	51.00	-0.65
Yelp2018	74.51	74.69	0.25	55.42	55.19	-0.41
Amazon-book	80.89	81.07	0.22	39.23	39.06	-0.44
Anime	84.66	84.89	0.27	47.35	47.52	0.35
Metric	**Recall@10** $\uparrow$			**Recall@20** $\uparrow$		
Dataset	**Tab.**	**Graph**	**%Δ**	**Tab.**	**Graph**	**%Δ**
ML-1M	10.73	11.72	9.25	16.81	17.93	6.66
Yelp2018	4.25	4.30	1.18	9.87	10.23	3.59
Amazon-book	3.28	3.73	13.79	7.53	8.30	10.23
Anime	15.73	16.97	7.92	23.76	24.91	4.84
Metric	**NDCG@10** $\uparrow$			**NDCG@20** $\uparrow$		
Dataset	**Tab.**	**Graph**	**%Δ**	**Tab.**	**Graph**	**%Δ**
ML-1M	10.87	12.31	13.26	12.59	13.93	10.59
Yelp2018	2.20	2.16	-1.91	3.75	3.79	1.23
Amazon-book	1.92	2.09	8.98	3.14	3.43	9.23
Anime	13.90	15.73	0.13	16.41	18.07	10.09
Metric	**Time (s) / Train Epoch** $\downarrow$			**Inference Time (s)** $\downarrow$		
Dataset	**Tab.**	**Graph**	**%Δ**	**Tab.**	**Graph**	**%Δ**
ML-1M	1.81	4.06	124.31	0.10	0.24	140.00
Yelp2018	3.53	15.18	330.03	0.17	0.88	417.65
Amazon-book	5.56	29.53	431.12	0.25	1.76	604.00
Anime	9.05	58.30	544.20	0.47	3.52	648.94

D. A Simple yet Effective Solution: Test-time Augmentation for CTR Methods

Although the introduced graph knowledge helps improve the CTR methods, significant computational overheads come along as well. The majority of computational overheads are brought by training with the graph encoder – the encoder repetitively performs the computationally expensive message-passing operation during training, where such an operation is conducted on every iteration.

To address the acute problem of the growth of computational resources, we divert the integration of graph knowledge from the training phase to the testing time and propose a novel strategy, called TagCTR, as shown in Figure 1 (d). Injecting graph knowledge at testing time enjoys two benefits: (i) TagCTR obviates the forward passing and backpropagation entailed by message passing during training in Equation 7, whose computational overhead increases quadratically wrt the dataset density [40], [41], and (ii) TagCTR avoids the computational overheads brought by repetitively performing message passing during training since it only performs message passing once at testing time. TagCTR can be decoupled into four steps illustrated in the following.

Step 1: Constructing Similarity Matrices
The first step of TagCTR is to construct two similarity matrices within users and items. Specifically, the similarity matrix within users, denoted as $\mathbf{A}_u$, and the matrix within items,

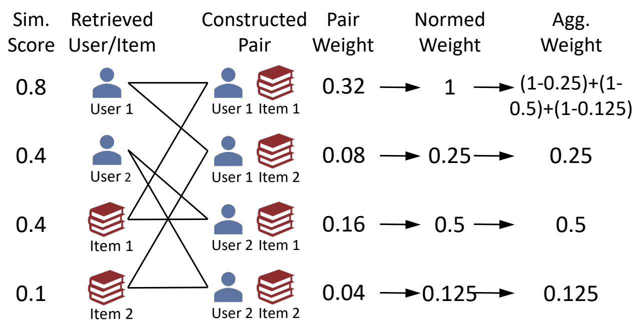

Fig. 2. An example of calculating the label aggregating weights used to aggregate the inference results of the constructed user-item pairs The top-2 similar users and items are retrieved to construct $2 * 2 = 4$ user-item pairs.

denoted as $\mathbf{A}_i$, both depict the co-purchase relationship. The two matrices are formulated based on the interaction matrix $\mathbf{M}$:

$$\begin{aligned}\mathbf{A}_u &= \mathbf{D}_u^{-\frac{1}{2}}\hat{\mathbf{A}}_u\mathbf{D}_u^{-\frac{1}{2}}, \hat{\mathbf{A}}_u = \mathbf{M}\mathbf{M}^\intercal,\\ \mathbf{A}_i &= \mathbf{D}_i^{-\frac{1}{2}}\hat{\mathbf{A}}_i\mathbf{D}_i^{-\frac{1}{2}}, \hat{\mathbf{A}}_i = \mathbf{M}^\intercal\mathbf{M},\end{aligned} \tag{8}$$

where $\mathbf{D}_u \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{U}|}$ and $\mathbf{D}_i \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{I}|}$ are diagonal matrices with $\mathbf{D}_{u/i}[k,k] = \sum_j \mathbf{A}_{u/i}[k,j]$. Intuitively, $\hat{\mathbf{A}}_u[i,j]$ represents the number of items interacted with both user i and j. Similarly, $\hat{\mathbf{A}}_i[k,j]$ represents the number of users interacted with both item k and j. For popular users/items associated with massive interactions, their corresponding entries in $\hat{\mathbf{A}}_{u/i}$ tend to numerically dominate the corresponding entries of unpopular users/items associated with relatively fewer interactions. In other words, the similarity scores between popular users/items are consistently larger than those between unpopular users/items. To remove the bias caused by node popularity (i.e., node degree), we further normalize $\hat{\mathbf{A}}_{u/i}$ by accounting for the varying degree of each user/item, and denote the normalized similarity matrices as $\mathbf{A}_{u/i}$. This normalization step prevents nodes (users or items) with disproportionately high degrees from occupying overly high similarity scores [25]. Therefore, we regard $\mathbf{A}_{u/i}$ as the similarity matrices of users/items.

Step 2: Retrieving Relevant User/Item Candidates
TagCTR then retrieves relevant users and items based on the similarity matrices $\mathbf{A}_{u/i}$. Specifically, for user i, we obtain n_k users with the corresponding top-n_k similarity scores in $\mathbf{A}_u[i]$, denoted as $\mathcal{U}_i$, where $\mathbf{A}_u[i]$ refers to the i-th row of $\mathbf{A}_u$. Similarly, for item j, we obtain n_k items with the corresponding top-n_k similarity scores in $\mathbf{A}_i[j]$, denoted as $\mathcal{I}_j$. The corresponding $2n_k$ similarity scores in $\mathbf{A}_{u/i}$ are extracted to further calculate the aggregation weights of the later constructed user-item pairs. Intuitively, the higher the user/item similarity score is, the more aggregation weights should be assigned to the involved user-item pairs.

Step 3: Constructing Relevant User-item Pairs
Since the users and items are selected based on the similarities in $\mathbf{A}_{u/i}$, which is constructed based on the collaborative filtering signals in $\mathbf{M}$, the interaction signals between the selected users and items naturally contain structural knowledge. Consequently, TagCTR constructs a set of relevant user-item pairs by combining each user in $\mathcal{U}_i$ with each item in $\mathcal{I}_j$. This set contains n_k^2 pairs and is defined as $\{\mathcal{M}_{ij} : (u,v)|\forall u \in \mathcal{U}_i, \forall v \in \mathcal{I}_j\}$. For each user-item pair in $\mathcal{M}_{ij}$, we calculate its weight as the multiplication of the corresponding user and item similarity scores, as shown in the middle of Figure 2. We denote the weight matrix for all user-item pairs in $\mathcal{M}_{ij}$ as $\text{Weight}(\mathcal{M}_{ij})$, where $\text{Weight}(\mathcal{M}_{ij})[u,v]$ represents the weight for the user u and item v pair.

Step 4: Label Aggregation Among Relevant Pairs
After obtaining the n_k^2 user-item pairs, TagCTR queries the CTR model for the corresponding n_k^2 inference results, and aggregates them based on each pair's weight $\text{Weight}(\mathcal{M}_{ij})[u,v]$:

$$p\prime_{ij} = \frac{\sum_{(u,v)\in\mathcal{M}_{ij}} \text{Weight}(\mathcal{M}_{ij})[u,v] * r(\mathbf{z}_{uv})}{\sum_{(u,v)\in\mathcal{M}_{ij}} \text{Weight}(\mathcal{M}_{ij})[u,v]}, \tag{9}$$

where $\mathbf{z}_{uv} = [f(x_u) \parallel f(x_v) \parallel h(\mathbf{c}_{uv})]$ is the embedded features and $r(\cdot)$ refers to an arbitrary trained CTR model such as the one we describe in Section III-A. The aggregated inference result p'_{ij} is the final output of TagCTR.

Although $\text{Weight}(\mathcal{M}_{ij})$ is feasible to aggregate the inference results, we notice that the contribution proportion of the most similar user-item pair (i.e., pairs constructed with the most similar user and item) is too small. For the example in Figure 2, the most similar pair only contributes to $1/(1 + 0.25 + 0.5 + 0.125) \approx 53\%$ of the final result. This overly small proportion may deviate the final result too much from the prediction of the original user-item pair, and thereby downgrade the performance. To resolve this issue, we adopt a pair-wise aggregation mechanism: (i) we first normalize the weights in $\text{Weight}(\mathcal{M}_{ij})$ by dividing them by the maximum value in the matrix. This step normalizes the values in $\text{Weight}(\mathcal{M}_{ij})$ between 0 to 1; (ii) we further modify the aggregation weight for the most similar pair (i.e., the pair with the weight value as 1) by considering aggregating the pairwise inference result between this pair and the other pairs, as the following:

$$\begin{aligned}\text{Weight}(\mathcal{M}_{ij})[u',v'] &= \sum_{(u,v)\neq(u',v')} (1 - \text{Weight}(\mathcal{M}_{ij})[u,v]),\\ &\text{where } u',v' = \arg\max_{(u,v)} \text{Weight}(\mathcal{M}_{ij})[u,v].\end{aligned} \tag{10}$$

The final result of TagCTR is derived using Equation 9 with the maximum entry in $\text{Weight}(\mathcal{M}_{ij})$ modified as above. In the example in Figure 2, the proportion of the most similar pair is modified to $(1-0.25)+(1-0.5)+(1-0.125) = 2.125$. This modified maximum weight is approximately 71% of the sum of the weights of all pairs, which is higher than 53% before its modification. In the following experiment section, we validate this design and empirically demonstrate how the proportion changes as n_k increases.

TABLE II
THE RANKING PERFORMANCE OF APPLYING OUR APPROACH TO THE BASELINES OVER FOUR BENCHMARK DATASETS, WHERE '*Original*' REPRESENTS THE PERFORMANCE EVALUATED FROM THE ORIGINAL MODELS, '***+Tag.***' REPRESENTS THE PERFORMANCE EVALUATED FROM THE MODELS APPLIED WITH TAGCTR, AND '%Δ' REPRESENTS THE RELATIVE CHANGE IN PERFORMANCE OF APPLYING TAGCTR TO THE ORIGINAL MODELS.

	NDCG@10			**NDCG@20**			**Recall@10**			**Recall@20**		
Model	**Original**	**+Tag.**	**%Δ**	**Original**	**+Tag.**	**%Δ**	**Original**	**+Tag.**	**%Δ**	**Original**	**+Tag.**	**%Δ**
						ML-1M						
NFM	9.42	9.44	0.21	10.88	10.96	0.74	8.97	9.03	0.69	14.26	14.50	1.67
DeepFM	10.17	10.41	2.32	11.70	12.02	2.72	9.61	9.98	3.87	15.08	15.68	3.98
xDeepFM	8.08	8.17	1.06	9.65	9.82	1.72	8.04	8.19	1.92	13.13	13.52	3.00
DCN	10.07	11.19	11.08	11.58	12.73	9.98	9.47	10.17	7.33	14.97	16.11	7.64
DCNv2	10.76	10.91	1.39	12.26	12.52	2.10	10.02	10.26	2.39	15.55	16.12	3.69
AutoInt	8.72	9.18	5.23	10.32	10.82	4.84	8.66	9.19	6.07	13.96	14.60	4.60
EulerNet	8.91	9.11	2.34	10.51	10.96	4.32	8.73	9.24	5.86	14.08	14.83	5.28
						Yelp2018						
NFM	2.92	3.35	14.58	4.58	5.10	11.35	5.28	6.06	14.77	11.23	12.30	9.58
DeepFM	1.95	2.18	11.78	3.52	3.83	8.63	3.90	4.33	10.86	9.66	10.32	6.86
xDeepFM	2.53	2.76	9.34	4.12	4.42	7.28	4.68	5.11	9.19	10.45	11.09	6.11
DCN	2.20	2.46	11.73	3.75	4.12	10.04	4.25	4.75	11.78	9.87	10.75	8.93
DCNv2	2.07	2.30	10.81	3.63	3.94	8.42	4.06	4.49	10.55	9.77	10.47	7.16
AutoInt	1.98	2.20	11.20	3.53	3.82	8.33	3.92	4.32	10.37	9.57	10.21	6.64
EulerNet	3.39	3.68	8.74	5.08	5.44	7.09	5.84	6.34	8.59	11.90	12.61	5.93
						Amazon-Books						
NFM	2.67	3.04	13.69	4.06	4.47	10.25	4.52	5.09	12.42	9.10	9.81	7.85
DeepFM	2.40	2.65	10.69	3.69	4.01	8.50	4.00	4.43	10.86	8.41	9.01	7.04
xDeepFM	2.24	2.46	9.74	3.46	3.75	8.32	3.69	4.08	10.62	7.88	8.46	7.34
DCN	1.91	2.20	15.15	3.13	3.52	12.40	3.28	3.81	16.23	7.53	8.27	9.94
DCNv2	2.42	2.68	10.65	3.76	4.09	8.72	4.16	4.60	10.52	8.67	9.30	7.31
AutoInt	2.35	2.57	9.63	3.65	3.94	7.89	3.93	4.30	9.47	8.36	8.92	6.75
EulerNet	2.46	2.64	7.57	3.68	3.91	6.37	3.93	4.30	9.30	8.08	8.57	6.14
						Anime						
NFM	14.74	14.68	-0.37	17.60	17.69	0.55	17.33	17.71	2.19	26.11	26.92	3.10
DeepFM	14.22	14.58	2.56	16.55	17.59	6.30	17.09	17.70	3.61	26.02	26.85	3.20
xDeepFM	15.09	15.08	-0.01	18.19	18.30	0.60	18.12	18.36	1.34	27.50	28.11	2.23
DCN	13.90	14.17	1.94	16.41	17.03	3.77	15.73	16.50	4.90	23.76	25.26	6.32
DCNv2	14.98	15.30	2.10	17.66	18.12	2.62	17.08	17.73	3.79	25.65	26.72	4.19
AutoInt	12.32	13.27	7.66	15.00	16.05	7.04	14.59	15.99	9.54	22.75	24.44	7.45
EulerNet	12.91	13.42	3.95	15.39	16.03	4.20	14.86	15.73	5.83	22.74	23.96	5.37

IV. EXPERIMENT

In this section, we first outline experimental setups in Section IV-A. Then in Section IV-B, we apply TagCTR to multiple well-trained CTR methods and evaluate them through both the predictive and ranking-based metrics. Furthermore, in Section V we show that TagCTR is a much more efficient strategy than the framework mentioned in Section III-B. Finally, in Section VI, we analyze how the number of considered neighboring user/item n_k affects TagCTR.

A. Setup

1) Datasets: We select four publicly available recommendation benchmark datasets for the experiments. Specifically, Yelp2018 [11] and Amazon-Books [30] are relatively sparse compared to MovieLens-1M [38] and Anime [39]. The statistics of the datasets are shown in Table III. For all datasets, we convert explicit user-to-item ratings to binary labels through thresholding. The rating for ML-1M, Yelp2018, and Amazon-Books ranges from 1 to 5, and we adopt 4 as the threshold. For Anime, the ratings range from 1 to 10 and we set the threshold as 7. We randomly split datasets with a ratio of 0.8/0.1/0.1 for training, validation, and testing, respectively.

2) Baselines: We select seven models as our baselines, including NFM [11], DeepFM [12], xDeepFM [35], DCN [13], DCNV2 [1], AutoInt [36], and EulerNet [37]. Specifically: NFM [11], DeepFM [12], and xDeepFM [35] combine the advantages of Factorization Machines [34] and deep neural networks to capture complex non-linear and high-order feature interactions. DCN [1], [13] learns explicit and implicit feature interactions through a cross-network and a deep neu-

TABLE III
THE STATISTICS OF FOUR BENCHMARK DATASETS.

Dataset	#User	#Item	#Interaction	Sparsity
ML-1M	6,041	3,261	998,539	0.9493
Yelp2018	77,278	45,639	2,103,896	0.9994
Amazon-Books	68,498	65,549	2,954,716	0.9993
Anime	55,119	7,364	6,270,078	0.9846

ral network, respectively. AutoInt [36] utilizes self-attentive neural networks to learn more effective feature interactions. EulerNet [37] learns the interactions of high-order features by transforming their exponential powers into linear combinations of the modulus and phase of complex features.

3) Evaluation: For all the baselines, we employ the AdamW optimizer for optimization and adopt binary cross-entropy as the loss function to train them on the training set. We run a fixed number of grid searches over all the baseline models' provided hyper-parameters for their best AUC performance on the validation set. With the best hyper-parameters, we train the models under five random seeds and save all the checkpoints. For each baseline model, we evaluate our TagCTR with the inference results from the corresponding saved models. We tune the number of candidates n_k with the same amount of grid-searches and re-evaluate the model performance. The ranking ability is evaluated by two ranking-based metrics, NDCG@K and Recall@K, both of which assign higher scores to models that accurately rank the most relative/highly rated items. Following previous works, we choose the predictive metric as the AUC score. All reported results are averaged over the results under the five random seeds. The training and inference processes are conducted on an NVIDIA RTX 3090 GPU with 24 GB of memory, and the user-user and item-item similarity matrices are pre-computed on a standard commercial CPU with 128 GB of RAM. We adopt the recommender system library named Recbole [42] to conduct all the experiments.

B. Performance Improvement by TagCTR

For all the baseline models, we first evaluate their recommendation performance on the four benchmark datasets by the ranking-based metrics, and denote the results under the **Original** columns in Table II. We apply our TagCTR to each of the baseline models, and report the re-evaluated ranking performance under the **+TagCTR** columns in Table II. Additionally, we report the relative performance change of applying TagCTR to the original method and denote the statistics under the **%Δ** columns. From the table, we observe that: (i) our strategy stably improves the performance over the original model, and only demonstrates a slight performance downgrade in some rare cases (i.e., NFM and xDeepFM in Anime). These results validate that our strategy is generally effective in improving ranking performance across various CTR models and benchmark datasets. (ii) the improvements are relatively evident in sparse datasets (i.e., Yelp2018, Amazon-Books) than dense datasets (i.e., ML-1M, Anime). This is because, in sparse datasets where the number of interactions associated with each node is relatively small, the node embeddings receive less collaborative filtering signals from their neighbors during training. Therefore, at testing time, our TagCTR is able to compensate for insufficient training of the user/item embeddings with the injected graph knowledge. In contrast, user/item embeddings trained in dense datasets receive more training signals from a larger number of neighbors, resulting in less space for improvement at testing time by TagCTR.

In addition to the ranking-based metrics, we also evaluate our strategy with AUC as shown in Table IV. In comparison with the original CTR models, we see that our strategy yields a slight performance downgrade in dense datasets (i.e., ML-1M, Anime), and comparable performance in sparse datasets (i.e., Yelp2018, Amazon-books). We credit the decreased AUC in dense graphs to the adoption of binary cross-entropy loss in training: when the models are trained on dense graphs, their abilities to distinguish between relevant and irrelevant items are more enhanced than those trained on sparse graphs – the user and item embeddings receive more training signals from a larger number of neighboring nodes. Therefore, any modification in predicted scores can impair the well-trained distinguish abilities, resulting in a downgraded AUC.

C. Beyond AUC: Practical Considerations

In terms of the practical deployment, we particularly emphasize that the slightly AUC metric downgrade is both **reasonable** and **acceptable**: *(i)* our strategy modifies the predicted scores by aggregating all relevant user-item predictions, which directly affect the item rankings. Ranking-based metrics, such as NDCG@K and Recall@K, are very sensitive to slight modifications of the predicted scores, as they directly affect the item rankings. In contrast, AUC is non-linearly affected by subtle modifications in predicted scores. As long as the predicted score is within the decision threshold, the corresponding AUC does not change. Therefore, AUC is less prone to demonstrate the direct influence of the modifications in predicted scores, and the deployment remains near the original performance level; *(ii)* More importantly, practical RecSys is expected to output the recommended items orderly, rather than equally showing the users a bundle of items. Although CTR methods are conventionally trained for binary classification, its fundamental objective in recommendation is to perform a ranking task. In practice, CTR models are extensively used as ranking tools to rank items that have been recalled, aligning closely with their core operational use. Therefore, enhancing CTR methods' ranking capabilities directly contributes to the quality of practical recommendations.

V. TIME EFFICIENCY FOR TAGCTR

We depict the performance and time relative to ones for DCN in Figure 3. From the figure, we see that applying TagCTR to DCN achieves performance comparable to that of the naive graph-enhanced DCN, with only a marginal extra time overall. Applying TagCTR to a well-trained CTR method does not introduce additional training time, and only

TABLE IV
THE COMPARED AUC PERFORMANCE OF APPLYING OUR APPROACH TO THE BASELINES, WHERE '*Original*' REPRESENTS THE PERFORMANCE EVALUATED FROM THE ORIGINAL MODELS, '***+Tag.***' REPRESENTS THE PERFORMANCE EVALUATED FROM THE MODELS APPLIED WITH TAGCTR, AND '$\%\Delta$' REPRESENTS THE RELATIVE CHANGE IN AUC OF APPLYING TAGCTR TO THE ORIGINAL MODELS.

Dataset	ML-1M			Yelp2018			Amazon-Books			Anime		
AUC	**Original**	**+Tag.**	**$\%\Delta$**	**Original**	**+Tag.**	**$\%\Delta$**	**Original**	**+Tag.**	**$\%\Delta$**	**Original**	**+Tag.**	**$\%\Delta$**
NFM	81.51	81.09	-0.52	74.35	74.40	0.07	80.76	80.85	0.11	84.81	83.98	-0.99
DeepFM	82.11	81.66	-0.55	74.61	74.66	0.06	80.97	81.05	0.10	84.84	84.09	-0.88
xDeepFM	81.09	80.77	-0.40	74.56	74.61	0.06	80.95	81.03	0.09	84.75	83.89	-1.02
DCN	81.97	81.45	-0.63	74.51	74.55	0.06	80.89	80.99	0.12	84.66	83.79	-1.03
DCNv2	82.13	81.63	-0.62	74.51	74.56	0.07	80.82	80.92	0.12	84.61	83.78	-0.98
AutoInt	82.10	81.62	-0.58	74.60	74.64	0.06	80.95	81.03	0.11	84.69	83.76	-1.09
EulerNet	82.10	81.62	-0.59	74.59	74.62	0.03	80.96	81.04	0.10	84.67	83.80	-1.03

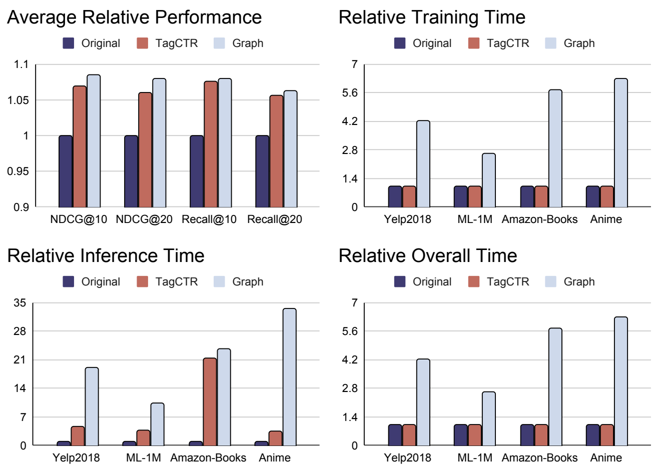

Fig. 3. The average relative performance is averaged over the four benchmark datasets, and is scaled based on the original model performance. The relative training and inference time is calculated wrt that of the original model, and the overall time is the summation of training and inference time.

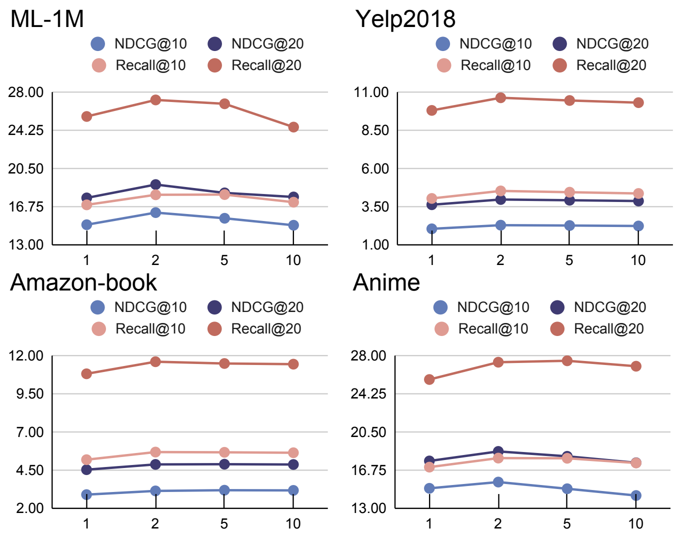

Fig. 4. The ranking-based performance of DCN to the varied number of neighbors n_k in label aggregation.

quadratically increases the inference time wrt n_k. For each dataset, the users and items most relevant n_k neighbors can be pre-computed and applied to all CTR methods, making the corresponding computational overhead one-off relative to a dataset. Therefore, applying TagCTR to well-trained CTR methods effectively enjoys the merits while introducing little to no additional time overall.

VI. EFFECT OF n_k TO TAGCTR

The only hyper-parameter of our TagCTR is the number of relative user/item candidates, denoted as n_k. To analyze how the number of relative user/item candidates n_k in TagCTR affects the performance, we apply TagCTR to DCN with varied n_k in $\{1, 2, 5, 10\}$ to compare the performance. The results are shown in Figure 4. Note that $n_k = 1$ refers to the original DCN model. From the figure, we observe that TagCTR, when applied to dense graphs such as Ml-1M and Anime, demonstrates improved performance with smaller n_k's. Conversely, on sparse graphs like Yelp2018 and Amazon-books, TagCTR yields better results with larger n_k's. This is because the value of n_k controls the range of the neighborhood considered for graph knowledge extraction. When n_k is small, the extracted graph knowledge is sufficient to improve CTR methods trained on dense graphs but insufficient for those trained on sparse graphs.

We further analyze how n_k affects the resultant contribution proportion of the most similar user-item pair in TagCTR. Specifically, we apply TagCTR to DCN with varied n_k in $\{1, 2, 5, 10\}$, and depict the contribution proportion distribution of the most similar user-item pair in Figure 5. From Figure 5, we see that as n_k increases (i) the contribution proportion for the most similar user-item pair increases, and (ii) the distribution is more concentrated (i.e., it spans fewer values). Intuitively, the two phenomena suggest that within a close neighborhood (i.e., n_k is small), TagCTR adjusts the contribution proportion of the most similar pair in a wider range with relatively smaller numerical values. Conversely, within an extensive neighborhood (i.e., n_k is large), TagCTR conservatively adjusts the proportions in a tighter range with relatively larger values.

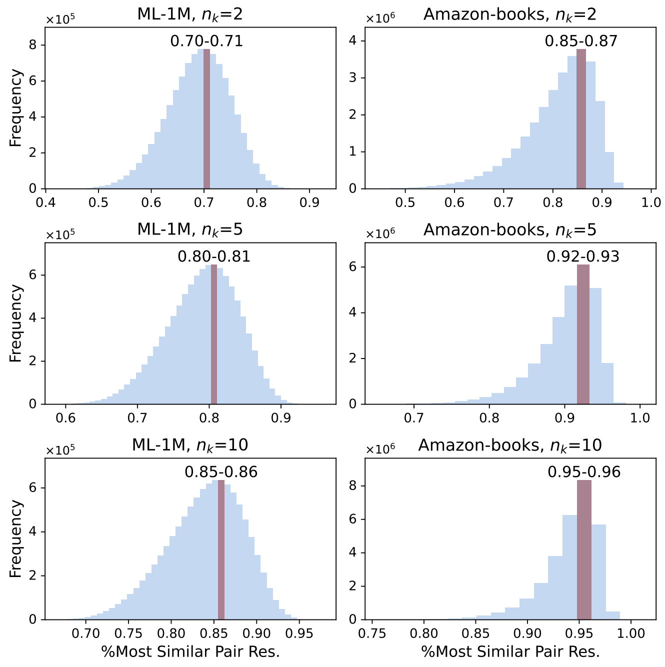

Fig. 5. Contribution proportion distribution for the most similar user-item pair, where most common proportion values are highlighted and denoted above.

VII. Conclusion

In this work, we investigate how to efficiently leverage graphs to CTR prediction where contextual features are available. We first demonstrate a naive graph-enhanced framework, where graph knowledge is incorporated in the encoder of CTR methods via the message-passing operation. While this framework is empirically effective in improving recommendation performance, the substantial computational overheads entailed by training with a graph encoder render this framework prohibitively expensive for real-world applications. In light of this, we propose TagCTR, a test-time augmentation strategy for CTR methods that utilizes graphs only once during testing to improve the target methods. Our TagCTR can be used as a plug-and-play module, and can be easily employed various CTR methods with little to no additional computational cost. We conduct comprehensive experiments across four benchmark datasets with various densities to demonstrate that TagCTR brings noticeable ranking performance improvements (i.e., **8.1%** on average) during testing time with little to no additional computational overheads (i.e., **0.5%** on average).

References

[1] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, "Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," in *ACM Web Conference*, 2021.
[2] J. B. Schafer, J. Konstan, and J. Riedl, "Recommender systems in e-commerce," in *ACM conference on Electronic commerce*, 1999.
[3] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems*, 2015.
[4] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in Neural Information Processing Systems*, 2013.
[5] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: social recommendation using probabilistic matrix factorization," in *ACM International Conference on Information and Knowledge Management*, 2008.
[6] M. Jamali and M. Ester, "A matrix factorization technique with trust propagation for recommendation in social networks," in *ACM Recommender Systems conference*, 2010.
[7] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *ACM Web Conference*, 2019.
[8] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *The Conference on Uncertainty in Artificial Intelligence*, 2009.
[9] C. Wang, Y. Yu, W. Ma, M. Zhang, C. Chen, Y. Liu, and S. Ma, "Towards representation alignment and uniformity in collaborative filtering," in *ACM SIGKDD Conference*, 2022.
[10] Y. Koren, S. Rendle, and R. Bell, "Advances in collaborative filtering," *Recommender systems handbook*, 2021.
[11] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *ACM SIGIR Conference*, 2017.
[12] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," in *International Joint Conference on Artificial Intelligence*, 2017.
[13] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," 2017.
[14] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *ACM SIGIR Conference*, 2019.
[15] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *ACM SIGIR Conference*, 2020.
[16] J. Yu, H. Yin, X. Xia, T. Chen, L. Cui, and Q. V. H. Nguyen, "Are graph augmentations necessary? simple graph contrastive learning for recommendation," in *ACM SIGIR Conference*, 2022.
[17] J. Wu, X. He, X. Wang, Q. Wang, W. Chen, J. Lian, and X. Xie, "Graph convolution machine for context-aware recommender system," *Frontiers of Computer Science*, 2022.
[18] Y. Wei, W. Liu, F. Liu, X. Wang, L. Nie, and T.-S. Chua, "Lightgt: A light graph transformer for multimedia recommendation," in *ACM SIGIR Conference*, 2023.
[19] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, 2009.
[20] H. Zhang, F. Luo, J. Wu, X. He, and Y. Li, "Lightfr: Lightweight federated recommendation with privacy-preserving matrix factorization," *ACM Transactions on Information Systems*, 2023.
[21] Y. Tay, L. Anh Tuan, and S. C. Hui, "Latent relational metric learning via memory-based attention for collaborative ranking," in *ACM Web Conference*, 2018.
[22] C. Chen, M. Zhang, Y. Zhang, Y. Liu, and S. Ma, "Efficient neural matrix factorization without sampling for recommendation," *ACM Transactions on Information Systems*, 2020.
[23] D. Lee, S. Kang, H. Ju, C. Park, and H. Yu, "Bootstrapping user and item representations for one-class collaborative filtering," in *ACM SIGIR Conference*, 2021.
[24] A. Zhang, L. Sheng, Z. Cai, X. Wang, and T.-S. Chua, "Empowering collaborative filtering with principled adversarial contrastive loss," in *Advances in Neural Information Processing Systems*, 2023.
[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.
[26] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2017.

[27] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems*, 2017.
[28] R. v. d. Berg, T. N. Kipf, and M. Welling, “Graph convolutional matrix completion,” in *ACM SIGKDD Conference*, 2018.
[29] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph convolutional neural networks for web-scale recommender systems,” in *ACM SIGKDD Conference*, 2018.
[30] J. Wu, X. Wang, F. Feng, X. He, L. Chen, J. Lian, and X. Xie, “Self-supervised graph learning for recommendation,” in *ACM SIGIR Conference*, 2021.
[31] Z. Lin, C. Tian, Y. Hou, and W. X. Zhao, “Improving graph collaborative filtering with neighborhood-enriched contrastive learning,” in *ACM Web Conference*, 2022.
[32] X. Cai, C. Huang, L. Xia, and X. Ren, “Lightgcl: Simple yet effective graph contrastive learning for recommendation,” in *International Conference on Learning Representations*, 2022.
[33] Y. Ma, Y. He, A. Zhang, X. Wang, and T.-S. Chua, “Crosscbr: cross-view contrastive learning for bundle recommendation,” in *ACM SIGKDD Conference*, 2022.
[34] S. Rendle, “Factorization machines,” in *IEEE ICDM*, 2010.
[35] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, “xdeepfm: Combining explicit and implicit feature interactions for recommender systems,” in *ACM SIGKDD Conference*, 2018.
[36] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, “Autoint: Automatic feature interaction learning via self-attentive neural networks,” in *ACM International Conference on Information and Knowledge Management*, 2019.
[37] Z. Tian, T. Bai, W. X. Zhao, J.-R. Wen, and Z. Cao, “Eulernet: Adaptive feature interaction learning via euler’s formula for ctr prediction,” in *ACM SIGIR Conference*, 2023.
[38] GroupLens Research, “Movielens 1m dataset,” https://grouplens.org/datasets/movielens/1m/, 2023, accessed: 2023-11-11.
[39] Cooper Union, “Anime recommendations database,” https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database, 2023, accessed on: 2023-11-11.
[40] X. Han, T. Zhao, Y. Liu, X. Hu, and N. Shah, “Mlpinit: Embarrassingly simple gnn training acceleration with mlp initialization,” in *International Conference on Learning Representations*, 2023.
[41] S. Zhang, Y. Liu, Y. Sun, and N. Shah, “Graph-less neural networks: Teaching old mlps new tricks via distillation,” in *International Conference on Learning Representations*, 2022.
[42] W. X. Zhao, S. Mu, Y. Hou, Z. Lin, Y. Chen, X. Pan, K. Li, Y. Lu, H. Wang, C. Tian *et al.*, “Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms,” in *ACM International Conference on Information and Knowledge Management*, 2021.