User Behavior Simulation for Search Result Re-ranking

JUNQI ZHANG and YIQUN LIU, Tsinghua University
JIAXIN MAO, Renmin University of China
WEIZHI MA, JIAZHENG XU, and SHAOPING MA, Tsinghua University
QI TIAN, Huawei Cloud & AI

Result ranking is one of the major concerns for Web search technologies. Most existing methodologies rank search results in descending order of relevance. To model the interactions among search results, reinforcement learning (RL algorithms have been widely adopted for ranking tasks. However, the online training of RL methods is time and resource consuming at scale. As an alternative, learning ranking policies in the simulation environment is much more feasible and efficient. In this article, we propose two different simulation environments for the offline training of the RL ranking agent: the Context-aware Click Simulator (CCS) and the Fine-grained User Behavior Simulator with GAN (UserGAN). Based on the simulation environment, we also design a User Behavior Simulation for Reinforcement Learning (UBS4RL) re-ranking framework, which consists of three modules: a feature extractor for heterogeneous search results, a user simulator for collecting simulated user feedback, and a ranking agent for generation of optimized result lists. Extensive experiments on both simulated and practical Web search datasets show that (1) the proposed user simulators can capture and simulate fine-grained user behavior patterns by training on large-scale search logs, (2) the temporal information of user searching process is a strong signal for ranking evaluation, and (3) learning ranking policies from the simulation environment can effectively improve the search ranking performance.

CCS Concepts: • **Information systems → Retrieval models and ranking;**

Additional Key Words and Phrases: Information retrieval, ranking, user simulation, reinforcement learning, generative adversarial networks

5

This article is an extension of Zhang et al. [79] Compared with the previous conference version, it introduces a fine-grained user behavior simulator UserGAN, which is based on the generative adversarial network. Besides click information, in this work, click time is also considered for ranking evaluation and optimization. An extensible ranking framework (UBS4RL) is also proposed to optimize online and offline evaluation metrics. Extensive experimental assessment shows the effectiveness of the proposed user simulators for ranking evaluation and the UBS4RL framework for ranking optimization. This work was supported by the Natural Science Foundation of China (grants 61732008 and 61902209), Beijing Academy of Artificial Intelligence (BAAI) and Tsinghua University Guoqiang Research Institute, Beijing Outstanding Young Scientist Program (no. BJJWZYJH012019100020098), and Intelligent Social Governance Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Renmin University of China.
Authors' addresses: J. Zhang, Y. Liu (corresponding author), J. Xu, and S. Ma, Department of Computer Science and Technology, Institute for Artificial Intelligence, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China; emails: zhangjq17@mails.tsinghua.edu.cn, yiqunliu@tsinghua.edu.cn, xujz18@mails.tsinghua.edu.cn, msp@tsinghua.edu.cn; J. Mao, Beijing Key Laboratory of Big Data Management and Analysis Methods, Gaoling School of Artificial Intelligence, Renmin University of China, Beijing 100872, China; email: maojiaxin@gmail.com; W. Ma, Institute for AI Industry Research (AIR), Tsinghua University, Beijing 100084, China; email: mawz@tsinghua.edu.cn; Q. Tian, Huawei Cloud & AI, China; email: tian.qi1@huawei.com.

1046-8188/2023/01-ART5 $15.00
https://doi.org/10.1145/3511469

ACM Reference format:
Junqi Zhang, Yiqun Liu, Jiaxin Mao, Weizhi Ma, Jiazheng Xu, Shaoping Ma, and Qi Tian. 2023. User Behavior Simulation for Search Result Re-ranking. *ACM Trans. Inf. Syst.* 41, 1, Article 5 (January 2023), 35 pages.
https://doi.org/10.1145/3511469

1 INTRODUCTION

Web search engines usually rank results according to their relevance scores in descending order. Assuming that users browse search results sequentially from top to bottom on **search engine result pages (SERPs)**, ranking the relevant results at the top positions reduces users' efforts in locating useful information. For example, **learning-to-rank (LTR)** methods can either adopt the relevance annotations or user click feedback to train a ranking model. Although this established approach has achieved much success in improving search ranking performance, it faces two important challenges with the development of recent search techniques.

First, the sequential browsing hypothesis no longer holds in today's heterogeneous search scenarios where images, videos, news, and even applications are aggregated with 10 blue hyperlinks in a unified result list. Previous studies have revealed that users' attention may be drawn toward non-textual information items [66] and vertical results [15, 71], which alters users' behavior and leads to a non-sequential examination sequence. Second, the interactions among search results are largely ignored by existing methods. The selection and ranking of search results depends not only on their own relevance but also on the context—for example, the complementarity and redundancy of search results [81].

Regarding the preceding two issues, some existing research efforts formulate the ranking problem as a **Markov decision process (MDP)** and solve it with the **reinforcement learning (RL)** paradigm [60, 67, 75, 76, 81]. The RL framework is suitable for modeling the interaction effects of search results; it optimizes the ranking performance globally while considering the context. The RL approach is more complex than linear ranking methods based on relevance.

However, training RL agents can be a great challenge. The main problem is how to evaluate the re-ranked result list and design reward functions for the RL agents. Both offline and online evaluation methods have been adopted in measuring the performance of information systems. Offline evaluation is based on relevance annotations and usually suffers from expensive annotation costs and search intent mismatches between assessors and practical search engine users. Online evaluation regards user feedback such as clicks to indicate satisfaction. However, poor-quality ranking lists will harm the user experience. It also takes time to collect enough trials and rewards from users to evaluate the ranking policy.

To not harm the online system performance and to train the RL agent more efficiently, a number of existing works have adopted a simulation way for online evaluation [67]. To model user interactions with practical search systems, click models such as UBM [25], DCM [32], and DBN [13] represent user behaviors as sequences of observable and hidden events. These works are mainly based on the **probabilistic graphical model (PGM)** framework and handcrafted behavior assumptions. However, most click models have been designed for homogeneous result lists. User interactions with search systems are much more complex in heterogeneous search scenarios. In this work, we propose two novel user simulators to mimic fine-grained user behavior: **Context-aware Click Simulator (CCS)** and **Fine-grained User Behavior Simulator with GAN (UserGAN)**.

The first simulator, CCS, is based on a hierarchical structure including the session level and the result level. It predicts click probability not only according to the multimodal contents of the search result but also the previous click and SERP context. The interactions among search results can be implicitly learned through the sequential predicting process. CCS is simpler in structure

and efficient in training, but it can only predict the click/skip signal of the user browsing process. Nevertheless, user behavior on SERPs is closely correlated with time and space. The click event triggered by a user is associated with both temporal signals (e.g., the click time since query submission) and spatio signals (e.g., the position and area of the search result clicked). Existing studies have shown that the time between user browsing actions is an important signal to indicate users' satisfaction in Web search [4].

To take temporal information into account, we propose the second simulator, UserGAN, to simulate fine-grained user behavior. We use the multidimensional spatio-temporal signal (described in Section 1) associated with the click event to provide a comprehensive understanding of user behavior patterns. We assume that user behaviors obey an implicit distribution. The user simulator aims to learn and generate an indistinguishable user behavior distribution from the genuine one. As GANs have shown great success in a wide range of areas, including computer vision [46, 63, 83], natural language processing [26, 78], and **information retrieval (IR)** tasks [35, 49, 72], we choose GAN to generate a simulated user behavior distribution through adversarial learning. To represent the multidimensional spatio-temporal signals of user behavior, we also design a modulation-demodulation process. The modulator encodes the spatial and temporal signals with a Gaussian distribution while the demodulator recovers the fine-grained user interaction signals with existing constrained optimization algorithms. UserGAN can simulate click times along with clicks, which helps perform accurate online evaluation for heterogenous Web search systems.

Based on the user simulators, we propose a **User Behavior Simulation for Reinforcement Learning (UBS4RL)** framework for result re-ranking in search engines. As shown in Figure 1, UBS4RL consists of three modules. First, a feature extractor (JRE [80] or TreeNN [48]) fuses the visual, textual, and structural information of vertical results to obtain the dense representation. The extracted result feature is delivered to the user simulator and the ranking agent for subsequent processing. In this work, we adopt TreeNN as the feature extractor. It embeds the visual and textual information into a HTML parse tree, which can better leverage the structural information. Second, a user simulator (CCS [81] or UserGAN) exploits user behavior patterns from search logs and mimics fine-grained user feedback including click and time information. The user simulator interacts with the ranking agent and provides rewards to guide the updating of the ranking policy. Third, a ranking agent (RLRanker [81]) formulates search result ranking as an MDP. At each step t, the ranking agent chooses a result from the candidates (`actions`) to place at position t in the ranking list. By regarding the already placed results as the current `state` and setting a proper immediate `reward`, the MDP ranking model can take the interactions among search results into consideration. In this work, the MDP problem is solved with the policy gradient algorithm called *REINFORCE* [68].

During training, the ranking agent generates re-ranked result lists and obtains feedback from the simulation environment (user simulator). Through a large number of trials, the ranking agent can ultimately learn an optimal ranking policy. For testing, the optimal ranking agent is used to iteratively fill results into an empty list from top to bottom for re-ranking task. The user simulator is not used in this stage, as we do not need the rewards to update the ranking agent.

Specifically, the contributions of the proposed UBS4RL framework can be summarized as follows:

- Two different user simulators are constructed with historical search logs to mimic fine-grained user behaviors. The user simulators provide reliable online evaluation for ranking optimization, which enable low-cost and efficient online training of RL agents for ranking tasks.
- A unified UBS4RL re-ranking framework is proposed to take the interactions among search results into consideration for re-ranking. It involves three subtasks: feature extraction of

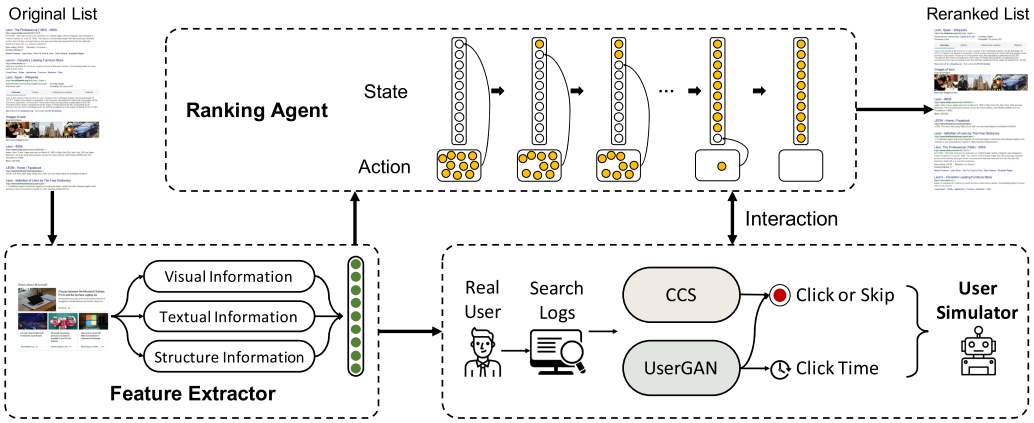

Fig. 1. The pipeline of the UBS4RL ranking framework, which consists of three modules: a feature extractor for heterogeneous search results, a user simulator for collecting simulated user feedback, and a ranking agent for generation of optimized result lists.

heterogeneous search results, ranking evaluation by simulated users, and ranking policy optimization with RL agents. The framework has the potential to be further extended to a wide range of optimization tasks for different information systems.

- By conducting experiments on both synthetic and practical datasets, we show significant improvement on search re-ranking performance, which verifies the effectiveness of the proposed re-ranking framework.

The rest of the paper is organized as follows. We describe related work in Section 2. Section 3 formally introduces the two user simulators: CCS and UserGAN, while Section 4 introduces the ranking agent. The experiment settings and results are presented in Sections 5 and 6, respectively. Finally, we conclude this paper and discuss future work in Section 7. The notations used in this paper are summarized in Table 1.

2 RELATED WORK

2.1 RL in IR

RL [39, 41] originates from the understanding of how humans learn to take actions in the environment from the domains of psychology and neuroscience. It can be formulated as an MDP. An MDP is typically defined as a tuple consisting of four elements <state space, action space, transition function, reward function>, which is well aligned with RL problem settings. In an MDP, the current state encapsulates all of the information needed to make a decision for the future. Similarly, we also take an action based on the current state in RL. Recently, combining RL and deep learning techniques has shown great success in broad applications [55], such as games [68, 70], robotics [79], computer vision [8], and natural language processing [69] tasks.

Recently, many works have applied RL methods for ranking tasks [60, 67, 75, 76]. Oosterhuis and De Rijke [60] adopt the RL approach to deal with complex ranking settings, which learns both the user preferred document order and displays the position order for result presentations. The page presentation optimization of SERPs has also been recast as an RL problem in the work of Wang et al. [75]. Wei et al. [76] formulate the ranking problem as an MDP to optimize the evaluation measure calculated at all positions. Some work has adapted RL for complex online

Table 1. Notations of the UBS4RL Ranking Framework

Notation	Description
q, f_q	Query, Query feature
$o_i, f_{o,i}$	The ith search result on the SERP, feature of o_i
U, u_i	User interaction feedback, feedback of o_i
$\boldsymbol{\tau}, \tau_i$	Click times from user entering the session, click time for o_i
$\boldsymbol{\eta}, \eta_i$	Rank of search results, rank of o_i
$\boldsymbol{\alpha}, \alpha_i$	Heights of search results, height of o_i
$\boldsymbol{\rho}, \rho_i$	Click probabilities of search results, click probability of o_i
$\boldsymbol{c}, c_i$	Click sequence (click or skip) for the SERP, click behavior of o_i
$\Omega_{\text{genuine}}, \Omega_{\text{simulate}}$	Genuine/simulation user behavior distribution
$S_{\text{time}}, S_{\text{t,i}}$	Temporal signals, the ith temporal signal
$S_{\text{space}}, S_{\text{s,i}}$	Spatio signals, the ith spatial signal
S_{mst}	Multidimensional spatio-temporal signal
$V_{\text{carrier}}, \widehat{V}_{\text{carrier}}$	Genuine/simulated carrier wave encoding the multidimensional spatio-temporal signal
$s, \mathcal{S}$	State, State space
$a, \mathcal{A}$	Action, Action space
$\mathcal{T}$	Transition function
$\mathcal{R}, r, R$	Reward function, Step reward, Discounted cumulative reward
γ	Discount factor
π, π^*	Policy, Optimal policy

system optimization [67], in which a virtual retail platform to train ranking policies offline is constructed. Assuming that the environment is static, inverse RL methods [56] can learn a reward function from the data and train policies according to the reward function.

There are several differences between our work and previous RL ranking methods [60, 75, 76]. First, previous works only use human-annotated relevance labels to learn a ranking policy. However, human annotations are very expensive. In this work, we leverage the abundant user search logs to optimize the ranking policy, which is more realistic in practical search systems. Second, to solve the online training problem, we construct a static simulation environment to simulate search engine users and provide rewards, which is similar to the work of Shi et al. [67]. Third, previous works only optimize offline evaluation metrics such as NDCG (normalized discounted cumulative gain) [64]. However, online evaluation metrics such as CTR (click through rate) and MRR (mean reciprocal rank) may align better with user satisfaction [17] and are widely adopted in search engines. In this work, the proposed UBS4RL framework can optimize both offline and online evaluation metrics.

2.2 Search System Evaluation

Evaluation serves as an important element of academic and industrial IR research. Offline and online evaluation are conducted to measure how well a search system can satisfy users' information needs. Existing RL-based ranking solutions [9, 60, 76, 77, 84] mainly focus on offline evaluation, which is mainly originated from the Cranfield approach [22]. First, professional experts or crowdsourcing annotators are required to judge the relevance of query-document pairs based on pre-defined evaluation criteria. Second, some evaluation metrics are designed according to user behavior analyses to compare the performance of different search systems. Typical offline metrics

include AP (average precision), NDCG [64], and RBP (rank-biased precision) [54]. However, offline evaluation has encountered two major problems. First, search intent mismatches exist between the judgment of annotators and actual users. Second, large-scale annotations are very expensive, making it infeasible to obtain an unbiased evaluation result on an incomplete dataset.

In contrast, online evaluation takes advantage of the actual interactions between users and search engines to measure the quality of ranking algorithms. Users' interactions such as click or skip of search results and dwell time can reflect the actual users' experiences in a search process. The aim of online evaluation is to extract the unbiased user satisfaction information from noisy and biased behavior data. Online evaluation is straightforward to indicate users' experience and is inexpensive to collect enough log data. Typical online metrics include CTR, UCTR (binary value representing click) [20], MRR, number of clicks divided by the PLC (position of the lowest click) [12], and so on. However, conducting online evaluation usually require a substantial quantity of user traffic for interactive experiments. It is risky to return a poor-quality ranking list to search engine users, which may heavily harm users' experiences. In addition, the deployment of online evaluation is costly and time consuming, making it difficult to perform timely performance feedback. To tackle these problems, simulation can be adopted as an alternative to online systems.

2.3 User Behavior Simulation

Simulation has become a key method in the investigation of user interactive systems such as dialog systems, search engines, and E-commerce platforms. The purpose of simulation studies is twofold: evaluation and exploration. First, simulations provide reliable evaluation for online systems such as CTR and the revenue of products in E-commerce platforms [67]. Second, simulations are usually adopted to determine how the systems' performance changes under different conditions [52]. For example, the simulation environment provides rewards for the agent in RL approaches when the actual environment is not available [33].

In search engines, click models can be adopted to simulate actual users' browsing behavior. Click models regard users' interactions with search engines (e.g., clicks) as implicit relevance feedback and represent user behavior as a sequence of observable and hidden events. They are mainly based on the PGM framework, and some prior hypotheses of user behavior have to be set manually [6]. Many click models have been proposed according to different user behavior hypotheses, such as the **User Browsing Model (UBM)** [25], **Dynamic Bayesian Network model (DBN)** [13], and **Dependent Click Model (DCM)** [32]. As click dwell time and click sequence information are strongly correlated with users' perceived relevance and search satisfaction, the Time-Aware Click Model (TACM) also incorporates temporal information into click models [47]. In heterogeneous search scenarios, the ordering of the vertical results influence users' browsing process to a large extent. Users often examine the higher-ranked vertical results first; the results beside them also gain more attention [15, 71]. To deal with the emerging vertical results, some advanced click models such as UBM-Layout [21] and the **Mobile Click Model (MCM)** [50] are proposed to take the vertical bias into consideration. Besides the PGM framework, some click models based on neural networks have been proposed in recent years, which are based on the idea of distributed representation of information and interactions, such as the **Neural Click Model (NCM)** [6] and Click Sequence Model (CSM) [7].

However, click models are mainly based on prior user behavior hypotheses that require extensive user studies. It is also impossible to consider all of the hypotheses in a single click model. Click models also suffer from the emergence of new queries and documents, because they can only deal with query-document pairs that are available in the logs. In addition, click models mainly focus on click behaviors; however, most of them ignore the click time, which is also informative.

In this work, we propose a new approach to simulate fine-grained user behavior with the generative model. The framework requires no prior user behavior hypotheses and directly learns user behavior pattern from log data. It also has a strong generalization ability in heterogeneous search scenarios.

2.4 Generative Adversarial Networks

Generative models can learn high-dimensional, complex real data distribution and may be a nice fit for understanding the distribution of user behavior patterns hidden deeply in feedback signals. The main goal of generative models is to learn an estimate of an unknown distribution p_{data} by minimizing the distance between the estimated probability distribution p_{model} and p_{data}. Some generative models estimate p_{model} explicitly such as Flow-based Generative Models [57, 58] and **Variational Autoencoder (VAE)** [40], whereas others can only generate samples from p_{model} like he **generative adversarial networks (GANs)** [27, 28].

The Generative Adversarial Network was first proposed by Goodfellow et al. [28] to generate realistic simulated images. It provides a novel approach to generate data samples through adversarial learning. The framework consists of a generator G and a discriminator D. The generator tries to capture the real data distribution and generate indistinguishable data samples to confuse the discriminator, whereas the discriminator tries to distinguish between real (drawn from p_{data}) and synthetic (drawn from p_{model}) data samples. The objective is to find the Nash equilibrium of the minimax two-player game.

The use of GANs has achieved great success in a wide range of deep learning fields like computer vision, natural language processing, and IR tasks. The IR tasks include image generation [63], image translation [83], image super-resolution [46], sentence generation [26, 78], audio generation [43, 57], and so on. In IR fields, Wang et al. [72] proposed IRGAN [72] to combine the generative retrieval method and the discriminative retrieval method. GAN has also been used for document common feature learning (CAN [16]), community detection (CommunityGAN [35]), expert retrieval (USGAN [44]), graph representation learning (GraphSGAN [24]), recommendation system (CFGAN [11]), personalized search (PSGAN [49]), and so on.

Although GAN is capable of generating new random examples for a given dataset, there is no way to control the generation process. This can be overcome by an enhanced Conditional GAN [53]. By conditioning the model on additional information, it is possible to control the data generation process. For example, in the MNIST dataset, we can generate a given number of targeted images. In this work, we adopt Conditional GAN to simulate a user behavior distribution. The generated user behavior is based on the multimedia content of each search result on SERPs. In this way, we can generate SERP-specific rather than random user behavior.

2.5 Counterfactual LTR

LTR methods have been widely adopted in information retrieval tasks. Generally, LTR algorithms take a set of search results as input and predict the ranking scores [2, 37, 73, 74] or a permutation of them [61]. Search results are then sorted with the ranking scores in descending order. Many LTR methods take users' click feedback for learning, as click feedback is abundant in search engines. However, click feedback is well known to be biased and noisy. To learn an unbiased ranking model with biased click feedback, a variety of unbiased LTR methods have been proposed, which can be broadly categorized into two families: the bandit LTR methods and the counterfactual LTR methods. The bandit LTR methods usually learn from online result manipulation and user feedback (e.g., [34, 59, 65]), which will negatively impact users' search experience. The counterfactual LTR methods originally leverage click logs for offline learning [2, 37, 73, 74]. The key idea of counterfactual LTR methods is to learn **Inverse Propensity Weighting (IPW)** [37, 73] and the

estimation of examination propensity from biased click logs. For example, Wang et al. [71] propose a few methods to estimate the selection bias and address it using IPW. Joachims et al. [37] derive a Propensity-Weighted Ranking SVM for discriminative learning from biased click feedback. They adopt click models as the propensity estimator. In addition, Wang et al. [72] also propose a **regression-based expectation-maximization algorithm (REM)** algorithm to debias click feedback without relying on randomization. To unify the learning of propensity models and ranking models, Ai et al. [2] propose a **dual learning algorithm (DLA)** that jointly learns an unbiased ranker and an unbiased propensity model.

Similar to counterfactual LTR methods, the proposed UBS4RL framework also learns a ranking model with biased click logs in a offline manner. However, counterfactual LTR methods mainly focus on learning a propensity model to estimate the user examination probability at each position and use the estimated weights to modify the training loss. Differently, UBS4RL neither learns the propensity model nor modifies the training loss of the ranking model. Instead, we propose the user simulator to learn both the intrinsic relevance of search results and user click bias on the SERP. The user simulator mimics biased user click and click time behavior on different randomization of result lists, which is not exactly a debiasing model. Specifically, the ranking policy will try a lot of different rankings of the same result list. The user simulator provides biased click feedback on each given ranking list. Although the feedback itself is biased, the ranking policy can eventually find a best ranking strategy through a large number of trials. More importantly, the RL methods make successive decisions based on current states, which can take the interactions among search results into consideration. However, the counterfactual LTR methods do not take the underlying assumption that ranking of search results can be formulated as an MDP.

3 UBS4RL: USER SIMULATOR

3.1 CCS: Context-aware Click Simulator

3.1.1 Session Level. As shown in Figure 2, the lower level of the CCS is the session based module. It is implemented as a **bi-directional GRU (BiGRU)** [19] framework. The concatenation of the last hidden states of the two directions in the BiGRU is adopted as the session feature. From top to bottom and bottom to top, the session feature captures the global context information of the result list:

$$h_t = [z_{1:t}, z_{N:N-t+1}], \tag{1}$$

$$h_{t+1} = \text{BiGRU}(h_t, f_{o,t+1}), \tag{2}$$

$$f_{\text{ses}} = h_N = [z_{1:N}, z_{N:1}], \tag{3}$$

where N is the number of search results in a query session, $z_{1:i}$ is the hidden state of BiGRU in the forward direction while $z_{N:i}$ is that in the backward direction at step i, $[\cdot]$ denotes the concatenation of different features, $f_{o,t}$ is the feature of the tth search result in the session, and f_{ses} is the session feature. The initial hidden state h_0 for BiGRU is the query feature f_q.

3.1.2 Result Level. The higher result-level module is designed to predict the click probability of each search result sequentially based on the session context. At each step, three different kinds of features are aggregated as the input to a single-direction GRU. The first one is the session feature encoding the global context. The second one is the result feature at each step. A common assumption in click models is that previous clicks influence the click behavior in the users' subsequent browsing. Following this assumption, we take the click behavior at the previous step as the third input feature.

The click behavior can be `click`, `skip`, or `unknown` (for the initial step), which is embedded into a vectorial representation through the `click encoder` (Equation (4)). At each step of the GRU, the

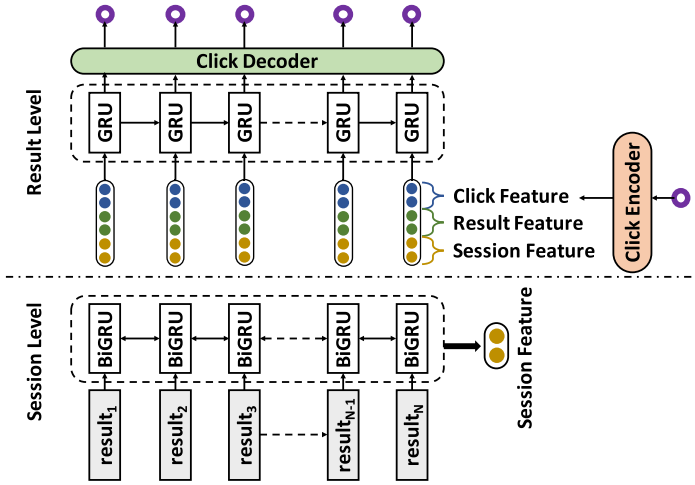

Fig. 2. The framework of CCS. The hollow purple circles represent user clicks. The session level compresses the global information of the SERP as the context. The result level predicts click probability according to the multimodal contents of the search result, previous click, and SERP context.

hidden state is projected into a click probability score between 0 and 1 through the `click decoder` (Equation (9)). The prediction of the result level module can be formulated as follows:

$$f_{\text{clk},t} = \text{embedding}(c_{t-1}), \tag{4}$$

$$\hat{f}_{o,t} = W_o f_{o,t} + b_o, \tag{5}$$

$$\hat{f}_{\text{ses}} = W_{\text{ses}} f_{\text{ses}} + b_{\text{ses}}, \tag{6}$$

$$f_t = [f_{\text{clk},t},\ \hat{f}_{o,t},\ \hat{f}_{\text{ses}}], \tag{7}$$

$$h_t = \text{GRU}(h_{t-1}, f_t), \tag{8}$$

$$\rho_t = \text{sigmoid}(W_{\text{clk}} h_t + b_{\text{clk}}), \tag{9}$$

where $c_{t-1} \in \{\texttt{click}, \texttt{skip}, \texttt{unknown}\}$ denotes the click behavior in step t-1; $W_o, W_{\text{ses}}, W_{\text{clk}}$ and $b_o, b_{\text{ses}}, b_{\text{clk}}$ are weights and biases for each kind of features, respectively; and ρ_t is the final predicted click probability of the tth search result in the ranking list $\rho_t \in [0, 1]$.

The loss function is `CrossEntropy`, which is defined as follows:

$$\mathcal{L}(\boldsymbol{\rho}, \boldsymbol{c}; \theta) = \frac{1}{M} \sum_{j=1}^{M} \sum_{i=1}^{N} (-c_{j,i} \log \rho_{j,i} - (1 - c_{j,i}) \log (1 - \rho_{j,i})) + \lambda ||\theta||_2^2, \tag{10}$$

where M is the number of training sessions; N is the number of results in the query session; $c_{j,i}$ and $\rho_{j,i}$ denote the click label and predicted click probability of the ith result in the jth training sample, respectively; θ includes all the parameters in the neural network; and λ denotes the $L2$ regularizer coefficient.

3.2 UserGAN: Fine-grained User Behavior Simulator with GAN

We use the multidimensional spatio-temporal signal to represent user behavior in UserGAN. However, it is composed of multiple separate and discrete signals such as click/skip signal, click time, click position, and result area, which cannot be directly learned and generated by GANs. To tackle this problem in generating discrete data distributions, there are mainly three kinds of existing

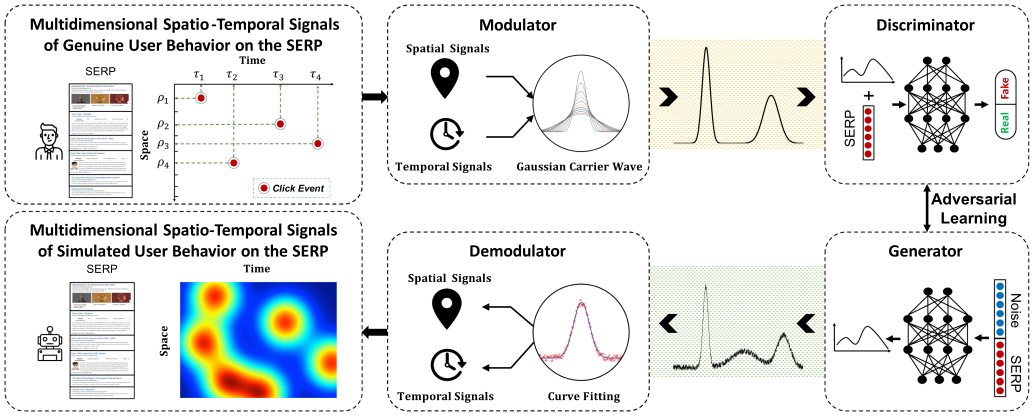

Fig. 3. The UserGAN framework consists of three components: modulator, simulator, and demodulator. The multidimensional spatio-temporal signal is used to describe fine-grained user behavior. The simulated user behavior can be used for Web search online evaluation. ρ_i denotes the result position. τ_i denotes the click time.

solutions: (1) integrating a GAN with VAE [40] or Autoencoder (AE) to project discrete data into continuous latent space [38]; (2) designing tailored objective functions instead of the standard GAN objective function [14, 82]; and (3) integrating GAN with RL, which regards the discriminator as the environment to provide rewards for the generator [26, 31, 45, 78]. Unfortunately, none of these solutions are suitable for the online evaluation scenarios since we aim to generate multiple discrete signals to simulate fine-grained user behavior. Inspired by signal transmission approaches in electronics [1], we leverage a modulation and demodulation framework. The modulator encodes the spatial and temporal signals with a Gaussian distribution while the demodulator can recover the fine-grained user interaction signals with existing constrained optimization algorithms [42]. The framework of UserGAN is shown in Figure 3, which is composed of three components: the modulator, the simulator, and the demodulator. The following gives some basic definitions of UserGAN.

3.2.1 *Preliminary.*

Definition 1 (Multidimensional Spatio-temporal Signal). User behavior in real-world online systems is linked to time and space—for example, the mouse movement event or fixation event [29] to a specific area at a specific time on a webpage. In this work, we focus on the click event on SERPs. The click event triggered by a user is associated with temporal signals $S_{\text{time}} = \{S_{\text{t},1}, S_{\text{t},1}, \ldots, S_{\text{t,m}}\}$ like the timestamp and spatial signals $S_{\text{space}} = \{S_{\text{s},1}, S_{\text{s},1}, \ldots, S_{\text{s,n}}\}$ like the result position and result area, where $S_{\text{t,i}}$ and $S_{\text{s,j}}$ indicate independent temporal and spatial signals, respectively. These separate signals together constitute the multidimensional spatio-temporal user behavior signal $S_{\text{mst}} = S_{\text{time}} \cup S_{\text{space}}$.

Definition 2 (Modulator). The modulator is used to construct a carrier wave with input signals. They are usually used in the real world to place multiple audio and video streams on the same wire. For online systems, the multidimensional spatio-temporal signal S_{mst} of user behavior can also be modulated as a carrier wave so that it can be transmitted to other modules for signal processing and learning. The multiple spatial and temporal signals can be subsequently demodulated to recover the original information [1]. The carrier wave V_{carrier} is defined as follows:

$$V_{\text{carrier}} = \text{Modulator}(S_{\text{mst}}). \tag{11}$$

Definition 3 (Simulator). The simulator captures the properties of real-world signals and generates the simulated signals when the real environment is not accessible. Assume that with different S_{mst} from the real environment, the modulator can generate a set of V_{carrier} samples denoted as Λ. The simulator can then learn about the characteristics of carrier waves from the samples Λ. When we feed the condition information ϱ to the simulator, it can produce a simulated carrier wave $\widehat{V}_{\text{carrier}}$ based on ϱ. The simulator is defined as follows:

$$\widehat{V}_{\text{carrier}} = \text{Simulator}(\varrho|\Lambda). \tag{12}$$

Definition 4 (Demodulator). A demodulator is a module that is used to recover the signal information from the modulated carrier wave. Demodulation is the reverse process of modulation. For example, the signals transmitted on the wire are recovered to the original audio and video streams by the demodulator. In our case, the carrier wave V_{carrier} is demodulated to the multidimensional spatio-temporal signal $\hat{S}_{\text{mst}}$ that contains the multiple original spatial and temporal signals of user behavior. The aim of the demodulator is to restore the original information with as little loss as possible.

$$\widehat{S}_{\text{mst}} = \text{Demodulator}(\widehat{V}_{\text{carrier}}) \tag{13}$$

In this work, the temporal signal Click Time $\boldsymbol{\tau}$ (the interval of a click action since the user enters the session), spatial signals Click Position $\boldsymbol{\eta}$ (the ranks of search results), and Result Area $\boldsymbol{\alpha}$ (the heights of search results as they have the same width) are considered jointly to describe the fine-grained user behavior on SERPs:

$$S_{\text{mst}} = S_{\text{time}} \cup S_{\text{space}} = \{\boldsymbol{\tau}, \boldsymbol{\eta}, \boldsymbol{\alpha}\}, \tag{14}$$

$$\boldsymbol{\tau} = \{\tau_i\}, \boldsymbol{\eta} = \{\eta_i\}, \boldsymbol{\alpha} = \{\alpha_i\}, i = 1, 2, \ldots, N, \tag{15}$$

where N is the number of search results on the SERP.

The UserGAN framework follows a pipeline that takes three steps to simulate user behavior. First, the multidimensional spatio-temporal signal S_{mst} is fed into the modulator and transformed into the continuous carrier wave V_{carrier}. Second, the simulator learns to generate the simulated carrier wave $\widehat{V}_{\text{carrier}}$, which is implemented as a Conditional GAN. Third, the simulated carrier wave is recovered to the original spatial and temporal signals $\widehat{S}_{\text{mst}}$ of user behavior by the demodulator for online evaluation. The detailed implementation of the three modules is introduced in the following.

3.2.2 Modulator: Gaussian Distribution. The Gaussian distribution (also known as the normal distribution) is a bell-shaped curve that occurs often in nature. It is important in statistics and is often used in the natural and social sciences to represent real-valued random variables whose distributions are not known [10]. The probability density function of the Gaussian distribution $\mathcal{N}$ is controlled by the mean value μ and the standard deviation σ.

$$\mathcal{N}: \quad \varphi(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{16}$$

The standard Gaussian distribution has its center at zero ($\mu = 0$) and a variance of unity ($\sigma = 1$). Every Gaussian distribution is a version of the standard Gaussian distribution whose domain has been stretched by σ and then translated by μ.

Gaussian modulation of single click event. The user's psychological activities during a click event on SERPs, such as attention, planning, and action, can be assumed to obey the Gaussian distribution. We use the location factor μ to describe the spatial signals $S_{\text{space}} = \{\boldsymbol{\eta}, \boldsymbol{\alpha}\}$, whereas the scale factor σ is used to describe the temporal signals $S_{\text{time}} = \{\boldsymbol{\tau}\}$.

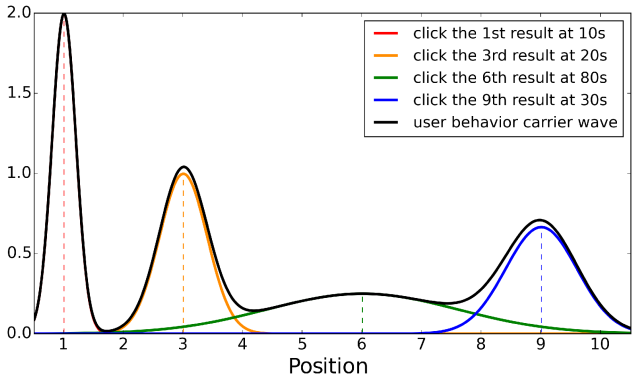

Fig. 4. An example of the user behavior carrier wave of a search session. The SERP contains 10 search results. The user clicks the 1st, 3rd, 6th, and 9th result at times 10s, 20s, 80s, and 30s, respectively. β is set to 0.02.

In this work, we use the unity height of the search results to denote the result area $\boldsymbol{\alpha}$ and thus $\alpha_i = 1, i = 1, 2, \ldots, N$. We assume that the click position $\boldsymbol{\eta}$ is at the center of each search result because there is no accurate click position recorded in the search log and $\eta_i = i, i = 1, 2, \ldots, N$. A hyperparameter β is used to control the sensitivity of the Gaussian distribution $\mathcal{N}$ to the click time $\boldsymbol{\tau}$.

$$\mu_i = \Theta_\mu(\boldsymbol{\eta}, \boldsymbol{\alpha}) \tag{17}$$

$$= \sum_{k=1}^{\eta_i - 1} \alpha_k + \frac{1}{2}\alpha_{\eta_i} = i - 0.5 \tag{18}$$

$$\sigma_i = \Theta_\sigma(\boldsymbol{\tau}) \tag{19}$$

$$= \beta\tau_i \tag{20}$$

For each click event, the representing Gaussian distribution $\mathcal{N}_c$ can be defined as follows:

$$\mathcal{N}_c : \quad \varphi_c(x|\mu_i, \sigma_i^2) = \frac{1}{\beta\tau_i\sqrt{2\pi}} e^{-\frac{(x-(i-0.5))^2}{2(\beta\tau_i)^2}}, \tag{21}$$

where $i = 1, 2, \ldots, N$ and N is the number of search results on the SERP.

Gaussian modulation of a search session. As shown in Figure 4, for each search session, the carrier wave of user behavior $\phi(x|\boldsymbol{\tau}, \boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\rho})$ is the superimposition of the Gaussian distribution corresponds to each click event. Each Gaussian distribution of a single click event is weighted by the corresponding click probability ρ_i of the ith search result. Finally, we define the modulator as follows:

$$V_{\text{carrier}} = \text{Modulator}(S_{\text{mst}}), \tag{22}$$

$$= \phi(x|\boldsymbol{\tau}, \boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\rho}), \tag{23}$$

$$= \sum_{i=1}^{N} \rho_i \times \varphi_c(x|\mu_i, \sigma_i^2), \tag{24}$$

$$= \sum_{i=1}^{N} \frac{\rho_i}{\beta\tau_i\sqrt{2\pi}} e^{-\frac{(x-(i-0.5))^2}{2(\beta\tau_i)^2}} \tag{25}$$

where $\rho_i = 1$ if the ith search result is clicked and $\rho_i = 0$ otherwise for the log data.

3.2.3 Simulator: Conditional GAN. As proposed in the work of Mirza and Osindero [53], the GAN can be extended to a conditional model if both the generator and discriminator are conditioned on some auxiliary information, such as class labels or data from other modalities. In this work, we perform the conditioning by feeding the multimedia information of SERPs into both the discriminator and the generator. The feature vector of each search result is concatenated as the representation of the SERP:

$$f_{\text{serp}} = [f_{o,1}, f_{o,2}, \ldots, f_{o,N}], \tag{26}$$

where $[\cdot]$ indicates the concatenation operation and N is the number of search results on the SERP.

Conditional adversarial learning. The simulator consists of a generator G and a discriminator D, which are conditioned on the SERP information f_{serp}. To learn a simulated user behavior distribution Ω_{simulate} over the genuine user behavior distribution Ω_{genuine}, the generator builds a mapping function from a prior noise distribution $p_z(z)$ to the data space Ω_{simulate} conditioned on SERP information f_{serp} as $G(z|f_{\text{serp}}, \theta_g)$, where θ_g represents the parameters of G. The discriminator $D(x|f_{\text{serp}}, \theta_d)$ outputs a single scalar to represent the probability that data point x came from Ω_{genuine} rather than Ω_{simulate}, which is also conditioned on SERP information f_{serp}. G and D are trained simultaneously until the Nash equilibrium of the minimax two-player game is achieved. The standard objective function of the conditional GAN in the work of Mirza and Osindero [53] is defined as follows:

$$\begin{aligned}\min_G \max_D V(D,G) = {} & \mathbb{E}_{x\sim\Omega_{\text{genuine}}(x)}[\log D(x|f_{\text{serp}})]+ \\ & \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z|f_{\text{serp}})))].\end{aligned} \tag{27}$$

However, this objective function may lead to the vanishing gradients problem during the learning process. To overcome this problem, LSGAN [51] proposed the use of the least squares loss function for the discriminator, which has shown a more stable learning process. We adapt the original objective function in LSGAN to be conditioned on auxiliary information as follows:

$$\begin{aligned}\mathcal{L}_D = {} & \frac{1}{2}\mathbb{E}_{x\sim\Omega_{\text{genuine}}(x)}[(D(x|f_{\text{serp}}) - b)^2]+ \\ & \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z|f_{\text{serp}})) - a)^2],\end{aligned} \tag{28}$$

$$\mathcal{L}_G = \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z|f_{\text{serp}})) - c)^2], \tag{29}$$

where a, b, and c are hyperparameters.

The simulator learns from Ω_{genuine} and generates simulated carrier waves of user behavior:

$$\widehat{V}_{\text{carrier}} = \underset{z\sim p_z(z)}{\text{Simulator}}(z, f_{\text{serp}}|\Omega_{\text{genuine}}), \tag{30}$$

$$= G_{z\sim p_z(z)}(z|f_{\text{serp}}, \theta_g). \tag{31}$$

Compared to Equation (12), here the carrier wave that samples Λ from the real environment is Ω_{genuine}, and the condition information ϱ for Conditional GAN includes both the SERP feature f_{serp} and the random noise z.

Generation based on specific SERP. The Conditional GAN can simulate user behavior conditioned on the specific SERP information. However, it is difficult for the generator and the discriminator to learn how a user should behave on a specific SERP because GAN generates data samples based on random noise. As a result, the two players pay more attention on learning how users behave in general when they are searching. To help the generator and the discriminator learn more SERP-specific user behavior, we guide the generator to produce the user behavior that is more similar to

the real one through $L2$ normalization.

$$\mathcal{L}_S = \mathbb{E}_{z\sim p_z(z)}[\ ||V_{\text{carrier}} - \widehat{V}_{\text{carrier}}||_2^2\ ] \tag{32}$$

Thus, the objective of the simulator is to minimize the overall loss $\mathcal{L}$ as follows:

$$\mathcal{L} = \lambda_D \mathcal{L}_D + \lambda_G \mathcal{L}_G + \lambda_S \mathcal{L}_S, \tag{33}$$

where λ_D, λ_G, and λ_S are the weights of $\mathcal{L}_D, \mathcal{L}_G$, and $\mathcal{L}_S$, respectively.

3.2.4 Demodulator: Constrained Optimization. For the simulated carrier wave $\widehat{V}_{\text{carrier}}$, the parameters $\boldsymbol{\tau}, \boldsymbol{\eta}, \boldsymbol{\alpha}, \boldsymbol{\rho}$ in Equation (25) need to be solved to recover the original multidimensional spatio-temporal signal of the user's behavior. Because $\boldsymbol{\eta} = \{\eta_i = i, i = 1, 2, \ldots, N\}$ and $\boldsymbol{\alpha} = \{\alpha_i = 1, i = 1, 2, \ldots, N\}$ are fixed parameters, we only need to find the optimal $\boldsymbol{\tau}^*$ and $\boldsymbol{\rho}^*$ to make the curve $\phi(x|\boldsymbol{\tau}^*, \boldsymbol{\rho}^*, \boldsymbol{\eta}, \boldsymbol{\alpha})$ fit $\widehat{V}_{\text{carrier}}$ best. As the click probability is limited to $\rho_i \in [0, 1]$ and the click time is limited to $\tau_i \in [\tau_{\text{min}}, \tau_{\text{max}}]$, finding the optimal fitting curve can be formulated as a constrained optimization problem. The demodulator is thus defined as follows:

$$\widehat{S}_{\text{mst}} = \text{Demodulator}(\widehat{V}_{\text{carrier}}) = \boldsymbol{\tau}^*, \boldsymbol{\rho}^*. \tag{34}$$

Assuming that the Gaussian distribution for search results on the SERP have little overlap, the generated carrier wave $\widehat{V}_{\text{carrier}}$ can be split into slices corresponding to each search result. For the ith search result on the SERP, the corresponding carrier wave interval is $v_i = \widehat{V}_{\text{carrier},(\text{i}-1)\times \text{s}_\text{f}+1:\text{i}\times \text{s}_\text{f}}$, where s_f (set to 100 in this work) is the sample frequency for the Gaussian distribution of a single search result, and $n_{\text{samples}} = N \times s_f$, N is the number of search results on the SERP.

According to Equation (25) and the assumptions stated previously, for the ith search result,

$$\rho_i^* \approx \int_{i-1}^{i} \frac{\rho_i}{\beta\tau_i\sqrt{2\pi}} e^{-\frac{(x-(i-0.5))^2}{2(\beta\tau_i)^2}} dx, \tag{35}$$

$$= \lim_{s_f \to +\infty} \sum_{k=(i-1)\times s_f+1}^{i\times s_f} \widehat{V}_{\text{carrier},k}. \tag{36}$$

To solve the parameter τ_i while reducing the computation complexity, we limit the candidate values of τ to $\boldsymbol{\tau}_{\text{candidate}} = \{\tau_c = e^{t_c} | t_c = log(\tau_{\text{min}}) + 0.1 \times c, t_c \le log(\tau_{\text{max}}), c \in \mathbb{N}\}$. Thus, $\boldsymbol{\tau}_{\text{candidate}}$ is more sensitive on small values. As a result, for the early clicks with small click time values, the solved τ_i is more accurate. However, for the late clicks, a larger calculation error is acceptable. The candidate Gaussian distribution φ_{τ_c} is then defined as follows:

$$\varphi_{\tau_c}(x) = \frac{1}{\beta\tau_c\sqrt{2\pi}} e^{-\frac{(x-0.5))^2}{2(\beta\tau_c)^2}}, \tag{37}$$

The final click time parameter τ_i is calculated as follows:

$$\tau_i^* = \arg \min_{\tau_c \in \boldsymbol{\tau}_{\text{candidate}}} \delta\left(\varphi_{\tau_c}(x), \frac{v_i}{\rho_i^*}\right), \tag{38}$$

where

$$\delta(\varphi_{\tau_c}(x), \frac{v_i}{\rho_i^*}) = \frac{1}{s_f} \sum_{k=1}^{s_f} \left(\varphi_{\tau_c}(x_k) - \frac{v_{i,k}}{\rho_i^*}\right)^2, \tag{39}$$

where x_k represents the uniformly sampled points from the interval $[0, 1]$ with frequency s_f and $v_{i,k}$ is the kth point in v_i.

4 UBS4RL: RANKING AGENT

To model the interactions among search results, we recast the ranking problem as a sequential filling game (as shown in Figure 1). At the initial timestep, the result list is empty, whereas all of the results are in the candidate set. Based on the query information, one search result is chosen for the first position. Regarding the results already placed in the ranking list, we compare the candidate results and choose the most proper one for the next position iteratively.

The ranking problem can be formulated as an MDP, which is described by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma\}$. $\mathcal{S}$ denotes the state space, and $\mathcal{A}$ denotes the action space. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$ to generate the next state s_{t+1} from the current state s_t and action a_t. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, whereas the reward at the tth timestep $r_t = \mathcal{R}(s_t, a_t)$. $\gamma \in [0, 1]$ is the discounting factor for future rewards. Formally, the MDP components are specified with the following definitions:

State s is the global information of the search results already ranked in the list. At step t, $s_t = \{q, o_{i_j} | j = 1, 2, \ldots, t-1\}$, where o_{i_j} is the search result ranked at position j by the policy. At the initial time, $s_0 = \{q\}$ is the query information.

Action a is a search result that the ranking agent chooses for the next position in the ranking list. At step t, $a_t = o_{i_t}$, where o_{i_t} is the search result ranked at position t by the policy.

Transition $\mathcal{T}$ changes the state of the ranking list, adding one search result to the end of the list at each step.

Reward $\mathcal{R}$ is the reward function that can be the online users' feedback. In this work, the reward is given by the simulation environment based on simulated user feedback. The reward function is introduced in Section 4.2 in detail.

In this article, the MDP problem is solved with the policy gradient algorithm of REINFORCE [68]. At each step t, the policy $\pi(a_t|s_t)$ defines the probability of the sampling action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$. The sampling strategy can be either choosing an action randomly or the one with the highest probability. These two strategies are denoted as "RandomSample" and "MaxSample," respectively. The aim of RL is to learn an optimal policy π^* by maximizing the expected cumulative reward $R_t = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}]$.

Formally, let $S = (q, o_1, o_2, \ldots, o_N)$ denote a query session, where q is the query and o_j is the jth search result in the original ranking list of the search engine, and N is the number of results in the session. The ranking agent re-ranks the result list to $\hat{S} = (q, o_{i_1}, o_{i_2}, \ldots, o_{i_N})$, where $(i_1, i_2, \ldots, i_N)$ is a permutation of $(1, 2, \ldots, N)$. The user simulator samples interaction feedback $U = (u_{i_1}, u_{i_2}, \ldots, u_{i_N})$ on the re-ranked list $\hat{S}$. For CCS, $u_i = \{c_i\}$, whereas for User-GAN, $u_i = \{c_i, \tau_i\}$, where c_i is the click behavior of the ith search result (1 for click and 0 for skip) and τ_i is the click time of the ith search result. The time starts from the moment the user enters the session. The reward at each step is given by `reward` $= \mathcal{R}(U)$.

4.1 Policy Network

The structure of the policy network is shown in Figure 5. The deep neural network architecture can learn policies from high-dimensional raw input data in a complex RL environment. Specifically, the state feature is extracted by a single-direction GRU. The search results that have been placed at proper positions are input to the GRU sequentially according to their rankings. The last hidden state of the GRU is adopted as the state feature for RL. The state feature is then concatenated with the candidate result features and input to a **multi-layer perceptron (MLP)** to assess the probability of each candidate action that be chosen at this step. Then, the action is sampled according to different strategies (RandomSample or MaxSample) as the next result that is added to the ranking

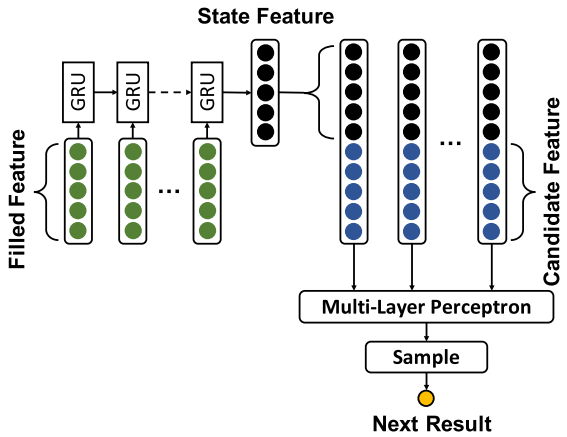

Fig. 5. The policy network for the ranking agent.

list. At step t, there are t and $N-t$ results in the ranking list and candidate set, respectively, where N is the number of results in the query session. The policy network can be formulated as follows:

$$h_k = \text{GRU}(h_{k-1}, f_{o,i_k}), k = 1, 2, \ldots, t, \tag{40}$$

$$f_{\text{state},t} = h_t, \tag{41}$$

$$f_{i_j} = \text{MLP}([f_{\text{state},t}, f_{o,i_j}]), j = t+1, t+2, \ldots, N, \tag{42}$$

$$\pi_t = \text{softmax}(f_{i_{t+1}}, f_{i_{t+2}}, \ldots, f_{i_N}), \tag{43}$$

$$a_t = \text{sample}(\pi_t), \tag{44}$$

where $\{i_1, i_2, \ldots, i_N\}$ is a permutation of $\{1, 2, \ldots, N\}$, MLP denotes the multi-layer perceptron, the sampling strategy can be RandomSample or MaxSample as described earlier, a_t is the chosen action in timestep t, and h_0 is the query feature f_q.

4.2 Reward Design

The user simulator serves as the simulation environment for the ranking agent. Given the re-ranked list of search results, the user simulator predicts user interaction feedback, such as clicks and click times. Numerous online evaluation metrics can be designed as the rewards based on the user feedback. In this work, for user clicks, we adopt a typical online metric (CTR) and two kinds of typical offline evaluation metrics (MRR and DCG (discounted cumulative gain)) [64] for reward design. All of these metrics are modified to utilize the sequential clicks as reward signals and are capable to give rewards at each RL step. The rewards used in the experiments include CTR@3, CTR@5, CTR@10, DCG@3, DCG@5, DCG@10, and MRR. The rewards based on user clicks at timestep t are defined as follows:

$$\textbf{CTR@K:} \quad \text{CTR@K(t)} = \begin{cases} \frac{c_t}{K} & \text{if } t \le K, \\ 0 & \text{else.} \end{cases} \tag{45}$$

$$\textbf{DCG@K:} \quad \text{DCG@K(t)} = \begin{cases} \frac{c_t}{log_2(t+1)} & \text{if } t \le K, \\ 0 & \text{else.} \end{cases} \tag{46}$$

$$\textbf{MRR:} \quad \text{MRR(t)} = \begin{cases} \frac{1}{t} & \text{if the first clicked result is } o_t, \\ 0 & \text{else.} \end{cases} \tag{47}$$

For user click times, we adopt FCT (time to first click), LCT (time to last click), and ACT (average click time) [18] as the rewards. The rewards based on user click times at timestep t are defined as follows:

$$FCT: \quad \text{FCT}(t) = \begin{cases} \frac{1}{\tau_i} & \text{if } t = N, \text{ and the first clicked result is } o_i, \\ 0 & \text{else.} \end{cases} \tag{48}$$

$$LCT: \quad \text{LCT}(t) = \begin{cases} \frac{1}{\tau_i} & \text{if } t = N, \text{ and the last clicked result is } o_i \\ 0 & \text{else.} \end{cases} \tag{49}$$

$$ACT: \quad \text{ACT}(t) = \begin{cases} \frac{\sum_{j=1}^{N} I(c_j=1)}{\sum_{i=1}^{N} I(c_i=1)\tau_i} & \text{if } t = N, \text{ and user clicks at least one result,} \\ 0 & \text{else.} \end{cases} \tag{50}$$

Here, N is the number of results in a query session, $I(c_i = 1) = 1$ if $c_i = 1$ else 0.

4.3 RL Training

A trajectory $\tau = s_0, a_0, s_1, a_1, \ldots, s_N, a_N$ is sampled according to the policy π in each episode. The episode terminates when the ranking list is filled with results. The objective of the training process is to maximize $J(\theta)$.

$$J(\theta) = \mathbb{E}_{\pi_\theta}[R(\tau)] \tag{51}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(\tau) R(\tau)] \tag{52}$$

Equation (52) can be approximated by a Monte Carlo estimator [41]:

$$\nabla_\theta J(\theta) \propto \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{N} \nabla_\theta log \pi_\theta(s_{i,j}, a_{i,j}) R_{i,j}, \tag{53}$$

where θ represents the parameters of the policy network, M is the number of samples, and N is the number of results in a session.

Pretraining of the policy network. It is known that training an RL model from scratch is not effective. As the original ranking list has already achieved a generally satisfactory performance, we pretrain the policy network to output the original rankings. At step t, $s_t = \{o_i | i = 1, 2, \ldots, t-1\}$, $a_t = o_t$. The training objective is to maximum $\pi_\theta(s_t, a_t)$. The policy network is trained in the form of supervised learning.

When training the ranking agent, the parameters of the feature extractor and the user simulator are all fixed to provide a stable environment. In this manner, we can make sure that the training of the ranking agent does not change the simulation environment or the result representations. This is important because if we do not fix the simulation environment, the performance improvement may be caused by the user simulator rather than the ranking agent.

5 EXPERIMENT SETUP

5.1 Datasets

The datasets used in our experiments are sampled from SRR[1] [80]. Each query has corresponding top 10 search results. Each search result is represented by the parse tree and annotated with a 4-graded relevance label. The parse tree of the search result incorporates the images and texts into the HTML codes.

[1] http://www.thuir.cn/data-srr/.

Table 2. Statistics of Real and Simulated Datasets

Dataset		Real Data		Simulation Data	
		Real 2017	Real 2018	Seen	Unseen
#Queries		1,971	1,239	1,569	402
#Logs	Training	1,352,425	–	940,745	241,164
	Validation	451,028	–	315,285	80,806
	Testing	452,864	491,429	312,970	80,030

The two datasets, Real 2017 and Real 2018, were constructed in September 2017 and December 2018, respectively, with search logs recorded by a popular commercial search engine Sogou[2] in China. The queries of Real 2018 are a subset of Real 2017. Search results of the same query in the two datasets are not exactly the same, as time has changed. There are 3.24 results that are the same for each query in the two datasets on average. Real 2018 is used to verify the robustness of different models, so there is only the testing set. Search sessions with click time less than 1 second or more than 600 seconds are discarded, as they are more likely to be from the crawler robot or the noisy behavior, thus $\tau_{\text{min}} = 1$ and $\tau_{\text{max}} = 600$. Search logs are split into training, validation, and testing sets at a ratio of 3:1:1 on Real 2017. Some statistics about the datasets are shown in Table 2.

For the simulation experiments in Section 6.2, we sample synthetic search logs according to the simulation rules in Equations (59) and (60). The queries and results in the Simulation dataset are the same as the Real 2017 dataset. One thousand search sessions are produced by the synthetic user for each query, resulting in 1,971,000 sessions. We split the 1,971 queries and corresponding results as well as synthetic search logs in the Simulation dataset into two parts at a ratio of 4:1 (1,569:402). The two sets are denoted as the Seen Set and the Unseen Set.

5.2 Evaluation Metrics

To measure the approximation accuracy between the predicted clicks and actual clicks, the user simulators are evaluated in terms of perplexity and log-likelihood as in most click model works [13, 21, 25, 32, 50]. Lower perplexity and higher log-likelihood values indicate better approximations. The definitions of the two metrics are as follows:

$$\text{Perplexity}_i = 2^{-\frac{1}{M}\sum_{j=1}^{M}(c_{ji}\log\rho_{ji}+(1-c_{ji})\log(1-\rho_{ji}))}, \tag{54}$$

$$\text{LL} = \frac{1}{MN}\sum_{j=1}^{M}\sum_{i=1}^{N}(c_{ji}\log\rho_{ji} + (1-c_{ji})\log(1-\rho_{ji})), \tag{55}$$

where c_{ji} and ρ_{ji} refer to the binary click in the search log and the predicted click probability $\boldsymbol{\rho}^*$ in Equation (34) or Equation (36) of the ith result in the jth search session, respectively. N is the number of results in the session, and M is the number of sessions in the dataset.

The click-related online evaluation metrics used in our experiments include CTR, DCG, and MRR [18]. DCG and MRR are typical offline evaluation metrics and use relevance annotations for computation. We adapt them to utilize the sequential clicks as the relevance signals (click for relevant and skip for irrelevant). The click time related online evaluation metrics include FCT, LCT, and ACT [18]. We use NDCG [64] for offline evaluation.

5.3 Baselines

- *Click Models*: We compare the simulation performance of the user simulators as well as the ranking performance of the UBS4RL framework with six existing click models, including

[2]https://www.sogou.com/.

five PGM-based click models and one NCM [6]. The five PGM-based click models can be divided into two categories. DCM [32], UBM [25], and DBN [13] do not take the vertical bias into account, whereas UBM-Layout [21] and MCM [50] incorporate the influence of heterogeneous verticals on users' click behavior into the assumptions. The estimated query-document relevance scores (default to 0.5 for unseen query-document pairs) are utilized to rank results.

- *CATM*: CATM [5] is a context-aware time model proposed to predict times between user actions. The authors choose Exponential, Gamma, and Weibull probability density function to describe the time distribution. Different attributes of general interactions, query actions, and click actions are used to represent the search context.
- *Counterfactual LTR*: We compare our proposed UBS4RL framework with three widely adopted counterfactual LTR methods: IPW [37, 73], the REM algorithm [74], and DLA [2]. IPW is one of the first counterfactual LTR methods, which estimates the propensity weight to modify the ranking loss. REM can debias click feedback without relying on randomization. DLA jointly learns an unbiased ranker and an unbiased propensity model. We adopt the implementation of the three counterfactual LTR methods in ULTRA_pytorch.[3] A total of 33 features are extracted as the input to the CLTR methods based on the texts of the landing page corresponding to each search result. We refer the reader to the work of Qin et al. [62] for detailed descriptions of the ranking features.
- *JRE*: JRE [80] is the state-of-the-art neural ranking method on the SRR dataset. However, JRE is trained based on human-annotated relevances. In this work, we only consider user search logs as the available information source for ranking. Thus, we adopt click models as the weak supervision for training JRE. As UBM-Layout performs well on both the Real 2017 and Real 2018 datasets for user feedback simulation as shown later in Table 8, the estimated query-document relevance scores on Real 2017 by UBM-Layout are used as weak labels for training with JRE.

5.4 Experiment Settings

We adopt TreeNN [48] as the feature extractor to represent the heterogeneous search result. The user simulator and the ranking agent both use the result representation extracted by TreeNN. In TreeNN, each search result is represented as a parse tree containing text phrases and images. The parse tree is processed with a recursive neural network. We refer the readers to the work of Liu et al. [48] for more details. The dimension of result features extracted by TreeNN is set to 1,000 in our experiments. For CCS, the dimensions of hidden states in the session level and the result level are all set to 100. For UserGAN, the generator is implemented as a four-layer MLP, whereas the discriminator is a two-layer MLP. The noise vector $z \in \mathbb{R}^{1,000}$ is randomly sampled from the standard Gaussian distribution. We uniformly sample 1,000 data points from the carrier wave for a vectorized representation, as the Gaussian distribution cannot be used directly. Correspondingly, n_{sample} =1,000 and $s_f = 100$. In Equations (28) and (29), a, b, c are set to $0, 1, 1$, respectively. $\lambda_D, \lambda_G, \lambda_S$ are set to $1.0, 1.0, 20.0$, respectively in Equation (33). In Equation (25), the hyperparameter β controls the sensitivity of the carrier wave to the click time $\boldsymbol{\tau}$. Intuitively, smaller values of β make the carrier wave steeper, whereas bigger values of β make it gentler. In the experiments, we set β to $0.01, 0.02, 0.03$ and find that UserGAN is not very sensitive to β; it achieves relatively better performance when $\beta = 0.02$. In the following experiments, β is set to 0.02. The initial learning rates for the generator and discriminator are 10^{-4} and 10^{-5}, respectively. UserGAN is trained up to 10^5 iterations with a batch size setting to 1,000 until convergence.

[3] https://github.com/ULTR-Community/ULTRA_pytorch.

For the ranking agent, the dimension of the hidden states of the policy network is 1,000. The discounting factor γ is set to 0.9. The ranking agent under each reward function is trained up to 500 epochs until convergence. When training the ranking agent, we adopt the RandomSample strategy for π_θ, whereas MaxSample is adopted for the evaluation as we fully trust the learned policy.

The training process of UBS4RL can be divided into three steps. In step 1, the feature extractor is pretrained to learn dense representations of heterogeneous search results as described in the work of Liu et al. [48]. The parameters of the feature extractor are then fixed in subsequent training and testing. In step 2, user simulators are trained with the practical search logs to mimic practical users' behavior. The parameters of the user simulator are then also fixed. In step 3, the ranking agent is trained with the rewards (e.g., clicks and click times) given by the user simulator. The user simulator works as the simulation environment. The training process simulates practical online learning. All of the baselines are trained to achieve the best performance on the validation set.

6 EXPERIMENTS

The proposed UBS4RL framework aims to improve the ranking performance by adopting a user simulator as the simulation environment for the ranking agent. To demonstrate the effectiveness of this framework, we conduct a series of experiments to address the following research questions:

- *RQ1*: Can user simulators capture the important properties of practical search engine users?
- *RQ2*: Can the ranking framework optimize online evaluation metrics effectively in the simulation environment?
- *RQ3*: How does the ranking framework perform on a practical data environment?

6.1 Evaluating User Simulators (RQ1)

6.1.1 *Click Simulation Performance.*

Click probability prediction. To evaluate how different models can predict user click probabilities, we compare them in terms of perplexity and log-likelihood as in most click model work [13, 21, 25, 32, 50]. A lower perplexity and a higher log-likelihood indicate that the user simulator can predict users' click behavior more accurately. The click models and the two user simulators (CCS and UserGAN) are trained using the training set of Real 2017 and tested using the testing set of Real 2017 and Real 2018, respectively. From the results in Table 3, we can note the following.

First, conventional click models perform well in click probability prediction. However, for newly emerging search requests in Real 2018, there is sharp decline in performance, because they can only deal with repeated query-document pairs. This limits the practical application of click models for commercial search engines, because there are millions of emerging search requests conducted every day. It is not practical to train the click models on large-scale search logs frequently.

Second, CCS can achieve comparable performance with click models. CCS predicts user click probabilities based on the context information and previous click as well as result contents. For emerging query-document pairs in Real 2018, CCS can predict the click probability based on the multimodal content information of the search results, which makes its generalization ability stronger.

Third, UserGAN learns the implicit click pattern distribution of users from search logs and exhibits strong generalization ability. Because the click pattern distribution of users is consistent on different queries and documents, UserGAN can perform well on emerging query-document pairs in Real 2018 with just slight performance decline. We can also see that UserGAN cannot outperform the click models for predicting click probabilities. This may be because UserGAN learns both the temporal and spatial signals of click events, which is much more difficult than just learning

Table 3. Click Probability Prediction

Model	Real 2017		Real 2018	
	LL	Perplexity	LL	Perplexity
DCM	−0.1327	1.1528	−0.2255	1.2711
UBM	−0.1282	1.1478	−0.1808	1.2209
DBN	−0.1264	1.1455	**−0.1688**	1.2071
UBM-Layout	−0.1280	1.1476	−0.1808	1.2209
MCM	−0.1255	1.1448	−0.1689	**1.2069**
NCM	**−0.1250**	**1.1442**	−0.1725	1.2291
CCS	−0.1289	1.1489	−0.1717	1.2140
UserGAN	−0.1550	1.1811	−0.1865	1.2351

The perplexity score is the average of Perplexity_i over all positions (top 10). The improvement of the best model (NCM on Real 2017 and DBN/MCM on Real 2018) compared to the others is not significant with a p-value < 0.01.

Table 4. Click Sequence Prediction

Model	CTR@3	CTR@5	CTR@10	DCG@3	DCG@5	DCG@10	MRR
DCM	0.1305	0.0871	0.0524	**0.3347***	**0.3346***	**0.3438**	0.2973
UBM	0.1425	0.1042	0.0734	0.3571	0.3740	0.4118	0.3103
DBN	0.1543	0.1143	0.0684	0.3738	0.4003	0.4231	0.3089
UBM-Layout	0.1455	0.1055	0.0739	0.3605	0.3776	0.4126	0.3091
MCM	0.2183	0.1682	0.1092	0.5101	0.5762	0.6443	0.3429
NCM	0.1661	0.1029	0.0540	0.4319	0.4357	0.4400	0.4131
CCS	0.1503	0.1085	0.0696	0.3803	0.4118	0.4489	**0.2956**
UserGAN	**0.1295**	**0.0841**	**0.0469***	0.3481	0.3529	0.3579	0.3049

The RMSE of the predicted click-related online metrics compared with the logs are shown in the table. Lower RMSE values indicate better performance. The metrics are computed with click sequences generated by practical search engine users ("Log"), different click models, and the two user simulators (CCS and UserGAN). The asterisk (*) denotes the significant improvement compared to the other methods with a p-value < 0.01.

users' click/skip behavior. It needs to not only predict how likely a user is going to click a search result but also when they will click it. As the temporal, spatial, and click probability information are modulated in a single carrier wave using Equation (25), mistakes on the click time prediction also affect the click probability prediction and vice versa.

Click sequence prediction. As revealed in previous studies [18], online evaluation metrics align well with user satisfaction in today's heterogeneous search scenarios. To measure how closely different click models and the two user simulators can simulate users' actual clicks, we evaluate them in terms of online evaluation metrics such as CTR, DCG, and MRR. We train the models on Real 2017 and evaluate them on Real 2018. For each session in the search logs, we sample a click sequence with click models or the user simulator, then compute the online metrics. The **root mean squared error (RMSE)** of the predicted click-related online metrics compared with the search logs are shown in Table 4. Lower RMSE values indicate better performance. From the results, we can note the following.

First, performance of click models varies a lot. Some models, such as DCM and UBM, perform well on click sequence prediction. Others, like MCM and NCM, have large differences with search logs. Second, CCS performs the best on MRR and achieves comparable performance with the click models on the other online metrics, like CTR or DCG. Third, UserGAN performs the best in terms

Table 5. Click Time Prediction

Model	Real 2017			Real 2018		
	FCT	LCT	ACT	FCT	LCT	ACT
CATM	42.2040	44.2649	42.7967	40.8261	41.3245	40.8754
UserGAN	**20.2716***	**24.7261***	**19.7227***	**19.1097***	**21.2440***	**17.9735***

The RMSE of the predicted click time-related online metrics compared with the logs are shown in the table. Lower RMSE values indicate better performance. The asterisk (*) denotes significant improvement compared to the baseline model CATM with a p-value < 0.01.

of CTR@3,5,10, which are among the most widely used online metrics. On DCG@3,5,10, although UserGAN dose not make the best prediction, it achieves very close performance with the best click model DCM and performs secondarily among all of the baselines. On MRR, UserGAN also achieves close performance with the best models, CCS and DCM.

Overall, the online metrics predicted by UserGAN are close with search logs, which indicates its capability in simulating user click behavior. As we use different online metrics as the rewards to train the ranking agent, we will adopt UserGAN as the user simulator in the following experiments.

6.1.2 Time Simulation Performance. Click time is an important aspect to characterize users' fine-grained behaviors as described in the work of Chen et al. [18]. Among all of the click models and the two user simulators, UserGAN is the only one that can simulate user click time corresponding to each click event. We compare UserGAN with CATM (a context-aware time model) [5] (see brief introduction in Section 5.3). As Weibull probability density function performs the best to describe time distributions according to Borisov et al. [5], we adopt Weibull distribution in CATM.

The performance of click time simulation is evaluated in terms of three click time related online metrics: FCT, LCT, and ACT [18]. The RMSE of the predicted click time related online metrics compared with the search logs are shown in Table 5. Lower RMSE values indicate better performance. From the results, we can note that the click time simulated by UserGAN is close to that of search logs. On both Real 2017 and Real 2018 datasets, UserGAN outperforms CATM in terms of FCT, LCT, and ACT. The results show that besides click prediction, UserGAN can also predict the time corresponding to each click event accurately. We will show in the following experiments that the predicted time information will help a lot in both ranking evaluation and optimization.

6.1.3 Online Evaluation with Click Time. For the online evaluation, the user simulator should be able to judge which ranking of search results is better to guide the ranking algorithm optimization. In this section, we analyze whether the click time can help to provide a more accurate online evaluation.

Five click models and UserGAN are trained on Real 2017 and tested on Real 2018. There are a total of 1,239 queries in Real 2018. For each query and the corresponding result list in Real 2018, we try to assess three different ranking strategies: the original ranking of search engine ("Original"), the reversed ranking ("Reverse"), and the randomly shuffled ranking ("Random"). For each query, we compare three pairs of ranking lists: Original with Reverse, Original with Random, and Random with Reverse. Preferences with regard to ranking list pairs are assessed through a commercial crowdsourcing company. Given two different ranking lists (with the same search results) of a query side by side, the assessors need to grade the ranking pair with a five-level preference criterion (from −2 to +2, a higher absolute value indicates stronger preference). For the sake of fairness, result lists are randomly positioned on the two sides. Each ranking list pair has three assessments. The average score is regarded as the final preference (a positive score indicates the right ranking list is better, whereas a negative score indicates the left one is preferred, and zero means the same). The Cohen's κ [23] of preference assessments is 0.6281.

We sample 100 click sequences with UserGAN or click models for each ranking list and compute different online evaluation metrics, including CTR@3,5,10, DCG@3,5,10, and MRR based on the predicted clicks. For UserGAN, each click has a corresponding click time signal. Clicks that occur early may be more valuable to indicate user preference. Thus, click time is adopted as a threshold to filter out less important clicks for UserGAN. The ranking list with a higher predicted online evaluation metric is regarded as the better one. The accuracy of the predicted preference relative to the annotated one is shown in Figure 6. From the results, we can note the following.

First, UserGAN outperforms the click models on Original with Reverse, Original with Random, and Random with Reverse ranking pairs by a large margin in most cases. UserGAN is sensitive to different ranking strategies and can well predict which one users prefer. The click behavior simulated by UserGAN is more differentiated compared to the other click models and thus gives stronger evidence for preference judgment in online evaluation.

Second, the difficulty of judging different ranking strategy pairs is different. The highest accuracy achieved by UserGAN on Original with Reverse is nearly 0.6, whereas it is between 0.56 and 0.58 for Original with Random. The accuracy on Random with Reverse is the lowest, which is approximately 0.45 to 0.48. This shows that it is easier to judge which ranking is better for Original with Reverse pairs because there exists a greater difference. However, the preference between Random with Reverse pairs is not very easy to predict, which may be because both of the rankings have lower quality.

Third, UserGAN is more stable on different evaluation metrics for preference judgments. Click models that perform well based on one evaluation metric may perform poorly on others. For example, UBM-Layout performs the best on CTR@10 among click models. However, it does not achieve the best performance on the other metrics. In contrast, UserGAN is reliable on all metrics in most cases.

Fourth, by jointly considering click/skip behavior and click time, the performance of UserGAN can be improved substantially. For CTR@3,5,10, the accuracy achieved by using an appropriate time threshold compared to using all clicks (click time threshold = 600 seconds) can be improved by 2.88% to 15.50%. For MRR, the best performance is achieved when all the clicks are considered. Different online evaluation metrics have different sensitivity to the click time threshold. This verifies that user clicks are not equally important for online metric evaluation. With click time information, fine-grained user behavior can better indicate user preferences, which has great potential for more accurate online evaluation.

6.1.4 Qualitative Analysis of UserGAN. We randomly sample 10,000 carrier waves from the original search logs and the simulated carrier waves generated by UserGAN of the same search sessions. The distributions of carrier waves are shown in Figure 7. The genuine and simulated carrier waves show similar characteristics. First, the Gaussian distributions are steeper in the top positions of the SERP and gentler toward the bottom. In Equation (25), a smaller click time τ makes the wave steeper, whereas a bigger click time makes it gentler. The distribution indicates that users prefer to click the top-ranked search results at early times; they prefer to click the bottom ones later. This is consistent with previous findings that there exists position bias [29, 30] in the user browsing process. Second, the height of the wave crest decreases by position, which indicates descending click probabilities.

The consistency in click times and click probabilities between the genuine and simulated carrier waves shows that UserGAN has effectively captured the hidden click pattern in the user behavior and can generate similar user behavior based on specific SERP information.

6.1.5 Discussion about Click Models. The traditional click models, such as UBM, DCM, and DBN, have some major limitations. First, traditional click models are mainly based on the PGM

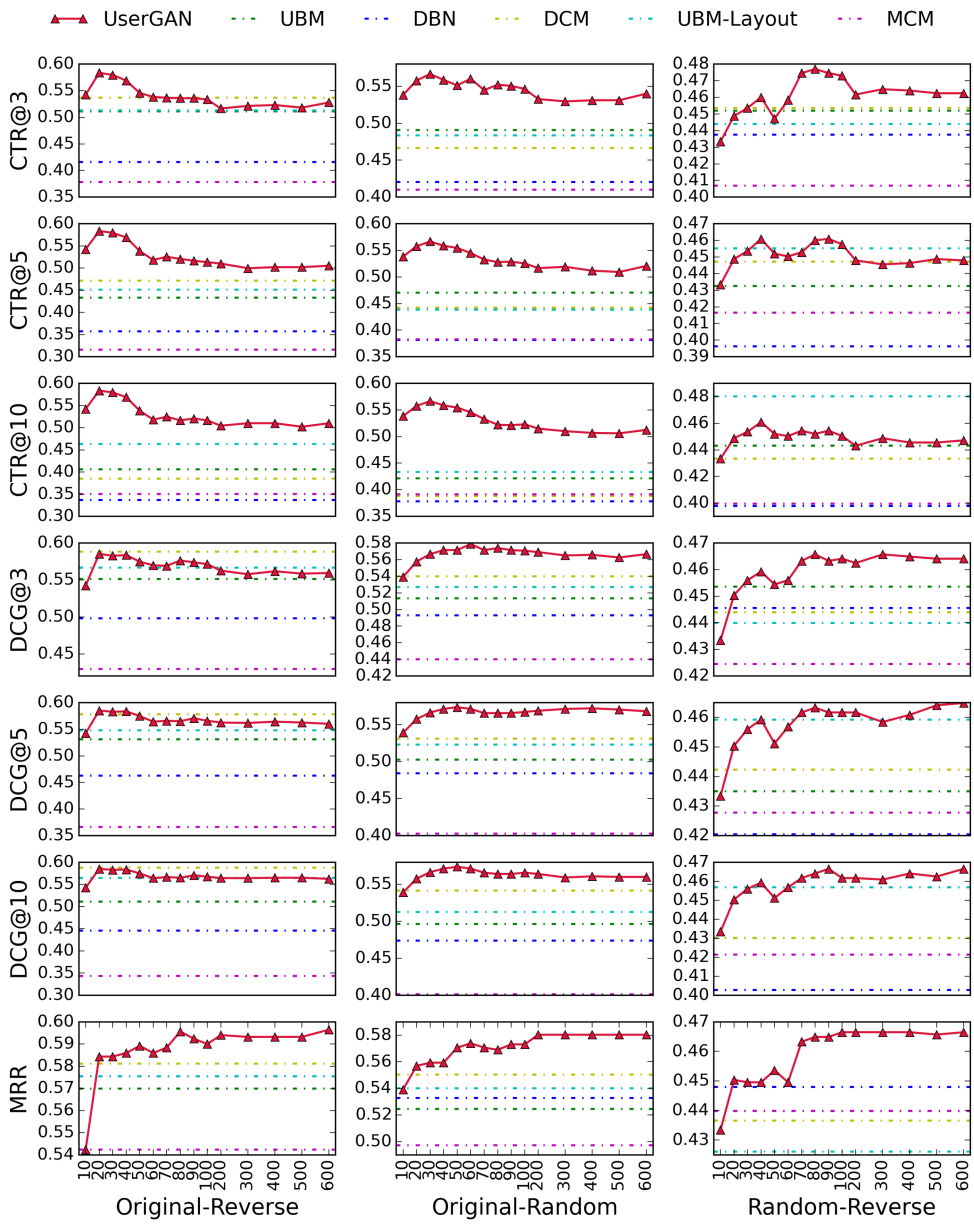

Fig. 6. The accuracy of preference judgments based on different online evaluation metrics, including CTR@3,5,10, DCG@3,5,10, and MRR. Each column of subgraphs shows the comparison between two ranking strategies out of Original, Reverse, and Random. The Y axis indicates accuracy, whereas the X axis indicates the click time threshold (in seconds).

framework and search behavior hypotheses. They use a binary random variable to indicate whether the user clicks or skips a search result and can only predict user click behavior. It is a non-trivial task to incorporate additional random variables to indicate other user behaviors, such as the click time predicted by UserGAN, the viewport time, and user scroll action. Second,

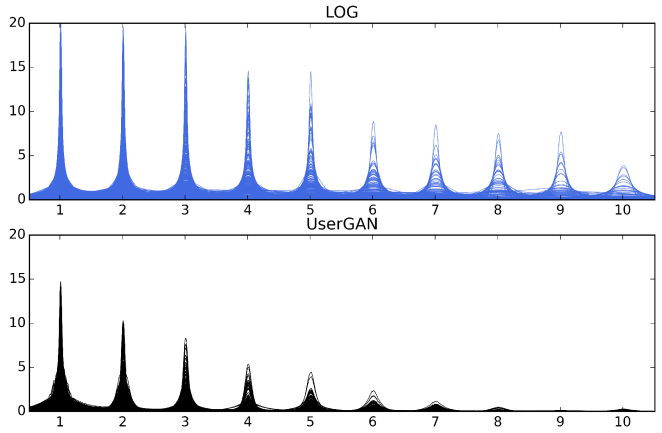

Fig. 7. A total of 10,000 carrier waves sampled from search logs and UserGAN. The X axis indicates the ranking position. A steeper wave shows smaller click time, whereas a gentler wave shows larger one. The higher the wave crest, the larger the click probability.

traditional click models can only deal with repeated query-result pairs. For an unseen query-result pair, they will use default values for prediction. For example, the relevance is set to 0.5 (ranging from 0 to 1) for search results that never appear in historical logs. This makes the traditional click models not applicable in practical search systems, as there are emerging search results every moment on the Web. For a large amount of new search results, the traditional click models will lose efficacy. Third, the contents of search results have changed from pure texts to multimodal information sources in modern search engines, such as texts, images, videos, and applications. The multimodal contents will largely affect user behavior on SERPs. To predict user clicks or click times, it is important to take result contents into consideration. However, the PGM-based click models are incapable of adopting result contents for prediction. In contrast, neural models can deal with multimodal contents of search results well.

6.1.6 Summary for Research Question 1. From the preceding experiments, we can see that the proposed user simulators UserGAN and CCS can capture the important properties of practical search engine users in the following ways:

- For click simulation, UserGAN and CCS can achieve comparable performance in click probability prediction compared with click models. UserGAN achieves superior performance to simulate click sequences, which has close values with search logs on different click-related online evaluation metrics.
- For time simulation, UserGAN can simulate user click times along with clicks at the same time. The simulated click time can help to more accurately distinguish between valuable and noisy click events. By considering time information, we can better judge different ranking strategies, which substantially helps ranking evaluation and optimization.
- By exploiting the consistent distribution of user behaviors from search logs, UserGAN shows strong generalization ability on emerging search requests. This is valuable for search engines dealing with ever-changing information on the Web.

6.2 Evaluation with a Synthetic Dataset (RQ2)

To verify that the proposed UBS4RL framework can optimize a range of online evaluation metrics, we conduct a simulation study with synthetic users, which is similar to those conducted in other

words [2, 75]. A simulation rule is designed according to some prior knowledge and assumptions to generate simulated user feedback based on relevance annotations, vertical types, and statistics of historical search logs. In this way, we create a synthetic user that can (1) generate simulation search logs for original ranking lists and (2) produce `ground truth` user feedback on re-ranked lists to evaluate the effectiveness of the ranking policy.

6.2.1 Simulation Rule. In this work, we extend the simulation rule in the work of Ai et al. [2] to simulate user clicks as well as click times at the same time for heterogeneous search scenarios. The extended rule consists of two parts: click simulation and click time simulation.

Click simulation. As the vertical bias substantially affects the users' browsing process, we incorporate it into the click simulation rule presented in the work of Ai et al. [2]. The extended assumption is that users click a search result o belonging to query q ($c_q^o = 1$) only when it is both estimated ($e_q^o = 1$) and perceived as relevant ($r_q^o = 1$). At the same time, the vertical type is preferred by users to click ($v_q^o = 1$). e_q^o, r_q^o, v_q^o, and c_q^o are all Bernoulli random variables, and we sample these variables by the following formulations.

e_q^o. The position bias η, estimated through eye-tracking experiments presented in the work of Joachims et al. [36], is adopted to sample e_q^o. $\alpha \in [0, +\infty)$ controls the severity of the position biases; it is set to 2.0 in our experiment. i denotes the ranking position of the search result.

$$P(e_q^o = 1) = \eta_i^\alpha \tag{56}$$

r_q^o. The relevance annotations are utilized to sample r_q^o. $y \in [0, 3]$ is the four-level relevance label for result o, and y_{max} is 3 in our dataset. The parameter ϵ introduces click noise into the click decision process. A search result not very relevant to the query also has a small but positive probability to be clicked. ϵ is set to 0.2 in our experiment.

$$P(r_q^o = 1) = \epsilon + (1 - \epsilon)\frac{2^y - 1}{2^{y_{\text{max}}} - 1} \tag{57}$$

v_q^o. The vertical bias ω is estimated on a large number of practical search logs. There are 19 different result types in SRR. The average CTR for each result type is computed as $\omega = \text{AverageCTR}(v)$, where v denotes the vertical type of the search result. The statistical results show that the variances of CTR values belonging to the same type across different queries are small (most of them are less than 0.01). This indicates that the vertical bias holds for a wide variety of searches. $\mu \in [0, +\infty)$ controls the severity of the vertical bias, which is set to 0.5 in our experiment.

$$P(v_q^o = 1) = \omega_v^\mu \tag{58}$$

c_q^o. The final clicks are sampled according to the multiplication of the probabilities of e_q^o, r_q^o, v_q^o. To control the severity of position bias and vertical bias, we need to adjust the hyperparameters α and μ, which can cause small values for $P(e_q^o = 1)$ and $P(v_q^o = 1)$. Thus, $P(e_q^o = 1)$ and $P(v_q^o = 1)$ are linearly mapped to $[0.3, 1]$ in our experiment. The click probability of result o is given by

$$P(c_q^o = 1) = P(e_q^o = 1)P(r_q^o = 1)P(v_q^o = 1). \tag{59}$$

Time simulation. To simulate the click time for each click event, we first analyze click time distributions in historical search logs. For each ranking position, click times $\boldsymbol{\tau}$ in search logs are first mapped to $\boldsymbol{\tau'} \in [0, 1]$ by $\boldsymbol{\tau'} = \frac{log(\boldsymbol{\tau}) - log(\tau_{\text{min}})}{log(\tau_{\text{max}}) - log(\tau_{\text{min}})}$. Then, they are fitted with a Beta distribution [3]. The Beta distribution for ranking position i is denoted as $\mathcal{B}_i$, which is shown in Figure 8. For a search result o ranked at position i, if $c_q^o = 1$, we then randomly sample a click time from $\mathcal{B}_i$ as τ_q^o.

$$\tau_q^o \sim \mathcal{B}_i \tag{60}$$

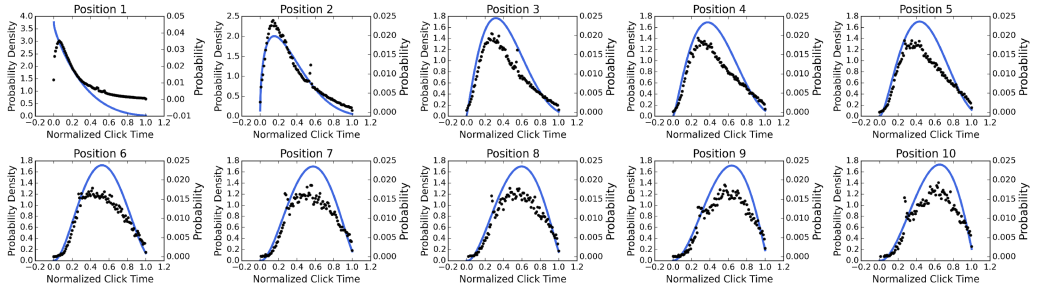

Fig. 8. The Beta distributions fitting click times at each position (from the 1st to the 10th) in search logs. The black dots indicate data points in search logs, whereas the blue line indicates the fitted Beta distribution. The left Y axis shows the probability density of the Beta distribution, whereas the right Y axis shows the statistical probability of different click times.

With Equations (59) and (60), we can sample clicks and corresponding click times at the same time for any given result list. The simulation rule is regarded as a synthetic user, who can give feedback on result lists and generate synthetic search logs.

6.2.2 Online Click Metric Optimization. As the search engine needs to handle both repeated queries and emerging queries that have never appeared in search logs, we train the ranking framework UBS4RL on the Seen Set and subsequently evaluate it on both the previously seen queries (Seen Set) and unseen queries (Unseen Set) as described in Section 5.1. The rule-based synthetic user introduced in Section 6.2.1 is regarded as the practical search engine user. The synthetic user interacts with the re-ranked result list and evaluates the ranking performance in terms of a collection of online metrics to simulate actual online testing.

We also adopt the synthetic user as the simulation environment to simulate actual online training. The UBS4RL ranking framework using the synthetic user for training is denoted as $\text{UBS4RL}_{\text{Oracle}}$. The experiment results of $\text{UBS4RL}_{\text{Oracle}}$ indicate the upper bound of ranking performance using all possible environments (actual online environment and simulation environment) under the adopted RL algorithm. In addition, the number of different rankings of a result list containing N search results is $N!$ By traversing all possible rankings for a result list and comparing the feedback (online evaluation metrics) provided by the synthetic user, we can find the optimal ranking of the search results. The optimal ranking performance is denoted as Oracle. The Oracle result indicates the upper bound of ranking performance achieved by all possible methods.

Table 6 shows the performance of the ranking framework that optimize online metrics on the Seen and Unseen datasets. The first row (denoted as Log) shows the performance of the original ranking lists while the clicks are sampled by the synthetic user (the click labels of the Simulation datasets). $\text{UBS4RL}_{\text{CCS}}$ denotes the ranking performance using CCS as the simulation environment, whereas $\text{UBS4RL}_{\text{UserGAN}}$ denotes the same for UserGAN. From the results, we can note the following.

First, $\text{UBS4RL}_{\text{CCS}}$ and $\text{UBS4RL}_{\text{UserGAN}}$ can all improve the online evaluation metrics over the original rankings by using the objective online metric as the reward, including CTR@3,5,10, DCG3,5,10, and MRR. This shows the potential capability of the UBS4RL ranking framework to optimize a wide range of objectives for commercial search engines as long as a proper reward is designed. $\text{UBS4RL}_{\text{UserGAN}}$ is slightly better than $\text{UBS4RL}_{\text{CCS}}$ on Seen Set and much superior on Unseen Set. The results shows that UserGAN has a stronger generalization ability than CCS. This is because UserGAN directly exploits the click pattern distribution of user behaviors, which is quite consistent in different search scenarios.

Table 6. Performance of Optimizing Click-related Online Metrics

Dataset	Model	CTR			DCG			MRR
		@3	@5	@10	@3	@5	@10	
Seen	Log	0.3191	0.2463	0.1703	0.7458	0.8584	1.0094	0.5896
	$\text{UBS4RL}_{\text{CCS}}$	**0.3420***	0.2614	0.1750	0.7789	0.8994	1.0411	**0.5968***
	$\text{UBS4RL}_{\text{UserGAN}}$	0.3415	**0.2623***	**0.1751***	**0.7855***	**0.9001***	**1.0466***	0.5967
	$\text{UBS4RL}_{\text{Oracle}}$	0.3392	0.2610	0.1744	0.7896	0.9086	1.0473	0.6050
	Oracle	0.3996	0.2968	0.1864	0.9307	1.0480	1.1712	0.6800
Unseen	Log	0.3227	0.2500	0.1730	0.7524	0.8682	1.0218	0.5911
	$\text{UBS4RL}_{\text{CCS}}$	0.3336	0.2522	0.1724	0.7734	0.8859	1.0242	0.5914
	$\text{UBS4RL}_{\text{UserGAN}}$	**0.3505***	**0.2683***	**0.1785***	**0.8079***	**0.9234***	**1.0708***	**0.6033***
	$\text{UBS4RL}_{\text{Oracle}}$	0.3519	0.2670	0.1774	0.8103	0.9267	1.0743	0.6137
	Oracle	0.4088	0.3034	0.1905	0.9516	1.0711	1.1968	0.6898

Results in bold font indicate the better performance between $\text{UBS4RL}_{\text{CCS}}$ and $\text{UBS4RL}_{\text{UserGAN}}$. The asterisk (*) denotes the significant improvement compared to the original ranking (Log) with a p-value < 0.01. The experiments are conducted on the simulation datasets "Seen Set" and "Unseen Set" as described in Section 5.1.

Second, $\text{UBS4RL}_{\text{CCS}}$ and $\text{UBS4RL}_{\text{UserGAN}}$ can achieve a close performance compared to $\text{UBS4RL}_{\text{Oracle}}$. The results show that training the ranking agent with the user simulator is comparable with practical online users. This verifies the effectiveness of the proposed user simulators serving as the simulation environment for RL agent training. We also note that the performance of $\text{UBS4RL}_{\text{UserGAN}}$ and $\text{UBS4RL}_{\text{CCS}}$ sometimes outperforms $\text{UBS4RL}_{\text{Oracle}}$. Although $\text{UBS4RL}_{\text{Oracle}}$ is better than $\text{UBS4RL}_{\text{UserGAN}}$ and $\text{UBS4RL}_{\text{CCS}}$ in theory, the variability in the training and prediction processes introduces some uncertainty.

Third, the Oracle value is much better than $\text{UBS4RL}_{\text{Oracle}}$ as well as $\text{UBS4RL}_{\text{CCS}}$ and $\text{UBS4RL}_{\text{UserGAN}}$. As the user simulator can already compare favorably with practical users for RL training, the results show that there is still large room for improvement of the RL algorithm. This is left for future work to explore more effective RL algorithms into the ranking framework.

6.2.3 Online Time Metric Optimization. UserGAN can simulate user clicks as well as click times. In this section, we optimize some click time related online evaluation metrics, including FCT, LCT, and ACT. The performance is also evaluated by the synthetic user. Figure 9 shows that the click time metrics can be decreased steadily during training on both Seen Set and Unseen Set. Table 7 shows the performance of optimizing FCT, LCT, and ACT. We can see that on Seen Set and Unseen Set, the metrics can all be decreased by approximately 1 second. Smaller FCT values indicate that users tend to make the first click earlier, as they can find the most desired search result more easily. Smaller LCT values indicate that users spend less time on the SERP, as they can obtain the useful information in a relatively short time. Smaller ACT values also show that the browsing process is accelerated, which makes information seeking more efficient. The results show that by optimizing click time related online metrics, the ranking performance is indeed improved.

6.2.4 Summary for Research Question 2. The preceding experiments show that the proposed UBS4RL framework can optimize different online evaluation metrics effectively in the simulation environment:

- UBS4RL can significantly improve click related online evaluation metrics over the original rankings by using the objective evaluation metric as the reward. This shows the potential capability of the UBS4RL ranking framework to optimize a wide range of objectives for commercial search engines as long as a proper reward is designed.

Table 7. Performance of Optimizing Click Time Related Online Metrics (in Seconds)

Model	Seen			Unseen		
	FCT	LCT	ACT	FCT	LCT	ACT
Log	19.4252	68.3609	40.3843	19.1232	69.0774	40.4708
UBS4RL$_{\text{UserGAN}}$	**18.7213***	**67.2422***	**39.3981***	**18.2153***	**67.6186***	**39.2869***

Lower values indicate better performance. The asterisk (*) denotes the significant improvement compared to the original ranking (Log) with a p-value < 0.01. The experiments are conducted on the simulation datasetsSeen Set and Unseen Set as described in Section 5.1.

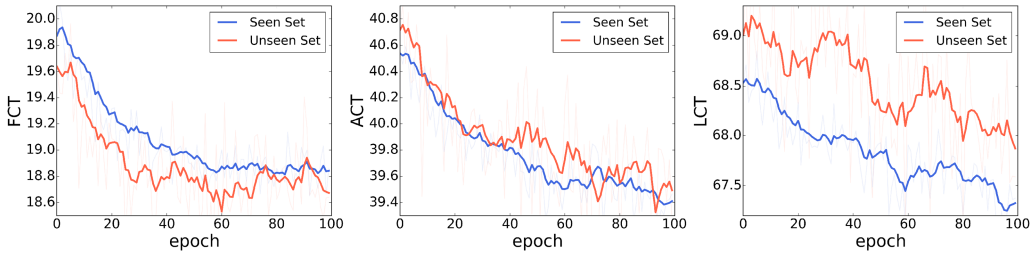

Fig. 9. The performance curves when optimizing the click time related online metrics (in seconds). The experiments are conducted on the simulation datasets Seen Set and Unseen Set as described in Section 5.1.

- By adopting UserGAN as the simulation environment, we can directly optimize click time related online evaluation metrics, such as FCT, LCT, and ACT. The click time metrics can be decreased steadily during training, which makes the user information seeking process more efficient and effective.
- Optimizing UBS4RL in the simulation environment can achieve close performance to that of training with the synthetic user that can be regarded as practical online learning. This shows the effectiveness of the simulation approach for offline training. However, there still exists a gap between UBS4RL and the oracle performance, which is upper bound of all possible ranking methods. This indicates that RL algorithms that are more effective and efficient than REINFORCE can be explored to further improve the performance of UBS4RL.

6.3 Evaluation with a Practical Dataset (RQ3)

As we have demonstrated that the proposed UBS4RL framework works well to optimize online evaluation metrics in the simulation study, we further investigate how it performs in practical data environments, and whether the ranking framework trained in the simulation environment actually helps to improve the offline evaluation metrics. We evaluate the ranking framework on datasets constructed at different times to show its generalizability, which is an important concern for online search systems. The experimental results on the Real 2017 and Real 2018 datasets are shown in Table 8. From the results, we can note the following.

First, click models that directly utilize the debiased query-result relevance for ranking perform the best among the baselines. The counterfactual LTR methods REM, DLA, and IPW, which learn a propensity weight to remove the bias in click logs, can hardly surpass the click models' performance. JRE achieves a performance that is close to UBM-Layout from which the weak relevance labels are obtained.

Second, although UBS4RL aims to optimize the online evaluation metrics, the learned policy also performs well on offline metrics. Under all of the reward functions (including CTR@3, CTR@5, CTR@10, DCG@3, DCG@5, DCG@10, MRR), UBS4RL$_{\text{CCS}}$ and UBS4RL$_{\text{UserGAN}}$ achieve

superior performance than all baselines in terms of NDCG@3,5,10. This shows the robustness of the ranking framework with respect to optimization objectives. The versatility of the ranking framework is potentially important in supporting the heterogeneous search scenarios. $\text{UBS4RL}_{\text{CCS}}$ and $\text{UBS4RL}_{\text{UserGAN}}$ achieve close performance when optimizing click-related online metrics, whereas $\text{UBS4RL}_{\text{UserGAN}}$ is slightly better on Real 2018. This also verifies that learning the implicit user behavior distribution from search logs makes the user simulator more generalizable.

Third, UserGAN is the only user simulator that can mimic user click times. From the results, we can see that optimizing click time related online metrics (FCT, LCT, ACT) can achieve superior performance than optimizing click-related online metrics (CTR, DCG, MRR). Optimizing the FCT metric can achieve the best performance on Real 2017; optimizing the ACT metric can achieve the best performance for Real 2018. This shows that the user click time is an important signal for ranking evaluation, which is usually ignored by previous works. UserGAN simulates fine-grained user behavior including clicks and click times; it performs the best as the simulation environment.

Fourth, UBS4RL has stronger generalization ability than other methods based on search logs. As shown in Table 8, UBS4RL performs much better than DLA, JRE, and click models on Real 2018. The ranking framework trained on historical data performs well on emerging queries and documents, whereas a sharp decline in performance is exhibited for the baselines. Generalization ability is an important concern for search engines as a large amount of Web documents or multimedia contents become available online every moment. It is infeasible to train the model constantly for new contents. The proposed ranking framework UBS4RL is more adaptable for practical search engines.

The overall experiment results show that the simulation environment captures important properties of practical users. It serves as a reliable reward provider and performance judger for the ranking agent. The UBS4RL framework can effectively leverage the implicit relevance signals in search logs, even without making any explicit assumptions on user behavior. UBS4RL can substantially improve the ranking performance by optimizing a wide range of online metrics, including click-related and click time related metrics. The click time information is more effective for ranking evaluation and optimization.

6.3.1 Summary for Research Question 3. By training the user simulators on practical search logs, the proposed UBS4RL framework can effectively improve ranking performance in terms of offline evaluation metrics:

- Although the UBS4RL framework is trained to optimize online evaluation metrics, it also performs well on offline evaluation. UBS4RL achieves superior performance than all of the baselines, including click models, the neural ranking model, and the counterfactual LTR methods.
- When adopting UserGAN as the simulation environment, UBS4RL can achieve stronger generalization ability than the baselines. This verifies that the learned implicit user behavior distribution is consistent over time; thus, they can be transferred to emerging search requests.
- By optimizing click time related online metrics, the UBS4RL framework can achieve even better ranking performance than optimizing click-related online metrics. This shows that the click time information within users' searching process is vital for ranking evaluation and optimization. This time information is usually ignored by previous works.

7 CONCLUSION

Ranking heterogeneous search results with complex interactions is an emerging and critical problem for modern search engines. In this work, we propose a ranking framework UBS4RL based on user simulation to optimize online evaluation metrics. The framework consists of three modules:

Table 8. Offline Performance of Different Models on Real 2017 and Real 2018 Datasets

Model		Real 2017			Real 2018		
		NDCG@3	NDCG@5	NDCG@10	NDCG@3	NDCG@5	NDCG@10
REM		0.7205	0.7849	0.8943	0.6383	0.6870	0.8443
DLA		0.7279	0.7819	0.8951	0.6206	0.6925	0.8416
IPW		0.7382	0.7768	0.8932	0.6542	0.7016	0.8514
JRE		0.7596	0.8046	0.9073	0.6808	0.7275	0.8648
DCM		0.7937	0.8329	0.9223	0.6998	0.7365	0.8762
UBM		0.7883	0.8265	0.9196	0.7006	0.7385	0.8756
DBN		0.7836	0.8208	0.9168	0.6900	0.7292	0.8715
UBM-Layout		0.7883	0.8265	0.9196	0.7025	0.7389	0.8768
MCM		0.7760	0.8170	0.9135	0.6994	0.7354	0.8749
NCM		0.7414	0.7883	0.8992	0.6298	0.6819	0.8434
$\text{UBS4RL}_{\text{CCS}}$	CTR@3	0.8171	0.8566	0.9291	0.7465	0.7851	0.8930
	CTR@5	0.8209	0.8587	0.9301	0.7504	0.7860	0.8934
	CTR@10	**0.8280***	**0.8607***	**0.9315***	**0.7556***	**0.7877***	**0.8945***
	DCG@3	0.8128	0.8556	0.9283	0.7462	0.7848	0.8931
	DCG@5	0.8166	0.8569	0.9290	0.7485	0.7852	0.8934
	DCG@10	0.8141	0.8553	0.9284	0.7449	0.7831	0.8924
	MRR	0.8155	0.8567	0.9288	0.7466	0.7839	0.8927
$\text{UBS4RL}_{\text{UserGAN}}$	CTR@3	0.8203	0.8544	0.9286	0.7723	0.7942	0.8979
	CTR@5	0.8214	0.8552	0.9287	0.7723	0.7936	0.8976
	CTR@10	**0.8237***	**0.8573***	**0.9299***	**0.7736***	0.7942	**0.8982***
	DCG@3	0.8215	0.8552	0.9291	0.7644	0.7930	0.8965
	DCG@5	0.8226	0.8567	0.9295	0.7645	0.7922	0.8961
	DCG@10	0.8217	0.8562	0.9292	0.7637	0.7927	0.8965
	MRR	0.8209	0.8539	0.9285	0.7698	**0.7953***	0.8979
	LCT	0.8058	0.8323	0.9233	0.7904	0.8069	0.9072
	ACT	0.8244	0.8553	0.9311	**0.7935***	**0.8104***	**0.9088***
	FCT	**0.8333***	**0.8641***	**0.9345***	0.7923	0.8092	0.9083

The second column shows the optimization reward (CTR, DCG, MRR, FCT, ACT, LCT) for UBS4RL. The asterisk (*) denotes the significant improvement of UBS4RL under different simulation environments (CCS and UserGAN) and reward functions compared to the baselines with a p-value < 0.01.

feature extractor, user simulator, and ranking agent. Different user simulators are constructed to mimic fine-grained user feedback as the simulation environment for the offline training of the RL agent. Offline training avoids the time and resource consumption issues inherent in online training methods for practical search engines. The online training can also be built upon the policy learned offline with search logs and the simulation environment, which is more efficient and brings little harm to the online system. By extensive experiments on both synthetic and practical data, we demonstrate the effectiveness of the framework in improving ranking performance in terms of online as well as offline evaluation metrics. We also find that the click time information during users' search process is vital for the ranking evaluation and optimization, which is usually ignored by previous works. The ranking framework has a strong generalizability over time, which is valuable for search engines dealing with ever-changing information on the Web.

The framework has the potential to be extended to a wide range of optimization tasks for different information systems, such as mobile search, product search, image search, and recommendation systems. It can be further improved from three aspects. First, the feature extractor can incorporate the information of landing pages, which contain more detailed information of search results, into the multimedia contents of SERPs. Meanwhile, the components of the ranking framework

can utilize different feature extractors in practical applications. For example, the user simulator can be trained offline with highly computational deep neural features to achieve better performance, whereas the ranking agent, which is used to rank the results online, can utilize more cost-efficient handcrafted features. Second, the user simulator can adopt other generative models such as flow-based generative models and VAEs for user feedback simulation. Additional fine-grained spatial and temporal signals of user behavior can also be explored, such as fixations and mouse movements during the users' information seeking process. Third, the ranking agent can leverage a wide range of RL algorithms to improve the ranking performance, which have shown effectiveness and efficiency in other research fields like games and robotics. In addition, different online metrics can be jointly optimized. User satisfaction can also be directly improved through tailored reward designs. The list-wise ranking framework also has the potential to be extended to the two-dimensional whole-page optimization problem, which takes the results on the right side of SERPs into consideration, such as knowledge cards and related searches.

REFERENCES

[1] Keith Jack. 2008. Instant access. In *Digital Video and DSP*. Newnes, Burlington, MA, 223–231. https://doi.org/10.1016/B978-0-7506-8975-5.00009-1

[2] Qingyao Ai, Keping Bi, Luo Cheng, Jiafeng Guo, and W. Bruce Croft. 2018. Unbiased learning to rank with unbiased propensity estimation. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. 385–394.

[3] Robert H. Berk. 1996. *Continuous Univariate Distributions*, Vol. 2. Wiley Series in Probability and Statistics. Wiley.

[4] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A context-aware time model for web search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 205–214.

[5] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A context-aware time model for web search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 205–214.

[6] Alexey Borisov, Ilya Markov, Maarten De Rijke, and Pavel Serdyukov. 2016. A neural click model for web search. In *Proceedings of the International Conference on World Wide Web*.

[7] Alexey Borisov, Martijn Wardenaar, Ilya Markov, and Maarten De Rijke. 2018. A click sequence model for web search. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. 45–54.

[8] Juan C. Caicedo and Svetlana Lazebnik. 2015. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*.

[9] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 335–336.

[10] George Casella and Roger L. Berger. 2002. *Statistical Inference*, Vol. 2. Duxbury, Pacific Grove, CA.

[11] Dong-Kyu Chae, Jin-Soo Kang, Sang-Wook Kim, and Jung-Tae Lee. 2018. CFGAN: A generic collaborative filtering framework based on generative adversarial networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, New York, NY, 137–146.

[12] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM, New York, NY, 621–630.

[13] Olivier Chapelle and Ya Zhang. 2009. A dynamic Bayesian network click model for web search ranking. In *Proceedings of the 18th International Conference on World Wide Web*. ACM, New York, NY, 1–10.

[14] Tong Che, Yanran Li, Ruixiang Zhang, R. Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. 2017. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983* (2017).

[15] Danqi Chen, Weizhu Chen, Haixun Wang, Zheng Chen, and Qiang Yang. 2012. Beyond ten blue links: Enabling user click modeling in federated web search. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining*. ACM, New York, NY, 463–472.

[16] Qin Chen, Qinmin Hu, Jimmy Xiangji Huang, and Liang He. 2018. CAN: Enhancing sentence similarity modeling with collaborative and adversarial network. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 815–824.

[17] Ye Chen, Ke Zhou, Yiqun Liu, Min Zhang, and Shaoping Ma. 2017. Meta-evaluation of online and offline web search evaluation metrics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 15–24.

[18] Ye Chen, Ke Zhou, Yiqun Liu, Min Zhang, and Shaoping Ma. 2017. Meta-evaluation of online and offline web search evaluation metrics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 15–24.

[19] Kyunghyun Cho, Bart Van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of the 8th Workshop on Syntax, Semantics, and Structure in Statistical Translation*. 103–111.

[20] Aleksandr Chuklin, Pavel Serdyukov, and Maarten De Rijke. 2013. Click model-based information retrieval metrics. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 493–502.

[21] Aleksandr Chuklin, Pavel Serdyukov, and Maarten De Rijke. 2013. Using intent information to model user behavior in diversified search. In *Proceedings of the European Conference on Information Retrieval*.

[22] Cyril W. Cleverdon, Jack Mills, and E. Michael Keen. 1966. *Factors Determining the Performance of Indexing Systems; Volume 1, Design; Part 1, Text*. 1: Design. College of Aeronautics, Cranfield.

[23] Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin* 70, 4 (1968), 213.

[24] Ming Ding, Jie Tang, and Jie Zhang. 2018. Semi-supervised learning on graphs with generative adversarial nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, New York, NY, 913–922.

[25] Georges E. Dupret and Benjamin Piwowarski. 2008. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 331–338.

[26] William Fedus, Ian Goodfellow, and Andrew M. Dai. 2018. MaskGAN: Better text generation via filling in the_. *arXiv preprint arXiv:1801.07736* (2018).

[27] Ian Goodfellow. 2016. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).

[28] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*. 2672–2680.

[29] Laura Granka, Matthew Feusner, and Lori Lorigo. 2008. Eyetracking in online search. In *Passive Eye Monitoring*. Springer, 283–304.

[30] Zhiwei Guan and Edward Cutrell. 2007. An eye tracking study of the effect of target rank on web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 417–420.

[31] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. 2017. Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. *arXiv preprint arXiv:1705.10843* (2017).

[32] Fan Guo, Chao Liu, and Yi Min Wang. 2009. Efficient multiple-click models in web search. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*. ACM, New York, NY, 124–131.

[33] David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122* (2018).

[34] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*. 2042–2050.

[35] Yuting Jia, Qinqin Zhang, Weinan Zhang, and Xinbing Wang. 2019. CommunityGAN: Community detection with generative adversarial nets. In *Proceedings of the World Wide Web Conference*. ACM, New York, NY, 784–794.

[36] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2005. Accurately interpreting clickthrough data as implicit feedback. *ACM SIGIR Forum* 51, 1 (2005), 4–11.

[37] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*. 781–789.

[38] Junbo, Zhao, Yoon Kim, Kelly Zhang, Alexander Rush, and Yann LeCun. 2017. Adversarially regularized autoencoders for generating discrete structures. *arXiv preprint arXiv:1706.04223* (2017).

[39] Huang A. Maddison, C. J. Guez, A. D. Silver, and D. Hassabis. 2016. Mastering the game of Go with deep neural 1139 networks and tree search. *Nature* 529, 7587, 484–489.

[40] Diederik P. Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114* (2013).

[41] Levente Kocsis and Csaba Szepesvari. 2006. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*. Lecture Notes in Computer Science, Vol. 4212. Springer, 282–293.

[42] Dieter Kraft. 1988. *A Software Package for Sequential Quadratic Programming*. In Technical Report DFVLR-FB 88-28. DLR German Aerospace Center—Institute for Flight Mechanics, Koln, Germany.

[43] Sang-Gil Lee, Uiwon Hwang, Seonwoo Min, and Sungroh Yoon. 2017. Polyphonic music generation with sequence generative adversarial networks. *arXiv preprint arXiv:1710.11418* (2017).
[44] Shangsong Liang. 2019. Unsupervised semantic generative adversarial networks for expert retrieval. In *Proceedings of the World Wide Web Conference*. ACM, New York, NY, 1039–1050.
[45] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. 2017. Adversarial ranking for language generation. In *Advances in Neural Information Processing Systems*. 3155–3165.
[46] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. 2017. Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems*. 700–708.
[47] Yiqun Liu, Xiaohui Xie, Chao Wang, Jian-Yun Nie, Min Zhang, and Shaoping Ma. 2016. Time-aware click model. *ACM Transactions on Information Systems* 35, 3 (2016), 1–24.
[48] Yiqun Liu, Junqi Zhang, Jiaxin Mao, Min Zhang, Shaoping Ma, Qi Tian, Yanxiong Lu, and Leyu Lin. 2019. Search result reranking with visual and structure information sources. *ACM Transactions on Information Systems* 37, 3 (June 2019), Article 38, 38 pages. https://doi.org/10.1145/3329188
[49] Shuqi Lu, Zhicheng Dou, Xu Jun, Jian-Yun Nie, and Ji-Rong Wen. 2019. PSGAN: A minimax game for personalized search with limited and noisy click data. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 555–564.
[50] Jiaxin Mao, Cheng Luo, Min Zhang, and Shaoping Ma. 2018. Constructing click models for mobile search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY.
[51] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. 2017. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2794–2802.
[52] David Maxwell and Leif Azzopardi. 2016. Agents, simulated users and humans: An analysis of performance and behaviour. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. 731–740.
[53] Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).
[54] Alistair Moffat and Justin Zobel. 2008. Rank-biased precision for measurement of retrieval. *ACM Transactions on Information Systems* 27, 1 (2008), 1–27.
[55] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. 2018. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701-07274* (2018).
[56] Andrew Y. Ng and Stuart Russell. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*.
[57] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).
[58] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759* (2016).
[59] Harrie Oosterhuis and Maarten de Rijke. 2018. Differentiable unbiased online learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1293–1302.
[60] Harrie Oosterhuis and Maarten De Rijke. 2018. Ranking for relevance and display preferences in complex presentation layouts. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. 845–854.
[61] 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 *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 499–508.
[62] Tao Qin, Tie Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval* 13, 4 (2010), 346–374.
[63] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).
[64] Kalervo Rvelin, Kek, and Jaana Inen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.
[65] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. 2016. Multileave gradient descent for fast online learning to rank. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining*. 457–466.
[66] Chengyao Shen and Qi Zhao. 2014. Webpage saliency. In *Proceedings of the European Conference on Computer Vision*. 33–46.
[67] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2018. Virtual-Taobao: Virtualizing real-world online retail environment for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 845–854.

[68] Richard S. Sutton and Andrew G. Barto. 2011. Reinforcement learning: An introduction. MIT Press.
[69] Pei Hao Su, Milica Gasic, Nikola Mrksic, Lina M. Rojas Barahona, Stefan Ultes, David Vandyke, Tsung Hsien Wen, and Steve Young. 2016. On-line active reward learning for policy optimisation in spoken dialogue systems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2431–2441.
[70] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Andrei A. Rusu, Veness Joel, Marc G. Bellemare, Graves Alex, Riedmiller Martin, Andreas K. Fidjeland, and Ostrovski Georg. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
[71] Chao Wang, Yiqun Liu, Min Zhang, Shaoping Ma, Meihong Zheng, Jing Qian, and Kuo Zhang. 2013. Incorporating vertical results into search click models. In *Proceedings of the 36th international ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 503–512.
[72] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 515–524.
[73] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 115–124.
[74] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position bias estimation for unbiased learning to rank in personal search. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. 610–618.
[75] Yue Wang, Dawei Yin, Luo Jie, Pengyuan Wang, Makoto Yamada, Yi Chang, and Qiaozhu Mei. 2018. Optimizing whole-page presentation for web search. *ACM Transactions on the Web* 12, 3 (2018), 19.
[76] Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2017. Reinforcement learning to rank with Markov decision process. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 945–948.
[77] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. 2017. Adapting Markov decision process for search result diversification. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*.
[78] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence generative adversarial nets with policy gradient. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.
[79] Fangyi Zhang, Juergen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. 2015. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791* (2015).
[80] Junqi Zhang, Yiqun Liu, Shaoping Ma, and Qi Tian. 2018. Relevance estimation with multiple information sources on search engine result pages. In *Proceedings of the 2018 ACM on Conference on Information and Knowledge Management*. ACM, New York, NY, 1–10.
[81] Junqi Zhang, Jiaxin Mao, Yiqun Liu, Ruizhe Zhang, Min Zhang, Shaoping Ma, Jun Xu, and Qi Tian. 2019. Context-aware ranking by constructing a virtual environment for reinforcement learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, New York, NY, 1603–1612.
[82] Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial feature matching for text generation. In *Proceedings of the 34th International Conference on Machine Learning—Volume 70*. 4006–4015.
[83] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2223–2232.
[84] Yadong Zhu, Yanyan Lan, Jiafeng Guo, Xueqi Cheng, and Shuzi Niu. 2014. Learning for search result diversification. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Received 16 December 2020; revised 2 November 2021; accepted 10 January 2022