Implicit User Awareness Modeling via Candidate Items for CTR Prediction in Search Ads

Kaifu Zheng*, Lu Wang*, Yu Li†, Xusong Chen, Hu Liu, Jing Lu,
Xiwei Zhao, Changping Peng, Zhangang Lin, Jingping Shao
{zhengkaifu,wanglu241,liyu1078,chenxusong3,liuhu1,lvjing12,zhaoxiwei,pengchangping,linzhangang,shaojingping}@jd.com
Business Growth BU, JD.com
Beijing, China

ABSTRACT

Click-through rate (CTR) prediction plays a crucial role in sponsored search advertising (search ads). User click behavior usually showcases strong comparison patterns among relevant/competing items within the user awareness. Explicit user awareness could be characterized by user behavior sequence modeling, which however suffers from issues such as cold start, behavior noise and hidden channels. Instead, in this paper, we study the problem of modeling implicit user awareness about relevant/competing items. We notice that candidate items of the CTR prediction model could play as surrogates for relevant/competing items within the user awareness. Motivated by this finding, we propose a novel framework, named CIM (Candidate Item Modeling), to characterize users' awareness on candidate items. CIM introduces an additional module to encode candidate items into a context vector and therefore is plug-and-play for existing neural network-based CTR prediction models. Offline experiments on a ten-billion-scale production dataset collected from the real traffic of a search advertising system, together with the corresponding online A/B testing, demonstrate CIM's superior performance. Notably, CIM has been deployed in production at JD.com, serving the main traffic of hundreds of millions of users, which shows great application value. Our code and dataset are available at https://github.com/kaifuzheng/cim.

CCS CONCEPTS

• **Information systems** → **Sponsored search advertising**; **Computational advertising**; • **Applied computing** → *Online shopping*; • **Computing methodologies** → Neural networks.

KEYWORDS

Search Engine, Recommendation System, Online Advertising, Click-through Rate

ACM Reference Format:
Kaifu Zheng, Lu Wang, Yu Li, Xusong Chen, Hu Liu, Jing Lu, Xiwei Zhao, Changping Peng, Zhangang Lin, Jingping Shao. 2022. Implicit User Awareness Modeling via Candidate Items for CTR Prediction in Search Ads. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3485447.3511953

*Equal contribution. †Corresponding author.

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04.
https://doi.org/10.1145/3485447.3511953

1 INTRODUCTION

Online advertising, such as search ads and display ads, is one of the most important business model for search engines, e-commerce platforms, social networks, etc. Cost-per-click (CPC) or pay-per-click (PPC) is a widely-used online advertising model, in which advertisers pay for each user click on their sponsoring items. With this mechanism, it is necessary for the advertising platform to estimate the click-through rate (CTR) for each candidate item and then select one item to display in order to maximize the expected revenue. Personalized and accurate CTR prediction not only benefits the final revenue, but also improves user experience and satisfaction.

CTR prediction in online advertising is typically based on a statistical machine learning model trained with historical data of users' viewing and click behavior. Examples of such statistical models include logistic regression [11], tree-based models [22, 52], and neural networks [33, 47, 66]. When serving in a typical online advertising scenario, a CTR prediction model derives a set of candidate items. Each candidate item will be assigned a ranking score considering both the pCTR (predicted CTR) and the bid price. Items with higher ranking scores own higher a probability to be exposed to the user.

User click behavior usually showcases strong comparison patterns, especially among competing/relevant items within personalized awareness of the given user. For example, a sponsored item that is much more economical than its competing items (or enjoys certain other distinguishing highlights) is obviously more likely to be clicked; in contrast, it is difficult for a less distinctive item to attract users' attention, and hence unlikely to be clicked. Note that comparison patterns play an effect only if the distinguishing highlights are actually *within user awareness* — if the user does not know anything about the competing items, certainly there will be no meaningful comparison pattern. Therefore, the relative positioning of the target item, i.e., the item to be predicted, among its competing/relevant items within user awareness could be a strong pattern for CTR prediction. Figure 1(b) shows an intuitive example.

User behavior sequence modeling could be seen as a way to capture comparison patterns [4, 14, 46, 65, 66]. The motivation is clearly intuitive — there is a straightforward comparison between the target item and the items that a user has viewed/clicked before, which is naturally within user awareness; hence the sequential user behavior data could work as an explicit comparison *context* and have a profound impact on CTR. Despite being successful in practice, user behavior sequence modeling still has several inherent limitations. On the one hand, it often suffers from the *cold start*

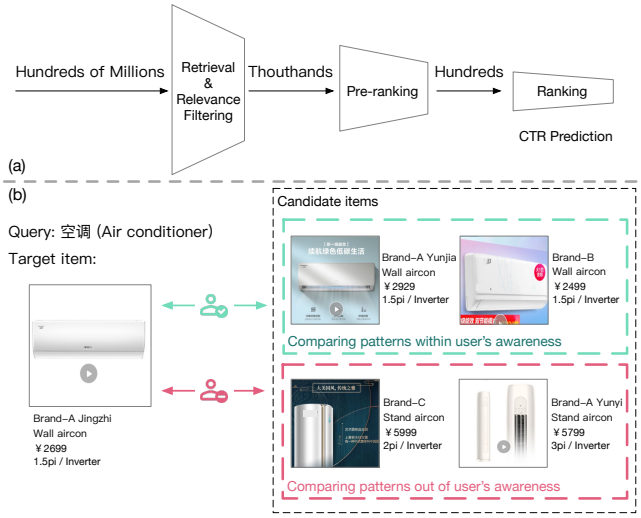

Figure 1: (a) A typical cascade architecture in an industrial search ads engine. Upstream retrieval and relevance filtering ensure that the candidate items are highly related to the query. (b) An intuitive illustration of our idea on comparing patterns and user awareness. When searching "air conditioner", all the candidate items are guaranteed to be air conditioners. For a target item "Brand-A Jingzhi", comparisons between the target item and the items with similar prices and attributes, which are more likely within user awareness, play an important role for click behavior.

problem in the new user/interest case [5, 42], where the user's behavior sequence does not contain any item competing/relevant with the target item — there is no meaningful comparison pattern in the user behavior sequence [55]. On the other hand, when the length of the user behavior sequence increases, most of the items in the sequence could be irrelevant with the target item — it is hard to precisely capture comparison patterns of the target item from the noisy sequence, i.e., the *behavior noise* problem [34, 37]. Furthermore, we notice that comparison patterns not necessarily stem from the context that users perceive directly on the online advertising platform. In fact, users could touch on competing/relevant items from a variety of channels, e.g., on other online platforms and even in offline scenarios. Since the online advertising platform probably has no access to these data, the CTR prediction model is not able to capture these comparison patterns from user behavior sequences. We refer to this problem as the *hidden channels* problem.

In this paper, to address the above challenges, we study modeling user awareness about competing/relevant items in an implicit manner by exploiting the business characteristics of search ads. In search ads scenarios, in order to ensure user experience, all of the candidate items for the CTR prediction model must be relevant to the query, which is usually guaranteed by an upstream relevance model [21, 39] filtering out all irrelevant items, as shown in Figure 1(a). Inspired by this observation, we propose to leverage these candidate items as surrogates for competing/relevant items about which the user could probably have prior knowledge; in other words, we take these candidate items as original materials for modeling implicit user awareness.

We propose CIM (Candidate Items Modeling, see Figure 2 for the overall architecture), a CTR prediction framework, to capture comparison patterns from candidate items and thus to implicitly model user awareness of competing/relevant items. To further characterize users' personalized awareness, we build the sub-module *Select* at the beginning of the prediction stage, which predicts the probability that the user is "aware" of each candidate item. Then, the CIE (Candidate Items Encoder) module encodes the set of candidate items, together with the outputs of *Select*, into a context vector. CIE, inspired by the Transformer architecture [54], enjoys permutation invariance with respect to candidate items, and characterizes cross-item interactions in order to effectively capture comparison patterns. Finally, the *Backbone* module outputs the predicted CTR based on the context vector. Notably, *Backbone* could be simply realized from retrofitting existing state-of-the-art CTR prediction models; in other words, our CIM framework could enhance a wide variety of CTR prediction models already in production, showing great application value. It is worth emphasizing that our proposed CIM characterize comparison patterns from a novel perspective, and is orthogonal to user behavior sequence modeling.

We conduct extensive experiments to validate CIM's effectiveness. It is noteworthy that CIM additionally requires sets of candidate items, whereas traditional CTR prediction datasets do not contain such context information. Instead, our experiments on public datasets are partially simulated from learning-to-rank datasets. Moreover, our experiments in real-world industrial scenarios are mainly based on a search ads system of one of the world's largest e-commerce platforms, JD.com. We collect a ten-billion-scale production dataset from the real traffic, which contains the set of candidate items for each request (exposed item). To facilitate further research, we also release a light-weight and anonymous version of the dataset, together with our code. Experimental results show that CIM could significantly and consistently improve performance of a variety of baseline CTR prediction models across all of the datasets.

Moreover, we have successfully deployed CIM in production for the aforementioned search ads system of JD.com. In our carefully designed large-scale online A/B testing, CIM brings up 1.81% improvement on CTR and 2.88% on RPM (Revenue Per Mille), respectively. Since then, CIM has been serving the main traffic.

In summary, our main contributions are as follows:

- To the best of our knowledge, we are the first to capture implicit comparison patterns within user awareness about competing/relevant items and the first to model candidate items directly for CTR prediction in search ads.
- We propose CIM to model the comparison patterns in an implicit manner by means of utilizing candidate items. It is a general framework, being able to improve performance of a wide variety of existing CTR prediction models.
- We release a large-scale industrial dataset that logs the set of candidate items for each exposed item, in order to facilitate further research in this novel direction.
- CIM has been successfully deployed in production serving the main traffic, showing great application value.

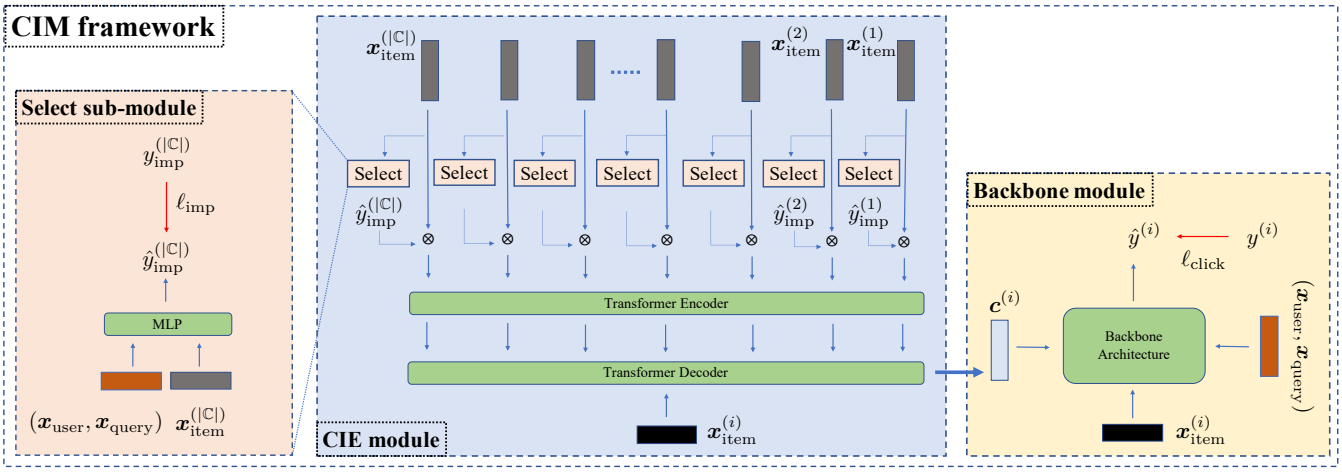

Figure 2: The proposed CIM framework consists of three components: (1) The *Select* sub-module (left) scores each candidate item indicating the probability the user is aware of each item; (2) The CIE (Candidate Items Encoder) module (middle), exploiting the inductive bias introduced by *Select*, encodes the set of candidate items into a context vector; (3) The *Backbone* module (right) outputs the predicted CTR based on the context vector.

2 RELATED WORK

In this section, we review previous studies on CTR prediction and learning-to-rank.

2.1 CTR prediction

CTR prediction is one of the crucial tasks in online advertising and recommendation systems. The accuracy of CTR prediction model impacts online performance greatly, which attracts tremendous attention from both academe and industry. Recent CTR prediction methods are usually designed following three typical paradigms: feature representation, feature interaction, and loss function.

Many works [10, 18, 33, 43, 52] attempt to improve feature representations of users or items by incorporating more information to the model. Among these works, user behavior modeling is one of the most important branches, which is closely related to our work. User behavior is highly personalized and reflects spatio–temporal user interest. Early works focus on learning representations of latent user interest using different neural network architectures such as CNN [50, 61], RNN [65], Transformer [17, 48] and etc. DIN [66] first introduces the attention mechanism to capture users' diverse ranges of interest on different target items. DIEN [65] and DRN [64] further introduce sequence modeling and reinforcement learning to describe user's drifting interest. KFAtt [34] proposes to use Kalman filtering to overcome bias of frequent behaviors. SIM [37] and UBR4CTR [40] model long-term user behavior. Besides user interest, user behavior sequences could also indicate user's awareness of relevant/competing items as discussed in Section 1. However, this information is usually not sufficient due to the cold start, behavior noise, and hidden channel problem. Instead, we propose to model user awareness via candidate items. Since the information introduced by candidate items is orthogonal to user behavior sequences, we gain extra performance lift on our complicated baseline in production which has been optimized many times and utilizes user behavior modeling and other SOTA techniques.

Methods exploring feature interactions, such as FM, Wide&Deep, Deep Crossing, Deep&Cross (DCN), DeepFM, xDeepFM, CrossMix [12, 20, 31, 43, 45, 57, 58], aim to memorize co-occurrence patterns in the union of the feature space and the label space. Since our proposed CIM is an independent feature extractor module (extract a context vector), it could be easily incorporated to any of the above models to improve the overall performance.

Pointwise classification loss such as the cross-entropy is most commonly used in CTR prediction. With the development of metric learning [28, 56] and learning-to-rank, pairwise loss [8, 25] is introduced to CTR prediction [49], which employs the relative ranking supervision. It could be naturally used to capture comparison patterns between positive and negative instances during training, by establishing auxiliary pairwise loss between the positive instance and a negative instance. We show that our proposed method is also compatible with pairwise loss in our experiments (see Section 5.4).

In this paper, we propose to implicitly model user awareness from the candidate item set by predicting their impression probabilities. To the best of our knowledge, we are the first to model candidate items in the CTR prediction task. Furthermore, as discussed above, our method is orthogonal to almost all current SOTA CTR prediction methods, and introduces performance lift by simple plug and play.

2.2 Learning to rank

Another closely related task in search ads scenarios is learning-to-rank (LTR). Learning-to-rank algorithms aim to apply machine learning techniques to solve ranking problems [2, 6, 8, 9, 18, 25, 60], which has been applied to various ranking tasks, such as web search [24], recommendation [15], e-commerce search [26]. Most traditional LTR approaches follow the univariate scoring paradigm to compute the ranking score for each item independently based on their feature vectors. Context-aware LTR models are proposed to rank items based on their context (i.e., other items in the candidate set), which take multiple items together as input and jointly predict their final ranking scores [2, 3, 35, 38]. For instance, SetRank [35] is

proposed to capture local context information in a query by modeling cross-item interactions. It has been shown that CTR prediction methods could be easily applied to LTR, e.g., by taking the predicted CTR as the ranking score [49]. Accordingly, our proposed CTR prediction method CIM could also be easily extended to context-aware LTR with slight modifications. Our experiments show the superiority of CIM over the SOTA context-aware LTR method SetRank, which further indicates the generality of CIM (see Section 5.5).

3 PRELIMINARIES

A CTR prediction model could be formalized as a function $f_\theta : \mathbb{X} \to [0, 1]$ parameterized by $\theta \in \Theta$ where $\mathbb{X}$ denotes the input space and Θ is the parameter space of the model.

In the scenarios of search ads, the input space often encompasses features from users, items, queries and maybe other related context, i.e., $\mathbb{X} = \mathbb{X}_{\text{user}} \times \mathbb{X}_{\text{query}} \times \mathbb{X}_{\text{item}} \times \mathbb{X}_{\text{context}}$. When serving online, the CTR prediction model will be given a request $(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}) \in \mathbb{X}_{\text{user}} \times \mathbb{X}_{\text{query}}$, a list of candidate items $\mathbb{C} = \{\boldsymbol{x}^{(i)}_{\text{item}}\}_{i=1}^{|\mathbb{C}|}$ where $\boldsymbol{x}^{(i)}_{\text{item}} \in \mathbb{X}_{\text{item}}$, and context features $\boldsymbol{x}_{\text{context}} \in \mathbb{X}_{\text{context}}$ (optional, in other words, we might have $\mathbb{X}_{\text{context}} = \emptyset$). Then the model is utilized to predict CTR for each instance,

$$\hat{y}^{(i)} = f_\theta(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}, \boldsymbol{x}^{(i)}_{\text{item}}, \boldsymbol{x}_{\text{context}}),\ \forall i \in [|\mathbb{C}|], \tag{1}$$

and the item with the highest predicted CTR will be chosen and exposed to the user.[1]

Training sets of CTR prediction models are constructed from logs for impressions, i.e, exposed items and corresponding click feedback. Let $\mathbb{D} \in (\mathbb{X} \times \{0, 1\})^*$ denote the training set; for any $(\boldsymbol{x}, y) \in \mathbb{D}$, the label y indicates whether the instance $\boldsymbol{x}$ is clicked. CTR prediction models are trained by solving the following optimization problem:

$$\min_{\theta \in \Theta} \left\{ \mathcal{L}(\mathbb{D}; \theta) \doteq \frac{1}{|\mathbb{D}|} \sum_{(\boldsymbol{x}, y) \in \mathbb{D}} \ell(y, f_\theta(\boldsymbol{x})) \right\}, \tag{2}$$

where $\ell : \{0, 1\} \times [0, 1] \to \mathbb{R}$ is a loss function.

4 PROPOSED ALGORITHMS: CIM

As we emphasize in Section 1, user click behavior shows strong comparison patterns among competing items; moreover, candidate items could reflect the feature distribution of competing items since in search ads scenarios, all of the candidate items are relevant to the query, usually guaranteed by an upstream relevance filter.

In order to capture the feature distribution of competing items within user awareness contained in the set of candidate items, we propose a novel CTR prediction framework — CIM (Candidate Items Modeling), which models the candidate item set directly.

4.1 Overview of CIM

Our proposed CIM framework takes the set of candidate items as additional context features for the CTR prediction model, i.e., $\mathbb{X}_{\text{cand}} = \mathbb{X}^{|\mathbb{C}|}_{\text{item}}$ and $\mathbb{X} = \mathbb{X}_{\text{user}} \times \mathbb{X}_{\text{query}} \times \mathbb{X}_{\text{item}} \times \mathbb{X}_{\text{cand}}$.[2] When serving online, the model predicts CTR for each instance $\boldsymbol{x}^{(i)}_{\text{item}}$, i.e., the target item, in the set of candidate items as

$$\hat{y}^{(i)} = f_\theta(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}, \boldsymbol{x}^{(i)}_{\text{item}}, \mathbb{C}),\ \forall i \in [|\mathbb{C}|]. \tag{3}$$

The overall architecture of our proposed CIM framework is shown in Figure 2. This framework mainly consists of three components — the *Select* sub-module, the CIE (Candidate Items Encoder) module and the *Backbone* module. The CIE module encodes the candidate items $\mathbb{C}$ along with the target item $\boldsymbol{x}^{(i)}_{\text{item}}$ to a context vector $\boldsymbol{c}^{(i)}$, while the *Select* sub-module introduces an inductive bias to further capture user awareness during this encoding process. The *Backbone* module could be simply realized from retrofitting a wide range of existing deep neural networks-based CTR prediction models. Besides the input of the vanilla CTR prediction model, i.e., $\boldsymbol{x}_{\text{user}}$, $\boldsymbol{x}_{\text{query}}$ and $\boldsymbol{x}^{(i)}_{\text{item}}$, *Backbone* also takes the context vector $\boldsymbol{c}^{(i)}$ as an additional input; it then outputs the predicted CTR, i.e., $\hat{y}^{(i)}$.

4.2 Sub-Module: *Select*

Select is a sub-module of CIE. It is used to select the items from the candidate set that the user could be probably aware of, and to introduce the inductive bias for the CIE module that these items play a more important role when modeling the implicit context contained in the candidate items. Specifically, *Select* computes an impression possibility, or selection score, $\hat{y}^{(j)}_{\text{imp}}$ for each candidate item $\boldsymbol{x}^{(j)}_{\text{item}}$ given the request $(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}})$:

$$\hat{y}^{(j)}_{\text{imp}} = \text{Select}(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}, \boldsymbol{x}^{(j)}_{\text{item}}),\ \forall j \in [|\mathbb{C}|], \tag{4}$$

where $\hat{y}^{(j)}_{\text{imp}} \in (0, 1)$. The impression probability is then utilized by the CIE module to balance weights of candidate items when computing the context vector.

Select could be a simple multi-layer perceptron with ReLU as the activation function, and the output is scaled to $(0, 1)$ using a standard logistic sigmoid function.

In order to ensure that *Select* acts as an impression predictor according to our requirements, we leverage impressions in the whole search session as auxiliary supervision. The corresponding loss function $\ell_{\text{imp}}(\cdot, \cdot)$ on $\hat{y}^{(i)}_{\text{imp}}$ will be introduced in Section 4.5.

4.3 Module: CIE

The CIE (Candidate Items Encoder) module is the core component of our CIM framework. It encodes the set of candidate items $\mathbb{C}$ into a context vector for each candidate item $\boldsymbol{x}^{(i)}_{\text{item}}$ (the target item for the downstream modules):

$$\boldsymbol{c}^{(i)} = \text{CIE}(\boldsymbol{x}^{(i)}_{\text{item}}, \mathbb{C}, \boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}),\ \forall i \in [|\mathbb{C}|], \tag{5}$$

where $\boldsymbol{x}_{\text{user}}$ and $\boldsymbol{x}_{\text{query}}$ are only used by the *Select* sub-module.

Since CIE aims to encode the *set* of candidate items, it is a reasonable inductive bias that the output of CIE (the context vector) should not be changed with the order of the candidate items; in other words, the context vector should be permutation invariant with respect to candidate items. Moreover, CIE should also characterize cross-item interactions among candidate items, especially the interactions between the target item and the other candidate items in order to capture comparison patterns.

[1] For simplicity, we assume all of the items have the same bid price.
[2] We omit all of the other context features for clarity.

Inspired by recent progress on learning-on-sets [62, 63] and the self-attention mechanism [13, 32, 36], we build CIE on the basis of the architecture of Transformer [54]. Specifically, the Transformer encoder derives the weighted embeddings of all the candidate items, of which the weights are the selection scores returned by *Select*; the output is then passed to the Transformer decoder. Together with the target item $\boldsymbol{x}_{\text{item}}^{(i)}$, the Transformer decoder outputs the final context vector $\boldsymbol{c}^{(i)}$.

4.4 Module: *Backbone*

The *Backbone* module is the essential component to predict CTR. It takes the request $(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}})$ and one candidate item $\boldsymbol{x}_{\text{item}}^{(i)}$ along with the corresponding context vector $\boldsymbol{c}^{(i)}$ returned by CIE as the input, and outputs the resulting CTR prediction $\hat{y}^{(i)}$:

$$\hat{y}^{(i)} = \text{Backbone}(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}, \boldsymbol{x}_{\text{item}}^{(i)}, \boldsymbol{c}^{(i)}),\ \forall i \in [|\mathbb{C}|]. \tag{6}$$

The *Backbone* module could be realized from retrofitting existing state-of-the-art CTR prediction models [20, 31, 57]. In our implementation, we simply concatenate $\boldsymbol{c}^{(i)}$ with the output of the penultimate layer of the vanilla CTR prediction model, and its last layer (usually a fully connected layer) is adjusted accordingly to match the dimensionality of the concatenated vector.

4.5 Objective function

CIM's supervision signals come from two sources — clicks (whether the exposed item is clicked) and impressions (whether the candidate item is exposed). The click supervision encourages accurate predictions for CTR, and the impression supervision instructs the *Select* sub-module to select important candidate items which are likely to be exposed to the user (within user awareness).

Given a logged instance $(\boldsymbol{x}, y) \in \mathbb{X} \times \{0, 1\}$ where

$$\boldsymbol{x} = (\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}, \boldsymbol{x}_{\text{item}}, \mathbb{C}), \tag{7}$$

the output of our CIM model on this instance is

$$\hat{y} = f_\theta(\boldsymbol{x}_{\text{user}}, \boldsymbol{x}_{\text{query}}, \boldsymbol{x}_{\text{item}}, \mathbb{C}); \tag{8}$$

the loss on this instance is defined as

$$\mathcal{L}_{\text{ins}}(\boldsymbol{x}, y; \theta) \doteq (1-\eta)\ell_{\text{click}}(y, \hat{y}) + \frac{\eta}{|\mathbb{C}|}\sum_{j=1}^{|\mathbb{C}|}\ell_{\text{imp}}(y_{\text{imp}}^{(j)}, \hat{y}_{\text{imp}}^{(j)}), \tag{9}$$

where $y_{\text{imp}}^{(j)}$ indicates whether $\boldsymbol{x}_{\text{item}}^{(j)}$ is exposed to the user, and $\eta \in [0, 1]$ is a hyper-parameter to balance the two losses.

In our implementation, we utilize the cross-entropy loss on both the click and impression supervision, i.e., both $\ell_{\text{click}}(\cdot, \cdot)$ and $\ell_{\text{imp}}(\cdot, \cdot)$ have the same form $\ell(\cdot, \cdot)$ as

$$\ell(y, y') \doteq -y\log(y') - (1-y)\log(1-y'), \tag{10}$$

where $y \in \{0, 1\}$ and $y' \in (0, 1)$.

Finally, given a training set $\mathbb{D} \in (\mathbb{X} \times \{0, 1\})^*$, our CIM model is trained by solving the following optimization problem in an end-to-end approach:

$$\min_{\theta \in \Theta}\left\{\mathcal{L}_{\text{set}}(\mathbb{D}; \theta) \doteq \frac{1}{|\mathbb{D}|}\sum_{(\boldsymbol{x}, y)\in\mathbb{D}}\mathcal{L}_{\text{ins}}(\boldsymbol{x}, y; \theta)\right\}. \tag{11}$$

In other words, we jointly train all of the modules of CIM.

Table 1: Statistics of the CandiCTR-Pub and CandiCTR-Pro dataset. #AvgNum is calculated by dividing the #Instances by the #User-query Groups. Bil are short for Billion.

Property	CandiCTR-Pub		CandiCTR-Pro	
	train	test	train	test
#Instances	5,189,525	1,118,420	10Bil	1M
#User-query Groups	1,014,182	223,449	2Bil	0.2M
#Queries	323,478	64,323	20M	30k
#Users	931,883	180,146	0.2Bil	100k
#Items	15,587,268	7,135,054	30M	1M
#Categories	3,335	2,745	4k	2k
#Brands	88,219	80,567	100k	20k
#Vendors	103,832	98,715	100k	20k
#AvgNum	5.117	5.005	4.64	4.58

5 EXPERIMENTS

In this section, we would like to answer the following questions:

(1) How does CIM reinforce different CTR prediction baselines on different datasets?
(2) How does each module of CIM perform?
(3) How good is the generality ability of CIM?

For fair comparison, all of the comparing methods are implemented with TensorFlow[3] and run on an NVIDIA Tesla P40 GPU. All categorical features are turned into dimension-16 embeddings. Both encoder and decoder are 1-layer transformer with dimension 80, #heads 4. We set the hyper-parameter $\eta = 0.5$. The Adam [27] optimizer is adopted for training, of which the learning rate is 0.001.

5.1 Datasets

Since public available CTR prediction datasets such as Criteo[4] and Avazu[5] do not contain the candidate item set for each instance, it is not feasible for us to conduct experiments on these benchmarks. Therefore, we collect two real-world industrial datasets, termed CandiCTR-Pub and CandiCTR-Pro, from the main traffic of a sponsored search advertising system of one of the world's largest e-commerce platform, JD.com. CandiCTR-Pub is a lightweight and anonymous version of CandiCTR-Pro, and we make it public available to facilitate further academic research. We conduct our main offline experiments and ablation studies on CandiCTR-Pub; we also show offline results on CandiCTR-Pro and its corresponding online A/B testing results. Furthermore, we carefully construct a dataset MSLR10K-Candi from the public learning-to-rank dataset MSLR10K [41] to simulate our application scenario. Due to page limitation, please refer to Appendix A for details of MSLR10K-Candi and its experimental results.

CandiCTR-Pub. Due to the limitation of public available datasets, we build a dataset with search logs from a search advertising system of one of the largest e-commerce platform, termed CandiCTR-Pub. We randomly sample a subset from the online search logs, and split the training and test set by log time to ensure the data distribution

[3] https://www.tensorflow.org/
[4] http://labs.criteo.com/downloads/2014-kaggle-display-advertising-challenge-dataset
[5] https://www.kaggle.com/c/avazu-ctr-prediction

Table 2: Experimental results on CandiCTR-Pub dataset. Each experiment is repeated for 5 times independently (mean±std). CI indicates directly incorporating candidate items via Transformer without the *Select* sub-module; SM indicates containing the *Select* sub-module. ↑ means the larger number is better; ↓ means the smaller number is better.

No.	baseline	CI	SM	AUC↑	ΔAUC	LogLoss↓	ΔLogLoss	GAUC↑	ΔGAUC
(1)	DNN			0.6176±0.0003		0.2554±0.0000		0.5300±0.0005	
(2)	DNN	✓		0.6220±0.0005	+0.0044	0.2547±0.0001	-0.0006	0.5339±0.0004	+0.0039
(3)	DNN	✓	✓	**0.6349±0.0007**	**+0.0173**	**0.2544±0.0001**	**-0.0009**	**0.5500±0.0004**	**+0.0200**
(4)	xDeepFM			0.6193±0.0001		0.2551±0.0000		0.5318±0.0002	
(5)	xDeepFM	✓		0.6215±0.0003	+0.0022	0.2548±0.0001	-0.0003	0.5338±0.0006	+0.0020
(6)	xDeepFM	✓	✓	**0.6360±0.0002**	**+0.0168**	**0.2544±0.0001**	**-0.0007**	**0.5499±0.0001**	**+0.0181**
(7)	DCN			0.6197±0.0002		0.2547±0.0000		0.5313±0.0004	
(8)	DCN	✓		0.6240±0.0002	+0.0043	0.2544±0.0000	-0.0002	0.5358±0.0003	+0.0045
(9)	DCN	✓	✓	**0.6352±0.0006**	**+0.0155**	**0.2543±0.0001**	**-0.0004**	**0.5496±0.0003**	**+0.0183**
(10)	CrossMix			0.6178±0.0001		0.2553±0.0000		0.5297±0.0006	
(11)	CrossMix	✓		0.6240±0.0003	+0.0062	0.2547±0.0000	-0.0006	0.5351±0.0004	+0.0054
(12)	CrossMix	✓	✓	**0.6355±0.0001**	**+0.0177**	**0.2543±0.0000**	**-0.0010**	**0.5503±0.0003**	**+0.0206**

is similar to that of the online environment. Every instance in the dataset consists of a user, a query, a target item, the click label, the corresponding candidate items, and labels indicating whether the candidate items have been exposed. The length of each candidate list is around 300. Each item is associated with the corresponding <brand ID, category ID, vendor ID>. All of the items, users, and queries are represented with unique and anonymous IDs for privacy protection. Our dataset is publicly available at https://github.com/kaifuzheng/cim. Statistics are shown in the middle area of Table 1.

CandiCTR-Pro. The real production dataset CandiCTR-Pro is a full version of CandiCTR-Pub. We use full logs in the first 32 days for training and sample 1 million instances from the following day for testing. We omit the detailed descriptions since it is similar to the CandiCTR-Pub dataset. Statistics are shown in the right area of Table 1. The volume of our CandiCTR-Pro is about 2,000 times larger than CandiCTR-Pub, which brings more challenges.

5.2 Evaluation metrics

For offline evaluation, we employ AUC, LogLoss, and GAUC as our metrics. **AUC** (Area Under the ROC Curve) [16] is the most important offline evaluation metric for CTR prediction that measures the global pairwise accuracy, and is consistent with the online performance in most cases. In addition, we group instances by their queries and user IDs, and use **GAUC** (Group AUC) to evaluate the overall performance of local pairwise accuracy in each group:

$$\text{GAUC} = \frac{\sum_{g=1}^{G} \text{Length}_g \times \text{AUC}_g}{\sum_{g=1}^{G} \text{Length}_g}, \tag{12}$$

where G is the number of groups, Length_g and AUC_g are the number of instances and AUC in the g-th group, respectively. Both AUC and GAUC are strong patterns reflecting the online performance. **Logloss** defined by Equation 10 is also reported to indicate the classification performance of the model.

For online A/B testing, **CTR** (Click-Through Rate) and **RPM** (Revenue Per Mille) are reported to show the effectiveness of our method in real-world systems (online performance).

5.3 Offline results on multiple baselines

Since our proposed CIM modules are plug-and-play, we conduct our experiments on 3 classical baselines: DNN, xDeepFM [31], and DCN [57], and a recent state-of-the-art method CrossMix [58]. The detailed network architectures are as follows:

- DNN: a 3-layer MLP (Multi Layer Perception) of size 256-128-1 with ReLU.
- xDeepFM: the *deep* part is the same as DNN, and the size of the *CIN* part is {128, 128}.
- DCN: 3 layers, and the *deep* part is the same as DNN.
- CrossMix: the number of experts is 3, the depth of the cross layers is 3, the rank of the low-rank cross layer is 16, and the *deep* part is the same as DNN.

Note that for user privacy protection, user behavior information is not contained in CandiCTR-Pub. Therefore, several popular methods like DIN [66], DIEN [65], and DRN [64] that model user behaviors are not feasible for CandiCTR-Pub. Nonetheless, we show in Section 5.4 that the improvement introduced by our proposed CIM is actually *compatible* with user behavior modeling in our online A/B testing experiments where the baseline model has been highly optimized for many times and leverage a variety of techniques to model user behaviors carefully.

Main Results. Results are shown in Table 2. "CI" indicates incorporating candidate items directly via Transformer *without* the *Select* sub-module; "SM" indicates the whole CIM framework containing the *Select* sub-module. To show the significance of our improvement, we conduct t-test and repeat each experiment for 5 times independently. The results are shown in the format of mean±std. Comparing the baseline models (1)(4)(7)(10) with their CIM counterparts (3)(6)(9)(12), we obtain significant and consistent performance gain across all of the baselines and all of the metrics with p-value $p < 0.0001$. AUC increases by 1.6%∼ 1.8%, and GAUC increases by 1.6%∼2.1%, indicating that ranking performance is highly improved by introducing CIM. Meanwhile, the classification LogLoss decreases 0.04%∼0.08%, suggesting the model converges better with

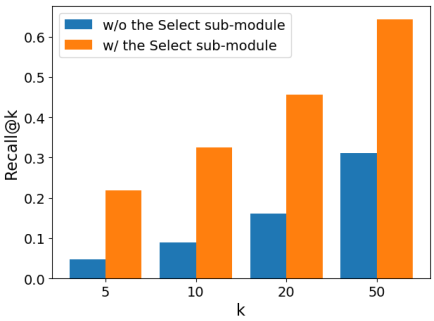

Figure 3: Performance comparison of selecting exposed items from the candidate set by the attention weights of the CIE module between w/ and w/o the *Select* sub-module. Size of candidate set ≈ 300.

the implicit comparison patterns provided by modeling candidate items. The results verify the effectiveness of our proposed CIM.

Effect of incorporating candidate items. As stated in Section 1, candidate items reflect user awareness under the current query. Comparing Model (2)(5)(8)(11) with their baselines (1)(4)(7)(10), it could be seen that although AUC and LogLoss are improved, the improvement is very limited compared with models that also incorporate the *Select* sub-module; and the auxiliary metric GAUC fluctuates slightly. One possible reason is that with lengthy and noisy candidate item lists, it is difficult to capture the user awareness about competing items without the help of other supervision. Hence, we conclude: (1) introducing candidate items is helpful to CTR prediction; (2) direct modeling candidate items by attention mechanism could not capture comparison patterns sufficiently.

Effect of the Select sub-module. In order to further characterize users' personalized awareness, the *Select* sub-module is proposed to select items about which the current user could probably have prior knowledge, and then the CIE module would pay more attention to them when generating the context vector for CTR prediction. Comparing Model (3)(6)(9)(12) with Model (2)(5)(8)(11), we observe that the *Select* sub-module plays an important role in candidate items modeling — it brings performance gain on all of the metrics. The experimental results validate the importance of *Select*.

How does the Select sub-module work? We conduct additional experiments to understand how the proposed *Select* sub-module works, of which the results are shown in Figure 3. Since the attention weight of the CIE module can be regarded as the contribution of each candidate item to the final contextual representation, we use Recall@k to measure how many exposed items are assigned with the top-k highest attention weights. With the help of the *Select* sub-module, obviously more exposed items are assigned with the top-k highest attention weights, which further confirms the effectiveness of the *Select* sub-module.

Intuitive examples on comparison patterns. To better understand comparison patterns learned by the CIE module, we visualize 3 examples in our dataset in Figure 4. Note that comparison patterns between the target item and candidate items could be characterized by the attention weights of the CIE decoder; therefore, we sort candidate items by the attention weights of the CIE decoder for visualization. Items in green boxes are top-scored, while items in red boxes are ranked at the bottom. Firstly, as we can see, candidate items are closely related to the query. Secondly, items with high scores are usually competitive and comparable to the target item in the aspect of the brand, price range, or attributes. Thirdly, low-scored items are obviously less related to the target item, and tend to be long-tailed items that are not frequently exposed to the public or are not commonly perceived by the public. More specifically, consider the following example. When we search "mobile phone" on an e-commerce platform, and see the item "Brand-A Xperial1", our click behavior is less likely to be affected by a "Brand-E N1" dumbphone, but is more likely to be influenced by a competitive item such as "Brand-B Ultra". This visualization suggests that the CIE module has actually learned valuable comparison patterns.

Table 3: Experiment on CandiCTR-Pro dataset. The model is trained on CandiCTR-Pro, and deployed online for A/B test. Offline results are evaluated on CandiCTR-Pro test set.

Metrics		Base	+CIM	+CIM	+CIM
Data split		All	All	CS	LB
Offline	AUC	0.7631	0.7677		
	ΔAUC	0	+0.0046		
Online A/B test	ΔCTR	0	+1.81%	+2.20%	+2.10%
	ΔRPM	0	+2.88%	+3.12%	+2.62%

5.4 Online A/B testing

We further conduct experiments to validate the effectiveness of CIM in the real-world search advertising system, and the experimental results are shown in Table 3. rIn the advertising system, a previous BaseModel was serving the main traffic of hundreds of millions of active users. Our BaseModel is very strong and has been highly optimized. It consists of complicated user behavior modeling modules, incorporates a wide variety of other features, and also takes pairwise loss as an auxiliary loss function.

We train the model on our 10 billion-scale CandiCTR-Pro dataset. To show the correlation between AUC and online metrics, we report both offline results on the CandiCTR-Pro test set and results of online A/B testing, which splits 95% and 5% flow of the main traffic respectively. This A/B test lasted for 3 days; 23.8 million requests, 6.9 million users, and 4.8 million items in total are involved in the 5% flow of Group-B (experimental group).

First, from the offline experimental results on the CandiCTR-Pro test set, we can observe that incorporating CIM (*+CIM*) outperforms the *Base* model by +0.46 percentage points of AUC. It is worth noting that +0.1 percentage points of AUC is a significant improvement, which further validates the effectiveness of CIM.

Second, owing to CIM's simplicity, we have successfully deployed this CIM-enhanced Base model in the search advertising system. In our online A/B test, *+CIM* achieves 1.81% CTR, and 2.88% RPM gain. We would like to emphasize again that this improvement is nontrivial since our *Base* model has already been highly optimized. Since then, CIM has been serving the main traffic of hundreds of millions of users.

Furthermore, besides overall results on all users, we analyze our performance on cold-start users (CS, users without behavior

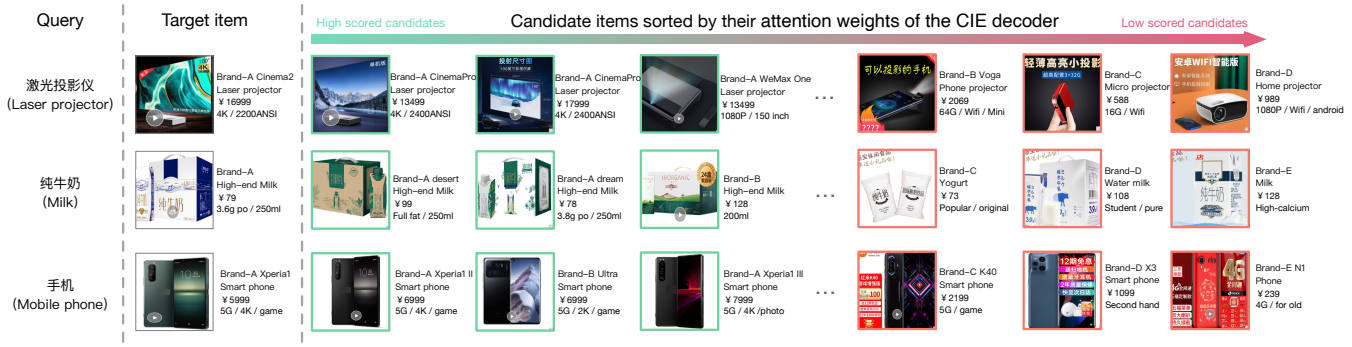

Figure 4: Intuitive examples of CIM on comparison patterns. Each row corresponds to an instance in our dataset, including the query, target item, and candidate items sorted by their attention weights of the decoder in the CIE module. We show the first 3 and the last 3 items respectively in green and red boxes. The text box on the right of each item picture shows the corresponding information about the item. The 4 rows represent brand&model type, category, price, and attributes, respectively.

in the last 30 days) and users with top 20% long behaviors (LB). CIM introduces extra performance gain for cold-start users for which user behavior modeling plays no effect: CTR +2.20% and RPM +3.12%. For long behavior users, even though the performance of the user behavior modeling baseline is already 7.8% higher (CTR) than the one on all users, we still further obtain 2.10% and 2.62% improvement on CTR and RPM respectively with CIM. These results show that our method is orthogonal to user behavior modeling.

5.5 Extension to LTR

Besides CTR prediction, our proposed CIM could also be easily extended to learning-to-rank by simply taking the pCTR as ranking score. Our CandiCTR-Pub dataset is also able to serve the learning-to-rank scenario by simply grouping instances by sessions, i.e., one search request. Thus, we design a set of experiments on CandiCTR-Pub for learning-to-rank, and evaluate on the commonly-used LTR metric NDCG (Normalized Discounted Cumulative Gain) [23].

A uni-variant neural LTR method that predicts the ranking score for each candidate independently is constructed as our base model (BaseModel); we enhance BaseModel with CIM (*Ours*). To show the superiority of the manner in which CIM utilizes candidate lists, we further compare our method with a recent context-aware LTR method SetRank [35], which also takes the candidate list into consideration by employing a stack of multi-head self-attention blocks. All of these three models are trained with an additional pairwise loss. Experimental results are shown in Table 4. Comparing *BaseModel* with *Ours*, all of the metrics are improved significantly. Meanwhile, *Ours* surpasses *SetRank* [35] remarkably. The experimental results further demonstrate the importance of modeling implicit user awareness and show generality ability of CIM.

Table 4: Experimental results under Learning-to-rank setting on CandiCTR-Pub dataset. ↑ indicates the larger number is better.

	NDCG@1↑	NDCG@5↑	NDCG@10↑
BaseModel	0.1689	0.3146	0.3822
SetRank [35]	0.1721	0.3172	0.3865
Ours	**0.1795**	**0.3245**	**0.3931**

6 DIVERGENT THINKING

In this section, we explain our proposed CIM from another two viewpoints, semi-supervised learning and inverse propensity scoring, expecting to introduce some divergent thinking to the community.

Semi-supervised learning. Semi-supervised learning (SSL) [53] is a branch of weakly supervised learning [67], aiming to utilize a large amount of unlabeled data along with a small amount of labeled data. Broadly speaking, our proposed CIM could also be seen as a SSL method. Nevertheless, it is noteworthy that SSL usually leverages unlabeled data during *training*, for which unlabeled data and labeled data are both at the instance level, e.g., self-training [30, 51], co-training [59, 68]. Despite the fact that CIM also employs unlabeled data, it works in a different and task-specific way. Specifically, for each instance, CIM takes a small portion of unlabeled data (i.e., candidate items) as auxiliary context features, and hence these unlabeled data are required during both training and inference. In other words, labeled data remain at the instance level, whereas unlabeled data are at the feature level.

Inverse propensity scoring. Recommender systems, including online advertising, are usually faced with the popularity bias issue — unpopular items do not derive the deserved impression [1, 7, 47]. To address popularity bias, inverse propensity scoring-based methods (IPS) [19, 29, 44], also known as propensity-based unbiased learning, could utilize candidate items to compute propensities (analogous to our selection scores) for the exposed items; the propensities are then used to re-weight each exposed item during training. Similar with semi-supervised learning, candidate items for IPS do not work during inference, and are not at the feature level as well.

7 CONCLUSION

In this paper, we propose a novel enhancement method for CTR prediction in search ads scenarios. Our proposed CIM (Candidate Items Modeling) characterizes users' implicit awareness on candidate items, and then encodes the candidate items into an additional context vector for the CTR prediction model. CIM is a general framework, being able to enhance a wide variety of state-of-the-art CTR prediction models. Owing to its simplicity and generality, we have successfully deployed CIM in production in a search advertising system at JD.com, achieving significant online improvement. The idea of implicitly modeling comparison patterns within awareness has great potential to extend to other learning problems.

REFERENCES

[1] Himan Abdollahpouri. 2019. Popularity Bias in Ranking and Recommendation. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 529–530.

[2] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *SIGIR*. 135–144.

[3] Qingyao Ai, Xuanhui Wang, Sebastian Bruch, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2019. Learning groupwise multivariate scoring functions using deep neural networks. In *SIGIR*. 85–92.

[4] Ismail Badache. 2019. Exploring differences in the impact of users' traces on Arabic and english facebook search. In *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 225–232.

[5] JesúS Bobadilla, Fernando Ortega, Antonio Hernando, and JesúS Bernal. 2012. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge Based Systems* 26 (2012), 225–238.

[6] Sebastian Bruch, Shuguang Han, Michael Bendersky, and Marc Najork. 2020. A stochastic treatment of learning to rank scoring functions. In *WSDM*. 61–69.

[7] Erik Brynjolfsson, Yu Jeffrey Hu, and Michael D. Smith. 2006. From Niches to Riches: Anatomy of the Long Tail. *Social Science Research Network* (2006).

[8] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *ICML*. 89–96.

[9] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *ICML*. 129–136.

[10] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2014. Simple and scalable response prediction for display advertising. *TIST* 5, 4 (2014), 1–34.

[11] Haibin Cheng and Erick Cantú-Paz. 2010. Personalized click prediction in sponsored search. In *WSDM*. 351–360.

[12] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[13] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long Short-Term Memory-Networks for Machine Reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 551–561.

[14] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *RecSys*. 191–198.

[15] Yajuan Duan, Long Jiang, Tao Qin, Ming Zhou, and Heung Yeung Shum. 2010. An empirical study on learning to rank of tweets. In *COLING*. 295–303.

[16] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[17] Yufei Feng, Fuyu Lv, Weichen Shen, Menghan Wang, Fei Sun, Yu Zhu, and Keping Yang. 2019. Deep session interest network for click-through rate prediction. *arXiv preprint arXiv:1905.06482* (2019).

[18] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[19] Alois Gruson, Praveen Chandar, Christophe Charbuillet, James McInerney, Samantha Hansen, Damien Tardieu, and Ben Carterette. 2019. Offline evaluation to make decisions about playlistrecommendation algorithms. In *WSDM*. 420–428.

[20] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI*. 2782–2788.

[21] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *CIKM*. 55–64.

[22] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *ADKDD*. 1–9.

[23] Kalervo Järvelin and Jaana Kekäläinen. 2017. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, Vol. 51. 243–250.

[24] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *KDD*. 133–142.

[25] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *KDD*. 217–226.

[26] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On application of learning to rank for e-commerce search. In *SIGIR*. 475–484.

[27] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *ICLR*.

[28] Brian Kulis. 2013. *Metric Learning: A Survey*.

[29] John Langford, Lihong Li, and Miroslav Dud k. 2011. Doubly Robust Policy Evaluation and Learning. In *ICML*. 1097–1104.

[30] Dong-Hyun Lee et al. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, Vol. 3. 896.

[31] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. XDeepFM: Combining explicit and implicit feature interactions for recommender systems. In *KDD*. 1754–1763.

[32] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A Structured Self-Attentive Sentence Embedding.. In *ICLR*.

[33] Hu Liu, Jing Lu, Hao Yang, Xiwei Zhao, Sulong Xu, Hao Peng, Zehua Zhang, Wenjie Niu, Xiaokun Zhu, Yongjun Bao, et al. 2020. Category-Specific CNN for Visual-aware CTR Prediction at JD. com. In *KDD*. 2686–2696.

[34] Hu Liu, Jing Lu, Xiwei Zhao, Sulong Xu, Hao Peng, Yutong Liu, Zehua Zhang, Jian Li, Junsheng Jin, Yongjun Bao, and Weipeng Yan. 2020. Kalman Filtering Attention for User Behavior Modeling in CTR Prediction. In *NeurIPS*.

[35] Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. 2020. SetRank: Learning a permutation-invariant ranking model for information retrieval. In *SIGIR*. 499–508.

[36] Ankur P. Parikh, Oscar Tackstrom, Dipanjan Das, and Jakob Uszkoreit. 2016. A Decomposable Attention Model for Natural Language Inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2249–2255.

[37] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based User Interest Modeling with Lifelong Sequential Behavior Data for Click-Through Rate Prediction. In *CIKM*. 2685–2692.

[38] Przemysław Pobrotyn, Tomasz Bartczak, Mikołaj Synowiec, Radosław Białobrzeski, and Jarosław Bojar. 2020. Context-aware learning to rank with self-attention. *arXiv preprint arXiv:2005.10084* (2020).

[39] Prafull Prakash, Julian Killingback, and Hamed Zamani. 2021. Learning Robust Dense Retrieval Models from Incomplete Relevance Labels. In *SIGIR*. 1728–1732.

[40] Jiarui Qin, Weinan Zhang, Xin Wu, Jiarui Jin, Yuchen Fang, and Yong Yu. 2020. User behavior retrieval for click-through rate prediction. In *SIGIR*. 2347–2356.

[41] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013).

[42] Al Mamunur Rashid, George Karypis, and John Riedl. 2008. Learning preferences of new users in recommender systems: an information theoretic approach. *SIGKDD Explorations* 10, 2 (2008), 90–100.

[43] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE ICDM*. IEEE, 995–1000.

[44] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as treatments: debiasing learning and evaluation. In *ICML*, Vol. 48. 1670–1679.

[45] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *KDD*. 255–262.

[46] Yang Song, Ali Mamdouh Elkahky, and Xiaodong He. 2016. Multi-Rate Deep Learning for Temporal Recommendation. In *SIGIR*. 909–912.

[47] Yuhai Song, Lu Wang, Haoming Dang, Weiwei Zhou, Jing Guan, Xiwei Zhao, Changping Peng, Yongjun Bao, and Jingping Shao. 2021. Underestimation Refinement: A General Enhancement Strategy for Exploration in Recommendation Systems. In *SIGIR*. 1818–1822.

[48] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*. 1441–1450.

[49] Yukihiro Tagami, Shingo Ono, Koji Yamamoto, Koji Tsukamoto, and Akira Tajima. 2013. Ctr prediction for contextual advertising: Learning-to-rank approach. In *ADKDD*. 1–8.

[50] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*. 565–573.

[51] Isaac Triguero, Salvador García, and Francisco Herrera. 2015. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems* 42, 2 (2015), 245–284.

[52] Ilya Trofimov, Anna Kornetova, and Valery Topinskiy. 2012. Using boosted trees for click-through rate prediction for sponsored search. In *ADKDD*. 1–6.

[53] Jesper E. van Engelen and Holger H. Hoos. 2020. A survey on semi-supervised learning. *Machine Learning* 109, 2 (2020), 373–440.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*.

[55] Jianling Wang, Kaize Ding, and James Caverlee. 2021. Sequential Recommendation for Cold-start Users with Meta Transitional Learning. In *SIGIR*. 1783–1787.

[56] Lu Wang, Xuanqing Liu, Jinfeng Yi, Yuan Jiang, and Cho-Jui Hsieh. 2020. Provably Robust Metric Learning. In *NeurIPS*.

[57] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *ADKDD*. 1–7.

[58] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In *WWW*. 1785–1797.

[59] Wei Wang and Zhi hua Zhou. 2010. A New Analysis of Co-Training. In *ICML*. 1135–1142.

[60] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *ICML*. 1192–1199.

[61] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A simple convolutional generative network for next item recommendation. In *WSDM*. 582–590.

Table 5: The statistics of MSLR10K and MSLR10K-Candi.

	MSLR10K	MSLR10K-Candi
#Features	136	136
#Qureies training	6000	6000
#Queries test	1928	1928
#Docs in training	723412	510256
#Docs in test	239652	168031

[62] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. In *NeurIPS*, Vol. 30. 3391–3401.
[63] Yan Zhang, Jonathon Hare, and Adam Prügel-Bennett. 2020. FSPool: Learning Set Representations with Featurewise Sort Pooling. In *ICLR*.
[64] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *WWW*. 167–176.
[65] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *AAAI*, Vol. 33. 5941–5948.
[66] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD*. 1059–1068.
[67] Zhi-Hua Zhou. 2018. A brief introduction to weakly supervised learning. *National Science Review* 5, 1 (2018), 44–53.
[68] Zhi-Hua Zhou and Ming Li. 2010. Semi-supervised learning by disagreement. *Knowledge and Information Systems* 24, 3 (2010), 415–439.

A EXPERIMENTS ON MSLR10K-CANDI

As stated in Section 5, public CTR prediction datasets are not suitable for us to conduct experiments because the candidate items are not available. Fortunately, we find that public learning-to-rank datasets provide a set of items for each query, which is naturally similar to our application scenario. Therefore it becomes possible for us to construct a CTR prediction dataset with candidate items from public learning-to-rank dataset. Here, we describe how we construct a dataset MSLR10K-Candi from the public MSLR10K and report our performance on it. MSLR10K-Candi will also be published if we get permission from the original authors of MSLR10K.

MSLR10K-Candi. Microsoft LETOR 10K (MSLR10K)[6] is a public available learning-to-rank dataset labeled by human experts with relevance judgment ranging from irrelevant to perfectly relevant. For each query, there exists a list of documents to be ranked with different relevance scores from 0 to 4 (4 means most relevant). All documents in this ranking list have been exposed. We set the click label to 1 if the relevance score is 3 or 4, otherwise set to 0. To simulate the online environment where many items in the candidate list have not been exposed, for a query related to n documents, we randomly sample another $n/2$ documents from the whole dataset. The candidate item list consists of these $n + n/2$ documents. These sampled documents are regarded as non-exposed samples. We set $n \leq 100$. The longer ranking lists will be truncated to 100. Thus, we construct MSLR10K-Candi, each sample in which contains an item, its corresponding candidate item list with their exposure labels, and the click label. Statistics are shown in Table 5.

Results on MSLR10K-Candi. The baseline DNN is a 3-layer MLP of size 256-128-1 with ReLU. The dimension of Transformer is 136, and the number of heads is 4. As shown in Table 6, similar to the experiments on CandiCTR-Pub, incorporating CIM improves AUC by 0.016, LogLoss by 0.006, and GAUC by 0.019, which is consistent with our main experiment in Section 5, and further verifies the effectiveness of our method.

[6] http://research.microsoft.com/en-us/projects/mslr/

Table 6: Experimental results on MSLR10K-Candi dataset. ↑ means the larger number is better; ↓ means the smaller number is better.

No.	Model	AUC↑(Δ)	LogLoss↓(Δ)	GAUC↑(Δ)
(11)	DNN	0.729	0.112	0.713
(12)	**DNN+Ours**	**0.745(+0.016)**	**0.106(-0.006)**	**0.732(+0.019)**