RLMixer: A Reinforcement Learning Approach For Integrated Ranking With Contrastive User Preference Modeling

Jing Wang[1]*, Mengchen Zhao[2]*, Wei Xia[2], Zhenhua Dong[2], Ruiming Tang[2], Rui Zhang[3], Jianye Hao[2,4], Guangyong Chen[5]✉, and Pheng-Ann Heng[1]

[1] The Chinese University of Hong Kong, Hong Kong
{jing}@link.cuhk.edu.hk, {pheng}@cse.cuhk.edu.hk
[2] Huawei Noah's Ark Lab
{zhaomengchen, xiawei24, dongzhenhua, tangruiming, haojianye}@huawei.com
[3] www.ruizhang.info {rayteam}@yeah.net
[4] Tianjin University, Tianjin, China
[5] Zhejiang Lab, Zhejiang, China {gychen}@zhejianglab.com

Abstract. There is a strong need for industrial recommender systems to output an integrated ranking of items from different categories, such as video and news, to maximize overall user satisfaction. Integrated ranking faces two critical challenges. First, there is no universal metric to evaluate the contribution of each item due to the huge discrepancies between items. Second, user's short-term preference may shift fast between diverse items during her interaction with the recommender system. To address the above challenges, we propose a reinforcement learning (RL) based framework called RLMixer to approach the sequential integrated ranking problem. Benefiting from the credit assignment mechanism, RLMixer can decompose the overall user satisfaction to items of different categories, so that they are comparable. To capture the user's short-term preference, RLMixer explicitly learns user interest vectors by a carefully designed contrastive loss. In addition, RLMixer is trained in a fully offline manner for the convenience in industrial applications. We show that RLMixer significantly outperforms various baselines on both public PRM datasets and industrial datasets collected from a widely used AppStore. We also conduct online A/B tests on millions of users through the AppStore. The results show that RLMixer brings over 4% significant revenue gain.

Keywords: Integrated ranking, Reinforcement Learning, Contrastive Learning

1 introduction

Traditional ranking systems focus on ranking homogeneous items, such as a list of news, according to a specific metric like click-through rate. However, in practice, the final recommendation result presented to a user is usually a mixture of heterogeneous items. For example, in the news feeds scenarios, the recommendation list might consist of news, videos, advertisements and various form of cards. A straightforward way is to fix some slots for specific categories,

* The first two authors contributed equally to this work.

which is commonly adopted in industrial recommender systems. However, this is clearly not the optimal strategy since the user's preferences towards each category evolve during interaction with the recommender system.

The optimal integrated ranking module is required to rank items of different categories in order to maximize the overall utility of the recommender system. The unique challenges of integrated ranking are two-fold. First, we lack a unified metric to evaluate the qualities of items from different categories. Thus they cannot be compared in one dimension directly. Second, users' preferences towards each category could be essentially different and shift during the interaction. For example, a user might find an interesting video while reading news and keep looking for similar videos. This personalized short-term preference shifting is hard to capture since the user feedbacks are usually implicit.

To address the first challenge, existing works try to use reinforcement learning to allocate categories to slots [11]. Unfortunately, their model focuses on inserting advertisements to news feed and does not generalize to integrated ranking with multiple categories of items. To address the second challenge, various methods have been proposed to learn representations of users' short-term interests [20, 19]. However, they focus on implicitly mining knowledge from recent interacted items without explicitly modeling user preferences.

In this paper, we propose RLMixer for general integrated ranking problems. The novelties of RLMixer lie in the following three aspects. First, we propose a general and flexible MDP formulation that covers a broad range of integrated ranking problems. Second, we explicitly model the user's short-term preferences towards different categories and propose a carefully designed contrastive loss for learning them. Third, RLMixer can be trained fully offline, significantly saving online exploration costs and avoiding bias caused by simulation. Specifically, we implement RLMixer by conservative-q learning along with divergence penalty.

As far as we know, RLMixer is the first offline reinforcement learning approach to solve general integrated ranking problems. We successfully tackled the aforementioned issues with appropriate MDP modeling and a novel offline training framework. We compare RLMixer with several baselines on the public PRM datasets, as well as industrial datasets. We also deploy RLMixer on a widely used AppStore, where apps from different sources and categories are ranked together. Experimental results show that RLMixer significantly improves the original ranking quality and brings over 4% revenue gain.

2 Related Work

2.1 Integrated Ranking

Integrated ranking focuses on reranking items based on the roughly mixed heterogeneous items list while lacking a unified metric, while existing work mainly focuses on allocating advertisements to a list of organic items, which can be regarded as a particular case of integrated ranking. Koutsopoulos [7] defines ads allocation as a shortest-path problem on a weighted directed acyclic graph and apply the Bellman-Ford algorithm to solve it. Yan et al. [16] propose a uniform formula to rank advertisements and organic items together, considering the impact of interval between them. Zhao et al. [18] propose a novel deep Q-network to

determine when and how to interpolate advertisements. Liao et al. [11] propose Cross-DQN to extract the crucial arrangement signal by crossing the embeddings of different items and modeling the crossed sequence by multi-channel attention. Unfortunately, existing works consider only advertisements and organic items, which limits their application in general integrated ranking problems with more than two categories of items. Moreover, their methods require online or off-policy training, which might incur huge online exploration costs.

2.2 Offline RL for Recommendation

Reinforcement learning for recommendation systems has attracted increasing interest in recent years. Zheng et al. [6] propose DRN for news recommendation, an off-policy framework with an online exploring network to balance exploration and exploitation. Chen et al. [2] propose a policy gradient method with various techniques to reduce the variance of policy gradients. Zhao et al. [17] propose a DDPG-based algorithm for learning optimal ranking weights to combat cheating sellers in e-commerce. However, existing work directly applies RL without considering how the user's preferences evolve during the interaction.

Inspired by the abundant historical interactions in recommendation scenarios, offline reinforcement learning is an emerging topic that aims to learn agent policy purely from dataset [4, 8–10]. Offline RL strives for the issue of the overestimation of out-of-distribution actions, which introduces significant extrapolation error in policy learning. A popular method to address this is to use behavioral regularizations in RL training that compel the learned policy to stay close to the offline data. These regularizations consist of incorporating some divergence regularization into the critic [9], policy divergence penalties [14, 4], and appropriate network initializations [12]. Regarding its application in recommendation, Xiao et al. [15] summarize several offline learning tricks and demonstrate their effectiveness in recommendation.

3 Integrated Ranking via Reinforcement Learning

In this section, we formally define the problem formulation for integrated ranking with the reinforcement learning settings.

Integrated ranking serves as a re-ranking module in the whole chain of the recommendation system, and aims to output a re-ranked list that maximizes the overall utility of the system. Figure 1 illustrates our decision making process with three categories of items. Due to the huge combinatorial action space of processing the whole list at the same time, we re-rank items within a sliding window among the original list step by step.

Integrated ranking is naturally a sequential decision making problem, we model the integrated ranking as a Markov Decision Process $< \mathcal{S}, \mathcal{A}, r, \mathcal{P}, \gamma >$:

State space $\mathcal{S}$: $\mathcal{S}$ is the set of states describing the state space of the integrated ranking module. A state $s \in \mathcal{S}$ consists of the user information (e.g., age, gender, purchasing power), the originally ranked list, candidate items(i.e., items needed to be re-ranked at current step) and other contextual information.

Action space $\mathcal{A}$: An action $a \in \mathcal{A}$ is a sequence of categories whose length is the size of sliding window. Assume there are C related categories in total. An

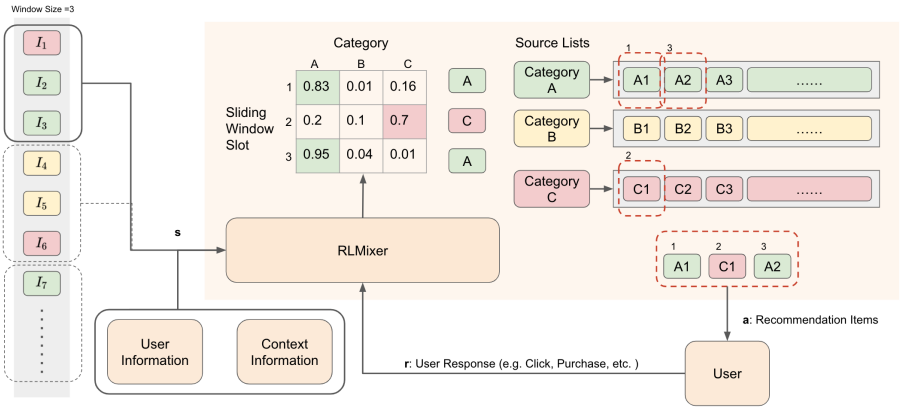

Fig. 1. Illustration of integrated ranking three categories of items with window size 3. The policy of RLMixer output a 3*3 matrix, where the rows represent the slots to be filled and the columns represent categories. During execution, the category with the highest score in each row is selected. Then the item that ranked highest in the corresponding category is fetched and filled in the slot.

action can be represented as a vector $a = (C_1, C_2, ..., C_W)$, where W is the size of the sliding window and $C_i \in \{1, 2, ..., C\}$ indicates its category.

Rewards: The reward is calculated based on the system's overall utility, which is the accumulated revenue(e.g., price) of cliked items in our case.

Transitions: $P(s_{t+1}|s_t, a_t)$ is the state transition function that indicates the state transferring from current state s_t to next state s_{t+1} after taking action a_t. Note that such updates on the raw states do not actually reflect the change of user preference. This motivates us to learn a mapping from raw states to explicit user preference representation. Please refer to Section 4.3 for details.

The optimal policy of the integrated ranking agent maximizes the system's total expected reward:

$$J = E_{\tau\sim\pi}\left[\sum_{t=0}^{\tau}\gamma^t r_t(s_t, a_t)\right], \tag{1}$$

where $\gamma \in [0, 1]$ is a discount factor and $t \in \tau$ is the discrete time step in the trajectory τ.

In the integrated ranking scenarios, the dataset $\mathcal{D}$ may include various kinds of user's feedbacks of different types of items. The goal of offline reinforcement learning is to learn a policy directly from $\mathcal{D}$, in order to maximize the expected cumulative discounted reward Equation (1). Actor-critic scheme is a classical framework for solving MDPs dynamically. It maintains a parametric Q-function, Q_θ, and a parametric policy, $\pi_\omega(a|s)$. It alternates between policy evaluation, computing the Q^π that iterating the Bellman operator $\mathcal{B}^\pi Q = r + \gamma E_{\mathbf{s}'\sim P(\mathbf{s}'|\mathbf{s},\mathbf{a}),\mathbf{a}'\sim\pi(\mathbf{a}'|\mathbf{s}')}\left[Q\left(\mathbf{s}', \mathbf{a}'\right)\right]$, and policy improvement, improving the policy $\pi(a|s)$ by updating it towards actions that maximize the expected Q-value. In this paper, we incorporate behavior regularization into the actor-critic framework via a critic penalty and policy regularization to address the overestimation and distribution shift. Details of training are illustrated in Section 4.5.

4 The Framework of RLMixer

4.1 Overview

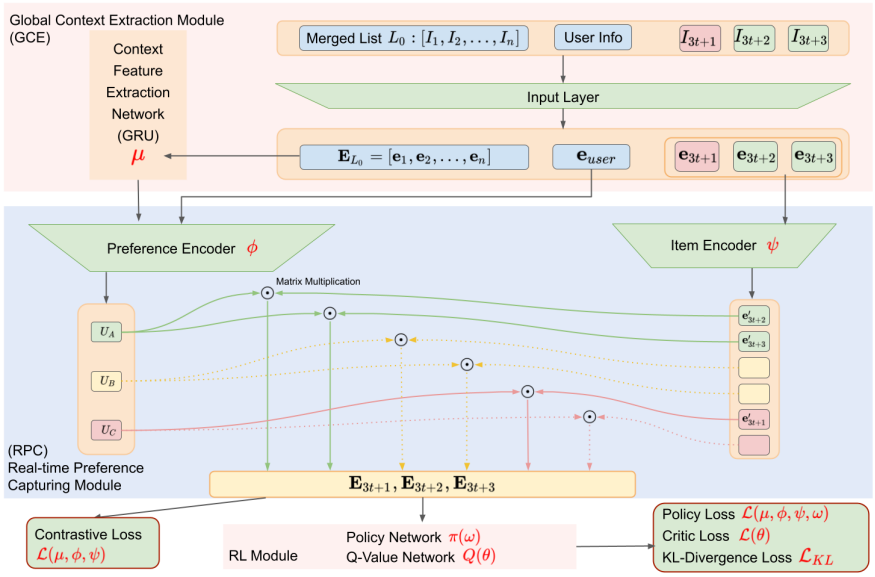

Fig. 2. Overall architecture of the policy network in RLMixer.

The architecture of the policy network in RLMixer is presented in the Figure 2. In order to capture local information for decision-making at every single step, we propose to maintain a sliding window that contains the current items to be re-ranked, which can also be interpreted as the user's current attention. And then, the global context extraction (GCE) module is expected to extract the context information from the original ranking list, the real-time preference capturing (RPC) module is utilized to learn the real-time user preferences on candidate items respectively. Finally, the user's preference on candidate items is concatenated together and fed into the RL module for policy execution.

In the following sections, we will take a sliding window size of 3 as an example to elaborate on implementing the aforementioned modules and networks.

4.2 Global Context Extraction Module

In the training stage, we take the original ranking list $L_0 = [I_1, I_2, ..., I_n]$ as part of the state information, where n is the total number of items in source lists. Let $W_t = [I_{3t+1}, I_{3t+2}, I_{3t+3}]$ denote the candidate items inside the sliding window W_t at time step t. As introduced in the section 3, the full state input consists of the user information(e.g., age, gender, purchasing power, etc.), the original ranking list L_0, and candidate items W_t. Initially, we employ the input layer to map the user information features to the user embedding vector $\mathbf{e}_{user}$, and obtain the item embedding $\mathbf{e}_i$ for each item I_i. Then we adopt the traditional GRU cells to extract the contextual information from the original ranking list through context feature extraction network μ, $\mathbf{h}_i = GRU(\mathbf{e}_i)$, where $GRU(\cdot)$ denotes the traditional GRU cell, $\mathbf{h}_i$ denote the hidden state about item I_i.

After feature extraction, we construct the overall embedded state information $\hat{\mathbf{s}} = [\mathbf{e}_{user}, \mathbf{e}_{3t+1}, \mathbf{e}_{3t+2}, \mathbf{e}_{3t+3}, \mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_n]$, and then feed it into the RPC module to model the user's real-time preferences toward different categories.

4.3 Real-time Preference Capturing Module

Preference Encoder ϕ The preference encoder employs the embedded state information $\hat{\mathbf{s}}$ to model the user preference embedding matrices. Assume there are C (i.e., 3 categories in our scenario) categorical lists that need to be reranked, we have the user real-time preference overall matrix $\mathbf{U} = [\mathbf{U}_1, \mathbf{U}_2, ..., \mathbf{U}_c]$ learned by the network ϕ, where $\mathbf{U}_c \in R^{d_{\text{pref}} \times d_{\text{emb}}}$ is the User Preference Matrix related to Category $\mathbf{c}$ respectively. The number of rows d_{pref} is the preference depth expected to be learned of the category, and the number of columns d_{emb} is the embedding dimension of each preference aspect.

Item Encoder ψ At the same time, we employ an embedding network ψ to utilize sliding window item embedding $\mathbf{E}_{W_t} = [\mathbf{e}_{3t+1}, \mathbf{e}_{3t+2}, \mathbf{e}_{3t+3}]$ to dig further features of the items within the sliding window. We expect to extract profound item features that are related to the its own categorical characteristics tightly through the deep item embedding network. Then we have $\mathbf{E}'_{W_t} = [\mathbf{e}'_{3t+1}, \mathbf{e}'_{3t+2}, \mathbf{e}'_{3t+3}]$ denote the item profound embedding, where $\mathbf{e}'_i \in R^{d_{emb}}$.

Item-wise Preference Calculation Let $C_i = \text{Category}(I_i)$ denote the category type for each item I_i. We design the matrix multiplication $\mathbf{E}_i = \mathbf{U}_{C_i}\mathbf{e}'_i$ between the user real-time preference $\mathbf{U}_{C_i}$ towards the corresponding category of item I_i and the item profound embedding $\mathbf{e}'_i$, to capture the user preference $\mathbf{E}_i \in R^{d_{\text{pref}} \times 1}$ towards the exactly candidate item. Then we feed the exact learned state information

$$\mathbf{s} = [\mathbf{E}_{3t+1}, \mathbf{E}_{3t+1}, \mathbf{E}_{3t+3}] \tag{2}$$

to the reinforcement learning network to get the final prediction.

In conclusion, the real-time preference capturing module help to unify a comparable embedding scheme for different category items to benefit further development in the reinforcement learning module.

4.4 Contrastive User Preference Modeling

To provide a supervision signal for strengthening the learning of user preferences, we propose an auxiliary contrastive user preference loss. Inspired by the fact that user has common interests among different items implicitly, we believe that items clicked by the same user has common interests factor. Hence, we divide items in the sliding window to two sets $S_{clicked}$ and $S_{unclicked}$. We expect the similarity between the user-item interests embedding of user clicked items and unclicked items to be far away as much as possible, and of the same set items to be closed. Then we optimize the contrastive loss:

$$\mathcal{L}_C(\mu, \phi, \psi) = -\frac{\sum_{i \in S_{clicked}} \sum_{j \in S_{unclicked}} e^{d(\mathbf{E}_i, \mathbf{E}_j)}}{\sum_{i,j \in S_{clicked}} e^{d(\mathbf{E}_i, \mathbf{E}_j)} + \sum_{i,j \in S_{unclicked}} e^{d(\mathbf{E}_i, \mathbf{E}_j)}} \tag{3}$$

$, \forall \mathbf{E}_i, \mathbf{E}_j \in \{\mathbf{E}_{3t+1}, \mathbf{E}_{3t+2}, \mathbf{E}_{3t+3}\}$, where d is the similarity calculation function such as the Euclidean distance or cosine similarity distance. We adopt the square of Euclidean distance in our work.

With the help of the auxiliary contrastive user-item preference, we suppose to learn representations of the user-item interests regardless of the category.

4.5 Conservative Offline Reinforcement Learning

We adopt the actor-critic framework in our RL module, which includes a policy network $\pi(\omega)$ and a Q-function network $Q(\theta)$. Both networks inherit the output of the RPC module as input, the learned state information $\mathbf{s}$ from Eq.(2). In order to address the overestimation of the OOD actions, we extend the conservative Q-learning [9] to our scenario with further policy divergence regularization.

Conservative Policy Evaluation: In consistent with the CQL, we penalize the Q function at states in the dataset for actions not observed in the dataset. Then the Q function associated with the current policy π is conservatively updated by the following optimization function:

$$\begin{aligned}\mathcal{L}_Q(\theta) = \lambda \left(E_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi(\cdot|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - E_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[Q(\mathbf{s},\mathbf{a})]\right) \\ + \frac{1}{2} E_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \widehat{\mathcal{B}}^{\pi} Q(\mathbf{s},\mathbf{a})\right)^2\right].\end{aligned} \tag{4}$$

The $\widehat{\mathcal{B}}^{\pi} Q(\mathbf{s},\mathbf{a}) := r(\mathbf{s},\mathbf{a}) + \gamma Q'(\mathbf{s}',\mathbf{a}')$ is the empirical bellman operator that only backs up a single sample, where $(\mathbf{s},\mathbf{a},\mathbf{s}')$ is a single transition from the given dataset, $\mathbf{a}' \sim \pi(\cdot|\mathbf{s}')$, Q'_θ is the target Q Network which has the same structure of Q_θ and is substituted by Q_θ network periodically. The second term in the Eq.(4) is the conventional loss that minimizes the squared error of the target Q value and prediction Q value. Significantly, the first term in the Eq.(4) enables a conservative estimation of the value function for learned policy to mitigate the overestimation bias.

Conservative Policy Improvement with divergence penalty: The goal of policy learning is to give prediction towards action that maximizes the expected Q value. With the help of a conservative critic Q, the policy network ω is improved by the optimization function:

$$\mathcal{L}_\pi(\mu,\phi,\psi,\omega) = -E_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi(\cdot|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right]. \tag{5}$$

In order to address the distributional shift challenge in the offline setting, we utilize the following KL-Divergence loss as regularization.

$$\mathcal{L}_{KL} = E_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')} D_{KL}(\pi(\cdot|\mathbf{s})||\pi_\beta(\cdot|\mathbf{s})). \tag{6}$$

This regularization aims to constrain the bound of the state distributional shift between the learned policy $\pi(\cdot|\mathbf{s})$ and the behavior policy $\pi_\beta(\cdot|\mathbf{s})$. However, the behavior policy $\pi_\beta(\cdot|\mathbf{s})$ is often a mixture of multiple policies due to the complex online logic. Note that in our scenario, the policy actually outputs a distribution of categories at each time step. With this observation, we could recover an approximate behavior policy by simply calculating the distribution of categories in the logged trajectories. Specifically, we calculate the overall category distribution from the real dataset $q = [q_1, q_2, ..., q_C]$. We denote the category distribution inferred from the training policy by $p = [p_1, p_2, ..., p_C]$. Then we can reformulate the KL-regularizert as Equation 7. This reformulated regularizer implicitly pushes our learned policy $\pi(\cdot|\mathbf{s})$ to be close to the behavior policy $\pi_\beta(\cdot|\mathbf{s})$.

$$\mathcal{L}_{KL} = D_{KL}(p||q) \tag{7}$$

Implementation Details During the training stage, the Q-function and policy network are updated separately. We train the policy network with the GCE and RPC module networks together through the gradient passed by the learned state

information **s** while we stop the gradient of the input **s** for the Q-function network training. In summary, we alternately train networks between the loss function Eq.(4) and Eq.(8).

$$\mathcal{L} = \mathcal{L}_\pi(\mu, \phi, \psi, \omega) + \alpha\mathcal{L}_C(\mu, \phi, \psi) + \beta\mathcal{L}_{KL}, \tag{8}$$

where the α and β are the hyper-parameters to adjust the weight of each loss.

5 Experiments

We present both offline and online evaluation results of RLMixer. In the offline evaluation, we compare RLMixer with existing baselines on the public PRM datasets and industrial datasets collected from an industrial AppStore. In the online experiments, we deploy RLMixer to provide re-ranking services to the industrial AppStore and conduct online A/B testing.

5.1 Offline Experiment Setting

Datasets. We give a detailed description of the two datasets as follows.

PRM dataset. We adopt the public PRM dataset released by [13], which is a large-scale dataset (E-commerce Re-ranking dataset) built from a real-world E-commerce recommender system. The dataset includes a huge number of sessions that record interactions between the recommender system and users. For each session, features (e.g., category, identity, price, etc.) of a recommendation list items recommended to a user and the corresponding user click-through response are stored. To avoid significant variance, we keep the interactions between the user and three main (i.e., most frequently presented) category items recommended by the system with primal orders presented in the recommendation list, which also matches the integrated ranking application scenario.

Industrial dataset. We collected a real-world dataset from an industrial AppStore platform for 15 consecutive days. It contains similar user and item features as the PRM dataset, with two additional high-level categories.

The statistics of two offline datasets are presented in the Table 1. We split each dataset into training and test sets with a ratio of 4:1.

Table 1. Statistics of two offline datasets

	Sessions	Users	Category A	Category B	Category C
PRM dataset	7,919,659	605,668	1,411,185	291,629	311,364
Industrial dataset	194,233	184,443	9,018	4,771	-

Baselines We compare RLMixer with the following representative methods.

Original. The primal recommendation list is presented in the original dataset.

MMR[1]. Maximal Marginal Relevance(MMR) is a ranking algorithm that allows controlling the diversity and the relevance of provided information.

LinkedIn-Det[5]. LinkedIn-Det proposes several deterministic algorithms for fair re-ranking of top-K results based on desired proportions over one or more protected attributes.

DHCRS[3]. Deep Hierarchical Category-based Recommender System utilizes a high-level DQN to select a category and then a low-level DQN to choose an item in this category. Due to the order preservation constraints in our integrated ranking scenario, we implement the category-level structure of DHCRS, and add the KL-Divergence loss to make it more suitable to the offline setting.

Evaluation Metrics The aim of an optimal integrated ranking system is to maximize the revenue for the platform. Accordingly, we adopt utility and $\alpha-$utility metrics to evaluate the performance of our method instead of NDCG. Furthermore, as mentioned in section 5.1, the diversity or we call the ratio of each category information, is also important to the integrated problem. We introduce the ratio metric to make all algorithms fit into similar comparable levels.

NDCG@K (Normalized Discounted Cumulative Gain) is a measure of ranking quality in information retrieval area. It evaluates the quality of the recommendation list by calculating the fraction of Discounted Cumulative Gain (e.g., the click signal in our scenario) over the Ideal Discounted Cumulative Gain.

utility@K is the average utility of a session with regard to the top-K recommended items. The utility of a single item is related to the section 3, we then formulate the calculation of $utility@K$ as the Eq.(9).

$$utility@K = \frac{1}{|\mathcal{S}|}\sum_{s\in S}\sum_{i=1}^{K} price(I_{s,i}) * Click(I_{s,i}), \tag{9}$$

where $\mathcal{S}$ is the set of sessions, $I_{s,i}$ is the i-th recommended item in the session s, $Click(I_{s,i})$ is the click signal indicating whether the item $I_{s,i}$ is clicked.

$\alpha-$**utility@K** is a metric that considers category information and evaluates whether the utility is balanced among different categories,

$$\alpha - utility@K = \frac{1}{|\mathcal{S}|}\sum_{s\in S}\sum_{i=1}^{K} \alpha^{N_{category(I_{s,i})}} price(I_{s,i}) * Click(I_{s,i}), \tag{10}$$

where α is the discounted factor, and $N_{(category(I_{s,i}))}$ is the total counts of category $category(I_{s,i})$ appears in the session s.

ratio@K is used to evaluate the distribution of each category among all items. The ratio among two categories C_i and C_j only is defined as follows:

$$\underset{i,j}{ratio}@K = \frac{\sum_{s\in S}\sum_{k=1}^{K} I(category(I_{s,k}) = C_i)}{\sum_{s\in S}\sum_{k=1}^{K} I(category(I_{s,k}) = C_j)}, \tag{11}$$

where I is the indicator function.

To compute the evaluate the distribution of category when the number of categories larger than two, we elaborate the rotate ratio to roughly represent the average distribution in the following:

$$ratio@K = \frac{\sum_{i=1}^{|C|-1} ratio_{i,i+1}@K + ratio_{|C|,1}@K}{|C|}, \tag{12}$$

where $C = C_1, C_2, ..., C_n$ is the set of n categories supposed to be constrained.

5.2 Offline Results and Analysis

Results on PRM dataset. We conduct experiments with our model and other baseline models on the PRM dataset and focus on the top-10 performance of the recommendation list since the average session length is 30. We first compute the basic

Table 2. Evaluation on PRM dataset.

Top-10	ratio 2.3 ± 10%			
	NDCG	utility	$\alpha-$utility	ratio
original	0.5114	0.0339	0.0106	2.3124
MMR	0.4088	0.0386	0.0144	2.1436
LInkedINn-Det	0.4213	0.0354	0.0130	2.2883
DHCRS	0.4158	0.0385	0.0096	2.1522
RLMixer	0.4003	**0.0471**	**0.0203**	2.4239

statistics of the testing set, which is presented in the Table 2 as original baseline. To evaluate variant methods fairly, we set the target desired ratio 2.3 during model training, which is the approximation of the ratio metric in original recommendation among dataset. And then we select the best model according to the utility performance of top-10 when the ratio of model predictions within the range of $2.3 \pm \%10$. The performance results are presented in the Table 2.

- Compared our RLMixer with other baselines, RLMixer outperforms them in both utility and $\alpha-$utility metrics, which is at least 22% and 44% higher than others respectively.
- Considering both MMR and Linkedin-Det are ranking algorithm related to balancing the diversity and utility, our RLMixer outperforms these two algorithms even though we were bound into the same level ratio. It shows that our algorithm can be applied to the scenario to earn profit much higher while fulfilling the desired distribution requirements.
- We achieve better performance than DHCRS even though DHCRS is a RL-based method and giving the category prediction first as well. This indicates that our real-time user preference capturing module and corresponding auxiliary contrastive loss design might contribute a lot to the final prediction. We will discuss this later in the ablation study.
- The NDCG value of the original list is the highest, but it brings the lowest utility. The reason is that the computation of NDCG uses only click signals, ignoring the real values of each click. Therefore, compared with NDCG, the utility-related metrics better align with the online performance. This gives us an intuition that pursuing the most clicks may not be the best strategy.

Results on industrial dataset. Since users are presented with at least 7 items at a time, we compare top-20 results of RLMixer with the original rank on the industrial dataset. Table 3 shows their performance comparisons. Similar to the PRM dataset, our RLMixer leads to a lower NDCG value but is compensated by a 1.7% utility gain.

Table 3. Evaluation on Industrial dataset.

Top-20	ratio $0.5 \pm 10\%$			
	NDCG	utility	$\alpha-$utility	ratio
Original	0.3659	3.056	1.152	0.5284
RLMixer	0.3455	**3.108**	**1.341**	0.5405

Ablation Study To verify the impact of Real-time Preference Capturing(RPC) module and the auxiliary contrastive loss, we conduct two sets of experiments on the public dataset PRM. The two sets of experiments train RLMixer without the entire RPC module or auxiliary contrastive loss, respectively. We still constraint the desired ratio within the range of $2.3 \pm \%10$ during the model training, and we select the best model based on top-10 performance. Then we evaluate top-3, top-5, and top-10 performance of the

Table 4. Ablation study of contrastive user preference modeling in RLMixers on PRM dataset.

Method	Top-K	NDCG	utility	$\alpha-$utility	ratio
RLMixer	3	0.2575	**0.0246**	**0.0216**	2.4061
	5	0.3140	**0.0302**	**0.0211**	2.0444
	10	0.4003	**0.0471**	**0.0203**	2.4239
RLMixer **w/o** RPC	3	0.2585	0.0192	0.0159	2.3125
	5	0.3144	0.0288	0.0166	2.5253
	10	0.4010	0.0446	0.0151	2.4495
RLMixer **w/o** Contrastive Loss	3	0.2564	0.0214	0.0171	2.1858
	5	0.3123	0.0296	0.0165	1.9008
	10	0.3997	0.0437	0.0150	2.1805

best model of each RLMixer variant. As shown in the Table 4, the full RLMixer comprehensively presents higher performance in utility and $\alpha-$utility metrics of all top-k levels, while compared with the variants without RPC module or auxiliary contrastive loss. Especially, it can be observed that the cooperation of RPC module and auxiliary loss brings key improvements to the primal algorithm over other baseline algorithms.

5.3 Online A/B test

The online setting is follow that of offline experiments, and the quantified criteria of the A/B experiment is to compare the revenue(i.e., utility) with the baseline in three days. We deploy RLMixer online and compare its performance with the fine-tuned rule-based model that is currently deployed online. Noted that the goal of our method is to adjust the ratio to a certain target value while maximizing the utility. Table 5 shows the regularized average utility obtained during three consecutive days. We find that RLMixer achieves 4.05% utility gain compared with the current model, which demonstrated the effectiveness of our method.

Table 5. The results of A/B experiments.

Policy	Baseline	RLMixer	Impr
utility	0.0074	0.0077	**4.05%**

6 Conclusion

We propose a general offline RL framework with contrastive user preference modeling called RLMixer for integrated ranking problems. With the aid of the Global Context Extraction (GCE) module and Real-time Preference Capturing (RPC) Module, RLMixer is able to synthesize values of items from different categories and capture the user's short-term preference shifting. Furthermore, it incorporates behavior regularization into the actor-critic framework to address the distribution shift problem that exists in the offline setting. We compare RLMixer with existing baselines on the public PRM datasets and datasets collected from an industrial AppStore. We also deploy RLMixer to provide re-ranking services to an industrial AppStore and conduct an online A/B test, which shows that RLMixer brings 4.05% utility gain.

References

1. Carbonell, J., Goldstein, J.: 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. pp. 335–336 (1998)
2. Chen, M., Beutel, A., Covington, P., Jain, S., Belletti, F., Chi, E.H.: Top-k off-policy correction for a reinforce recommender system. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. pp. 456–464 (2019)
3. Fu, M., Agrawal, A., Irissappane, A.A., Zhang, J., Huang, L., Qu, H.: Deep reinforcement learning framework for category-based item recommendation. IEEE Transactions on Cybernetics (2021)
4. Fujimoto, S., Meger, D., Precup, D.: Off-policy deep reinforcement learning without exploration. In: International Conference on Machine Learning. pp. 2052–2062 (2019)

5. Geyik, S.C., Ambler, S., Kenthapadi, K.: Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. In: Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining. pp. 2221–2231 (2019)
6. Guanjie, Z., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N.J., Xie, X., Li, Z.: Drn: A deep reinforcement learning framework for news recommendation. In: Proceedings of the 2018 World Wide Web Conference. pp. 167–176 (2018)
7. Koutsopoulos, I.: Optimal advertisement allocation in online social media feeds. In: Proceedings of the 8th ACM international workshop on hot topics in planet-scale mObile computing and online social neTworking. pp. 43–48 (2016)
8. Kumar, A., Fu, J., Soh, M., Tucker, G., Levine, S.: Stabilizing off-policy q-learning via bootstrapping error reduction. In: Advances in Neural Information Processing Systems. pp. 11761–11771 (2019)
9. Kumar, A., Zhou, A., Tucker, G., Levine, S.: Conservative q-learning for offline reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 1179–1191 (2020)
10. Levine, S., Kumar, A., Tucker, G., Fu, J.: Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643 (2020)
11. Liao, G., Wang, Z., Wu, X., Shi, X., Zhang, C., Wang, Y., Wang, X., Wang, D.: Cross dqn: Cross deep q network for ads allocation in feed. arXiv preprint arXiv:2109.04353 (2021)
12. Matsushima, T., Furuta, H., Matsuo, Y., Nachum, O., Gu, S.S.: Deployment-efficient reinforcement learning via model-based offline optimization. In: International Conference on Learning Representations (2021)
13. Pei, C., Zhang, Y., Zhang, Y., Sun, F., Lin, X., Sun, H., Wu, J., Jiang, P., Ge, J., Ou, W., et al.: Personalized re-ranking for recommendation. In: Proceedings of the 13th ACM conference on recommender systems. pp. 3–11 (2019)
14. Wu, Y., Tucker, G., Nachum, O.: Behavior regularized offline reinforcement learning. arXiv preprint arXiv:1911.11361 (2019)
15. Xiao, T., Wang, D.: A general offline reinforcement learning framework for interactive recommendation. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (2021)
16. Yan, J., Xu, Z., Tiwana, B., Chatterjee, S.: Ads allocation in feed via constrained optimization. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining. pp. 3386–3394 (2020)
17. Zhao, M., Li, Z., Bo, A., Haifeng, L., Yifan, Y., Chen, C.: Impression allocation for combating fraud in e-commerce via deep reinforcement learning with action norm penalty. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. pp. 3940–3946 (2018)
18. Zhao, X., Gu, C., Zhang, H., Yang, X., Liu, X., Tang, J., Liu, H.: Dear: Deep reinforcement learning for online advertising impression in recommender systems. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence. pp. 750–758 (2021)
19. Zhou, G., Mou, N., Fan, Y., Pi, Q., Bian, W., Zhou, C., Zhu, X., Gai, K.: Deep interest evolution network for click-through rate prediction. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 5941–5948 (2019)
20. Zhou, G., Zhu, X., Chenru, S., Ying, F., Han, Z., Xiao, M., Yanghui, Y., Junqi, J., Han, L., Gai, K.: Deep interest network for click-through rate prediction. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining. pp. 1059–1068 (2018)