# Building Personalized Simulator for Interactive Search

**Qianlong Liu**[1] , **Baoliang Cui**[2] , **Zhongyu Wei**[1*] , **Baolin Peng**[3] , **Haikuan Huang**[2] , **Hongbo Deng**[2] , **Jianye Hao**[4] , **Xuanjing Huang**[1] and **Kam-fai Wong**[3]
[1]Fudan University, China
[2]Alibaba Group, China
[3]The Chinese University of Hong Kong, Hong Kong
[4]Tianjin University, China
liuql17@fudan.edu.cn, moqing.cbl@taobao.com, zywei@fudan.edu.cn,
blpeng@se.cuhk.edu.hk, haikuan.hhk@alibaba-inc.com, dhb167148@alibaba-inc.com,
jianye.hao@tju.edu.cn, xjhuang@fudan.edu.cn, kfwong@se.cuhk.edu.hk

## Abstract

Interactive search, where a set of tags is recommended to users together with search results at each turn, is an effective way to guide users to identify their information need. It is a classical sequential decision problem and the reinforcement learning based agent can be introduced as a solution. The training of the agent can be divided into two stages, i.e., offline and online. Existing reinforcement learning based systems tend to perform the offline training in a supervised way based on historical labeled data while the online training is performed via reinforcement learning algorithms based on interactions with real users. The mis-match between online and offline training leads to a cold-start problem for the online usage of the agent. To address this issue, we propose to employ a simulator to mimic the environment for the offline training of the agent. Users' profiles are considered to build a personalized simulator, besides, model-based approach is used to train the simulator and is able to use the data efficiently. Experimental results based on real-world dataset demonstrate the effectiveness of our agent and personalized simulator.

## 1 Introduction

Online shopping platforms, such as Amazon, Taobao, and eBay, have dramatically changed people's living style. The volume of products on these platforms has grown tremendously in recent years, which escalate the need for personalized products recommendation. Extensive machine learning algorithms have been studied [Li *et al.*, 2011; Clark, 2015; Yin *et al.*, 2016; Liu *et al.*, 2017] for automatic product retrieval and recommendation. However, existing research tends to treat the problem as a static one without modeling interactions with users. The shopping process in complicated because users might not be certain with their need in the beginning, therefore, a single-step solution might not be enough

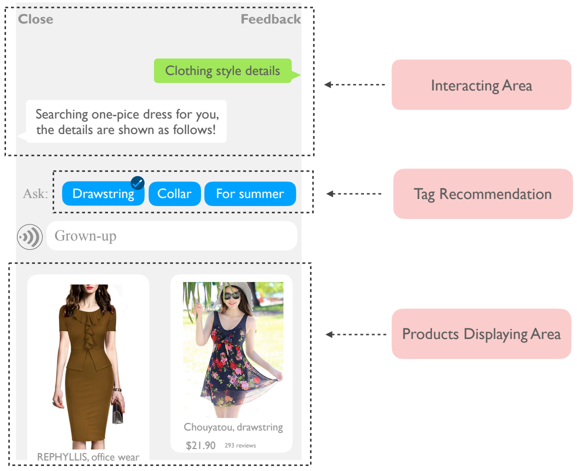

Figure 1: An illustration for interactive search.

to meet users' requirement. In order to improve the shopping experience and help users identify their need quickly, interactive search is emerging. An illustration of interactive search can be seen in Figure 1. At each turn, the system presents a set of tags related to the product together with search results based on user queries. A user can either type new keywords or click any tag provided to update the request. Based on users' feedback, a new set of ranked products and tags would be returned.

It usually involves several steps for users to identify their need. Therefore, interactive search can be naturally formulated as a sequential decision problem. Recently, researchers have explored to utilize Reinforcement Learning (RL) based agent for product recommendation and information retrieval [Taghipour and Kardan, 2008; Zhao *et al.*, 2018; Zheng *et al.*, 2018; Choi *et al.*, 2018]. The training of these models can be divided into two stages, namely, online and offline. Existing research usually trains the model in a supervised fashion offline based on historical data while the online learning is performed by interacting with real users via RL algorithms. Though promising results have been reported, the

[*]Corresponding Author: zywei@fudan.edu.cn

optimization objectives of offline training and online learning are different, which leads to a cold-start problem for the online usage of the agent. In order to address this issue, we propose to design a simulator to interact with the agent for its offline training.

Designing an environment simulator for online shopping platform is challenging for two reasons. First, different users have different preferences. Second, the labeled data for the offline training is usually limited. How to train the agent efficiently becomes a problem. In order to tackle these two challenges, we introduce a personalized, model-based environment simulator, which takes the users' profiles into consideration when mimicking the behavior of real users. In summary, our contribution is three-fold:

- We propose to employ an environment simulator for the offline training of the agent in interactive search.
- We propose a multi-task learning approach to model personalized user preferences based on real user profiles. Besides, we utilize a model-based RL algorithm to make full use of limited labeled data.
- Experimental results on a real-world dataset show the effectiveness of our simulator. By estimating the state transition probability, the collected dataset can be used efficiently via our simulator and the agent converges faster.

## 2 Task Formulation

### 2.1 Task

In interactive search (Figure 1), a search session is initialized by a user with his/her first query. At each turn $t$, the agent returns a set of ranked products (lower part) based on query $q_t$. Meanwhile, the agent receives a set of candidate tags $\mathcal{T}_t = \{\mathtt{t}_1, \cdots, \mathtt{t}_{n_t^{\mathtt{t}}}\}$[1], where $n_t^{\mathtt{t}}$ is the number of candidate tags at turn $t$, and a subset of $K$ (where $K \leq |\mathcal{T}_t|$) ranked tags (middle part) are recommended to help user search desired products. If the displayed products cannot meet the user's demand , the user can either click on one of the displayed tags or type keywords to update his/her query for a better search result. Here are some notations used in this paper:

- User $u$. To model different preferences of users, each user is represented as a vector $u \in \mathcal{U}$ with three features, i.e., the age, gender and purchase power.
- Query $q$. Each query is represented as a sequence of words $q = (w_1, \cdots, w_{n^q}) \in \mathcal{Q}$, where $w_i$ is the $i$-th word in $q$ and $n^q$ is the number of words in $q$.
- Tag $\mathtt{t}$. Each tag $\mathtt{t} = (w_1, \cdots, w_{n^{\mathtt{t}}})$ is also a sequence of words, where $n^{\mathtt{t}}$ is the number of words in tag $\mathtt{t}$.

### 2.2 MDP Formulation

We cast this problem as a Markov Decision Process (MDP) from the agent's perspective. An MDP consists of a tuple of five elements $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$ which are defined as follows:

[1]In this work, candidate tags are derived from logs via information retrieval based methods, and tag generation is not concerned.

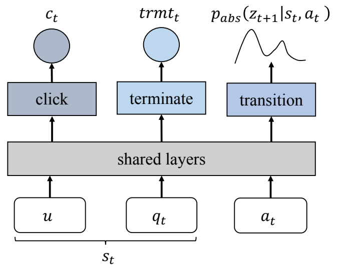

Figure 2: Our multi-task model-based simulator, including three sub-tasks of *clicking* (binary classification), *terminating* (binary classification) and *transition* (multi-classes classification).

**State space $\mathcal{S}$.** A state $s_t = (u, q_t) \in \mathcal{S}$, where $u \in \mathcal{U}$ is the user who initialized this session and $q_t$ is the query at turn $t$. After the user clicks on a tag or types several new keywords, the state will be updated to $s_{t+1} = (u, q_{t+1})$, where the $q_{t+1}$ is updated by appending the new keywords or the words of the clicked tag to $q_t$.

**Action space $\mathcal{A}$.** At turn $t$, an action $a_t = (\mathtt{t}_t^1, \cdots, \mathtt{t}_t^K) \in \mathcal{A}(s_t)$ is a sequence of $K$ tags displayed to the user at turn $t$. Given a set of candidate tags $\mathcal{T}_t$ at turn $t$, the action space $\mathcal{A}(s_t)$ is the set of all permutations of $K$ tags chosen from $\mathcal{T}_t$.

**Reward $\mathcal{R}$.** After the agent takes an action $a_t$ given a state $s_t$, i.e., displaying a sequence of tags to the user, the user can click one of the tags or type in several keywords. The agent will receive an immediate reward $r_t = \mathcal{R}(s_t, a_t)$ according to the feedback of the user $u$.

**Transition probability $\mathcal{P}$.** $p(s_{t+1}|s_t, a_t)$ is the probability of state transition from $s_t$ to $s_{t+1}$ after the agent takes an action $a_t$ at state $s_t$.

**Discount factor $\gamma$.** $\gamma \in [0, 1]$ defines the discount factor when we measure the present value of future reward.

In RL, the policy $\pi$ describes the behavior of an agent, which takes the state $s_t$ as input and outputs the probability distribution over all possible actions $\pi(a_t|s_t),\ \forall a_t \in \mathcal{A}(s_t)$. The agent can be trained via RL-based algorithms, where its goal is to find an optimal policy $\pi$ which maximizes the discounted cumulative reward.

## 3 Environment Simulator

In our task, there are three tasks that the environment needs to consider at each turn, i.e., returning a feedback to the agent, terminating or continuing the current session and transiting to a new state. Our simulator is thus designed as Figure 2 that combines three sub-tasks:

- **Click:** This sub-task predicts whether one of displayed tags will be clicked by the real user or not. According to the predicted result, the immediate reward will be returned to the agent.
- **Terminate:** This sub-task decides to terminate or continue this current search session.
- **Transition:** At each turn, the state will transit to a new state and this sub-task models the transitions between states.

As shown in Figure 2, the lower layers are shared across all three tasks, while the top layers are task-specific.

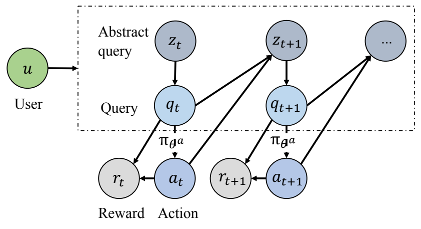

Figure 3: State transition with abstract query.

**Shared layer.** At each turn, the simulator takes the state $s = (u, q)$ and last agent action $a = (\mathtt{t}^1, \cdots, \mathtt{t}^K) \in \mathcal{A}(s)$ as input. Considering that the query $q$ and all tags are a sequence of words respectively, we first use a bidirectional LSTM to encode the query and tags. Then, the order information between tags in action $a$ is also important, so another bidirectional LSTM is employed to extract the order information between tags. The user $u$, query $q$ and action $a$ are encoded as $h^{s,u}$, $h^{s,q}$ and $h^{s,a}$ via a shared layer of our simulator.

**Task-specific layer.** The task-specific layer of each sub-task is a two fully connected layers with LeakyReLU as activation function. It takes $h^{s,s} = [h^{s,u}, h^{s,q}, h^{s,a}]$ as input and outputs the result of corresponding sub-task. Considering that all the three sub-tasks are classification problems, we use cross entropy as the loss function for all the three sub-tasks. Now, we can write the total loss function of our simulator as:

$$\mathcal{L}^s(\theta^s) = \mathcal{L}^s_{click}(\theta^s) + \mathcal{L}^s_{trmt}(\theta^s) + \mathcal{L}^s_{trt}(\theta^s) \tag{1}$$

where $\mathcal{L}^s_{click}(\theta^s)$, $\mathcal{L}^s_{trmt}(\theta^s)$ and $\mathcal{L}^s_{trt}(\theta^s)$ are the losses of clicking, terminating and transition sub-tasks respectively.

**Transition Probability Approximation**

In order to use the collected data efficiently, we approximate the transition probability matrix of states. Inspired by Serban et al. [2018], rather than estimating the full transition probability $p(s'|s, a)$ directly, we approximate it with:

$$\begin{aligned} p(s'|s, a) &= p(u, q'|u, q, a) = p^u(q'|q, a) \\ &\approx \sum_{z' \in \mathcal{Z}} p^u_{abs}(z'|q, a) p^u_{abs}(q'|z') \end{aligned} \tag{2}$$

where $u$ is the user that interacts with the agent in current session, and $\mathcal{Z}$ is a discrete abstract query space which meets the condition of $|\mathcal{Z}| \ll |\mathcal{Q}|$. In other word, we estimate a user-specific transition probability matrix for each user $u$. In this way, the size of original transition probability matrix $|\mathcal{A}||\mathcal{S}|^2$, where $|\mathcal{S}| = |\mathcal{U}||\mathcal{Q}|$, will be reduced to $|\mathcal{A}||\mathcal{U}||\mathcal{Q}|^2$. Further more, by introducing the abstract query space $\mathcal{Z}$, it will be further reduced to $|\mathcal{U}|(|\mathcal{A}||\mathcal{Q}||\mathcal{Z}| + |\mathcal{Q}||\mathcal{Z}|)$. After the above two steps, the sample complexity for accurately estimating the transition probability will be reduced greatly.

The transition is illustrated in Figure 3. Given a user $u$, a state $s_t = (u, q_t)$ is sampled with the probability $p^u_{abs}(q_t|z_t)$ conditioned on an abstract query $z_t \in \mathcal{Z}$. Then the agent takes an action $a_t \in \mathcal{A}(s_t)$ according to it's policy $\pi$ and receives the immediate reward $r_t$ from the environment. Conditioned on the query $q_t$ and action $a_t$, the next abstract query $z_{t+1}$ is sampled with the probability $p^u_{abs}(z_{t+1}|q_t, a_t)$.

**Algorithm 1:** Building real experience buffer with abstract query.

```
input  : $f_{q\to z}(q)$, $\mathcal{D} = \{(u, q, a, r, q', trmt)_i\}|_{i=1}^N$.
output: $\mathcal{D}^r$, $\mathcal{D}^u$, $\mathcal{D}^u_z$, $\forall z \in \mathcal{Z}, \forall u \in \mathcal{U}$.
1 initialize $\mathcal{D}^r$, $\mathcal{D}^u$, $\mathcal{D}^u_z, \forall z \in \mathcal{Z}, \forall u \in \mathcal{U}$ as empty set;
2 for $i \leftarrow 1$ to $N$ do
3     get $i$-th transition $(u, q, a, r, q', trmt)$ from $\mathcal{D}^r$ and
        $s = (u, q), s' = (u, q')$;
4     $z \leftarrow f_{q\to z}(q)$;
5     $z' \leftarrow f_{q\to z}(q')$;
6     append $(s, z, a, r, s', z', trmt)$ to $\mathcal{D}^r$, $\mathcal{D}^u$, $\mathcal{D}^u_z$
        respectively;
7 end
```

The transition distribution of $z$ is approximated via transition sub-task of our simulator, i.e., $p^u_{abs}(z_{t+1}|q_t, a_t) \approx \hat{p}_{abs}(z_{t+1}|u, q_t, a_t; \theta^s) = \hat{p}_{abs}(z_{t+1}|s_t, a_t; \theta^s)$. In terms of the transition from $z_t$ to $q_t$ with the probability $p^u_{abs}(q_t|z_t)$, we just sample a $q$ as $q_t$ uniformly from the pre-built buffer $\mathcal{D}^u_{z_t}$ (see Algorithm 1), where $f_{q\to z}$ is a known surjective function mapping from $\mathcal{Q}$ to $\mathcal{Z}$. And this will lead to:

$$p_{abs}(q|z) = 0 \;\; \forall q \in \mathcal{Q}, \; z \in \mathcal{Z} \text{ if } f_{q\to z}(q) \neq z. \tag{3}$$

## 4 RL-based Agent

In our scenario, an action $a_t = (\mathtt{t}^1_t, \cdots, \mathtt{t}^K_t) \in \mathcal{A}(s_t)$ is a sequence of $K$ tags (or sub-actions), and the number of candidate tags $\mathcal{T}_t$ is varying at different turns. This makes $\mathcal{A}(s_t)$ a discrete dynamic combinatorial action space. Thus, the traditional architecture of DQN that takes the state as input and outputs Q-values for each action at the output layer, is not suitable for this problem. We consider another architecture of Q-learning, named DRRN [He *et al.*, 2016], which takes the pair of state $s$ and action $a$ as input and outputs the Q-value for this pair. The architecture of our agent is similar to the shared layer of our environment simulator. The user $u$, query $q$ and action $a$ are encoded as $h^{a,u}$, $h^{a,q}$ and $h^{a,a}$ respectively. Then $h^{a,u}$, $h^{a,q}$ and $h^{a,a}$ are concatenated as $h^a$. $h^a$ is then fed into a two fully connected layers to calculate the Q-value for the pair of the state and the action $a$.

Given a set of transitions $\mathcal{D} = \{(s, a, r, s', trmt)_i\}|_{i=1}^N$ ($trmt$ indicates whether the state $s$ is a terminal state or not), the objective function is as follows:

$$\mathcal{L}^a(\theta^a) = \mathbb{E}_{s,a,r,s',trmt\sim\mathcal{D}} \left[ \left( y - Q(s, a; \theta^a) \right)^2 \right] \tag{4}$$

where:

$$y = \begin{cases} r + \gamma \max\limits_{a' \in \mathcal{A}(s')} Q(s', a'; \theta^{a^-}), & \text{if } trmt \text{ is } False. \\ r, & \text{otherwise} \end{cases}$$

where $Q(s, a; \theta^{a^-})$ is the target network with parameter $\theta^{a^-}$ from some previous iteration.

The rollout simulation between the simulator and agent is described in Algorithm 2. Given a state $s$, the $\epsilon$-greedy agent takes an action $a = \max_{a\in\mathcal{A}(s)} Q(s, a; \theta^a)$ with probability

**Algorithm 2:** Rollout Simulation

```
input  : Agent($s;\theta^a$), Simulator($s,a;\theta^s$), $\mathcal{U}$, $N^s$, $T$,
         $\mathcal{D}^s$, $\mathcal{D}^u$, $\mathcal{D}^u_z, \forall z \in \mathcal{Z}, \forall u \in \mathcal{U}$
output: Updated simulated experience buffer $\mathcal{D}^s$
1  for $i \leftarrow 1$ to $N^s$ do
2      sample a user $u$ from $\mathcal{U}$;
3      sample non-terminal transition $(s, a, z, r, s', z', trmt)$
        from $\mathcal{D}^u$;
4      $s_1 \leftarrow s$;
5      for $t \leftarrow 1$ to $T-1$ do
6          $a_t \leftarrow$ Agent($s_t;\theta^a$); // $\epsilon$-greedy
7          $c_t, trmt_t, p_{abs}(z_{t+1}|s_t, a_t) \leftarrow$
            Simulator($s_t, a_t; \theta^s$);
8          sample $z_{t+1}$ with probability $p_{abs}(z_{t+1}|s_t, a_t)$;
9          $r_t \leftarrow$ getReward($c_t$); // rule-based
10         sample $s$ uniformly as $s_{t+1}$ from $\mathcal{D}^u_{z_{t+1}}$;
11         append $(s_t, a_t, r_t, s_{t+1}, trmt_t)$ to $\mathcal{D}^s$;
12         if $trmt_t$ is ture then break;
13     end
14 end
```

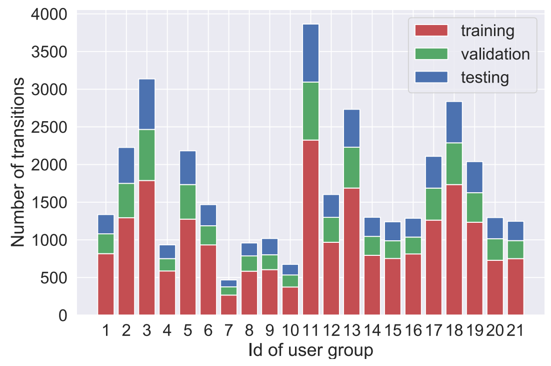

Figure 4: The distribution of transitions over different user groups.

$1-\epsilon$ and takes an action $a \in \mathcal{A}(s)$ randomly with probability $\epsilon$. The training procedure of our agent and simulator is described in Algorithm 3. The simulator is pre-trained to minimize the total loss via supervised learning firstly (line 3) and then minimize the losses of three sub-tasks respectively (line 4). At each step, the agent is first trained with real experience replay (line 8). Then, the agent interacts with the simulator (line 9) and simulated experience replay is performed on the simulated experiences buffer (line 11).

## 5 Experiments

### 5.1 Dataset and Abstract Query

Our dataset is derived from the log of Taobao APP [2], which is processed into transition tuple, i.e., $(s, a, r, s', trmt)$. The user feature $u \in \mathbb{R}^9$ is the concatenation of three one-hot vectors, namely, age (3 categories), gender (3 categories including null value) and purchase power (3 categories). And the data distribution is shown in Figure 4. For a transition, if a user clicked on one of the recommended tags, the reward $r = 1$, otherwise $r = 0$. We have $17,979$ transitions with

[2]The largest E-commerce platform in China.

**Algorithm 3:** Training Algorithm

```
input  : $\mathcal{D}$, $f_{s\to z}(s)$, $T$, $N^s$, $C$, $m$
output: Agent($s;\theta^a$)
1  build $\mathcal{D}^r$, $\mathcal{D}^u$, $\mathcal{D}^u_z, \forall z \in \mathcal{Z}, \forall u \in \mathcal{U}$ with Algorithm 1;
2  initialize Agent($s;\theta^a$), Agent($s;\theta^{a^-}$),
    Simulator($s,a;\theta^s$) randomly;
3  pre-train the Simulator($s,a;\theta^s$) with $\mathcal{D}^r$ via supervised
    learning to minimize Eq. 1;
4  pre-train the Simulator($s,a;\theta^s$) with $\mathcal{D}^r$ via supervised
      learning to minimize $\mathcal{L}^s_{click}(\theta^s)$, $\mathcal{L}^s_{trmt}(\theta^s)$ and
      $\mathcal{L}^s_{trt}(\theta^s)$ respectively; // parameters in
      shared layers keep fixed
5  initialize the simulated experience buffer $\mathcal{D}^s$ (fixed length)
    via Algorithm 2;
6  for $i \leftarrow 1$ to $\infty$ do
7      get $m$ minibatches from $\mathcal{D}^r$;
8      update $\theta^a$ with $m$ minibatches to minimize Eq. 4;
9      rollout simulation and update $\mathcal{D}^s$ with Algorithm 2;
10     sample $m$ minibatches from $\mathcal{D}^s$;
11     update $\theta^a$ with $m$ minibatches to minimize Eq. 4;
12     reset $\theta^{a^-} = \theta^a$ every $C$ steps;
13     if $\theta^a$ are converged then break;
14 end
```

| | | Training | Testing | Validation |
|---|---|---|---|---|
| Click | 1 | 10,788 | 3,595 | 3,596 |
| | 0 | 10,788 | 3,596 | 3,596 |
| Terminate | 1 | 5,394 | 1,798 | 1,798 |
| | 0 | 16,182 | 5,393 | 5,394 |
| Ave. # of tags | | 5.93 | 5.97 | 5.98 |
| Ave. length of tags | | 2.56 | 2.56 | 2.56 |
| Ave. length of query | | 5.55 | 5.20 | 5.22 |

Table 1: Description of our dataset, of which 60% for training, 20% for testing, 20% for validation.

$r = 1$. However, in terms of transitions with $r = 0$, we cannot say that the action $a$ derived from the log is bad because it is very likely that no click was made by the user in real world even the action is optimal. In order to reduce the noisy, only $17,980$ transitions with $r = 0$ sampled randomly are used.

As for abstract query $\mathcal{Z}$, we first represent $q$ as the average vector of words in $q$, then K-means is performed on these average vectors to assigns a cluster id for each $q$ (20 clusters). Considering the null value of query $q$, we have $|\mathcal{Z}| = 20 + 1 = 21$. Thus, the mapping function $f_{q\to z}(q)$ is getting the cluster id of query $q$.

### 5.2 Experiment Setup

**Implementation Details.** The simulator is pre-trained before interacting with our agent and fixed unchanged while interacting with the agent. Each word is represented by word embedding (200-dim) trained on the historical queries from other scenarios of Taobao APP (∼12.7 million queries, 332,922 words) via `word2vec`. $\epsilon = 0.2, \gamma = 0.9, m = 5, N^s = 5, T = 20$, learning rate is set to $10^{-5}$ and $10^{-3}$ for the training of environment simulator and agent respectively. At each turn, the agent recommends 3 tags to the user (i.e., $K = 3$). The target network of the agent is updated every 200 steps. The hidden size (both bidirectional LSTM and fully connected layers) of simulator and agent is 5 and 10. The length of simulated experiences buffer $\mathcal{D}^s$ is 1000.

### Evaluation Metrics

We evaluate our agent for both offline and simulated online setting and compare it with different training modes. For offline evaluation, we treat the tag clicked by real user as the positive one. We borrow metrics from information retrieval, including Mean Average Precision at $n$ (**MAP**@$n$), Normalized Discounted Cumulative Gain at $n$ (**NDCG**@$n$) and Recall at $n$ (**Recall**@$n$)[3]. Assuming that the number of the clicked tag is 1, we have MAP@1 = NDCG@1 = Recall@1 and only Recall is reported at position 1. It's worthy noting that only transitions with a clicked tag is used for offline evaluation.

For simulated online evaluation, we let agents trained to interact with our simulator. The metrics used include average cumulative reward and average number of turns per session. In the scenario of online shopping, it is better that users spend more time on the platform, so a larger average turn is preferred.

### Baselines

We compare our model with some baselines:

- **Random:** Given a state $s$, the random agent always takes an action $a \in \mathcal{A}(s)$ randomly.
- **DQN:** The agent is trained with real experiences replay without interacting with the simulator (Algorithm 3 without lines 9-11).
- **DQN-Sim:** The agent is trained via interacting with the simulator only (Algorithm 3 without lines 7-5).
- **DQN-Both:** The agent is trained with both real experiences replay and the simulator (Algorithm 3).

At each training step, DQN-Both is updated with $2m$ batches (both real and simulated experiences), while both DQN and DQN-Sim are updated with $m$ batches, i.e., real and simulated experiences respectively.

## 5.3 Experimental Results

In this section, we first compare our DQN agent with Random agent to evaluate the performance of our RL-based agents. Then, we compare the DQN agent with both DQN-Sim agent and DQN-Both agent to show that the training of the agent is converged faster and its performance is better with the assistance of the simulator.

Figure 5 shows the Recall of different agents at position 1, 2 and 3. We can see that all the RL-based agents outperform the random agent greatly on all metrics, which demonstrates the performance of our RL-based agents.

Table 2 shows the simulated online performance of different agents at training step = {4K, 10K}. As we can see, the two simulator-involved agents outperform the DQN agent in terms of both reward gained and number of turns lasted (the higher the better). At training step = 4K, the DQN-Both outperforms the two other agents. However, at training step = 10K, DQN-Sim agent reaches the best performance. The reason is that our environment simulator cannot mimic the behavior of real environment perfectly, which results in a dynamic environment for DQN-Both agent when replaying both real experiences and simulated experiences, making it difficult for DQN-Both agent to learn a optimal policy for both the real environment and simulated environment.

| Step | Agent | Reward | Turn |
|---|---|---|---|
| 4K | DQN | 4.147 ±0.582 | 9.358 ±1.072 |
| | DQN-Sim | 5.721 ±0.547 | 12.284 ±1.011 |
| | DQN-Both | 6.398 ±0.183 | 13.510 ±0.347 |
| 10K | DQN | 6.266 ±0.166 | 13.287 ±0.288 |
| | DQN-Sim | **6.432** ±0.088 | **13.640** ±0.152 |
| | DQN-Both | 6.395 ±0.117 | 13.544 ±0.238 |

Table 2: The performance of different agents at training step = {4K, 10K}) on 1K simulated sessions. The transition buffer $\mathcal{D}^u$, $\mathcal{D}^u_z$ for rollout simulation are built from testing set.

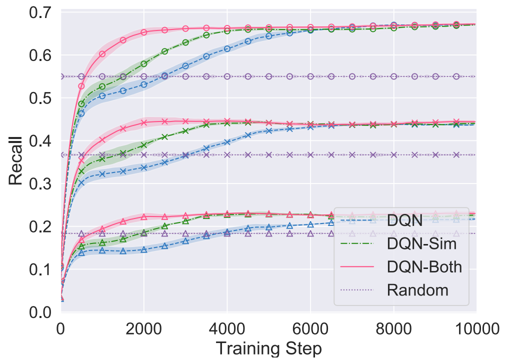

Figure 5: Learning curve of different agents on Recall. The △, × and ○ marked lines are the metrics at position 1, 2 and 3 respectively. The performance of different agents are evaluated on validation set (only transitions with click = 1) during the training of agents. The NDCG and MAP curve have similar trends with Recall curve due to the high relevance.

Table 3 shows the offline performance of different agents at training step = {2K, 4K, 10K}. At training step = 2K, 4K, both DQN-Sim agent and DQN-Both agent outperform DQN agent in a large margin. As the training continues, the gap between the DQN agent and the two simulator-involved agents is decreasing. The DQN-Sim agent, accessing the real data indirectly through our simulator, achieves comparable performance to DQN agent which indicates the effectiveness of our simulator.

More importantly, the DQN-Sim agent converges much faster than DQN agent during the training. We attribute this to the trade-off between exploration and exploitation. In other words, the real experiences buffer collected in advance is fixed during the training of the agents. Thus, the DQN agent is trained with a fixed buffer and there is no exploration of action space and state space. While the DQN-Both agent and DQN-Sim agent are trained via interacting with the simulator, during which they can have a better exploration.

[3]Considering that there exists only one positive tag for one query (i.e., one transition), we use Recall@$n$ rather than Precision at $n$ (Recall@$n$=$n$×P@$n$ in this work).

| Step | Agent | Recall@1 | Recall@2 | Recall@3 | NDCG@2 | NDCG@3 | MAP@2 | MAP@3 |
|---|---|---|---|---|---|---|---|---|
| 2K | DQN | .1445 ±.0213 | .3328 ±.0270 | .5305 ±.0282 | .2633 ±.0234 | .3622 ±.0239 | .2387 ±.0225 | .3046 ±.0228 |
| | DQN-Sim | .1877 ±.0176 | .3947 ±.0179 | .5880 ±.0238 | .3183 ±.0177 | .4149 ±.0198 | .2912 ±.0176 | .3556 ±.0188 |
| | DQN-Both | .2279 ±.0194 | .4472 ±.0280 | .6552 ±.0171 | .3663 ±.0244 | .4703 ±.0186 | .3375 ±.0232 | .4069 ±.0193 |
| 4K | DQN | .1864 ±.0204 | .3949 ±.0199 | .6113 ±.0251 | .3179 ±.0200 | .4261 ±.0226 | .2906 ±.0200 | .3628 ±.0218 |
| | DQN-Sim | .2233 ±.0076 | .4305 ±.0068 | .6471 ±.0067 | .3540 ±.0062 | .4623 ±.0059 | .3269 ±.0063 | .3991 ±.0060 |
| | DQN-Both | .2235 ±.0101 | .4414 ±.0131 | .6573 ±.0081 | .3610 ±.0111 | .4689 ±.0085 | .3325 ±.0106 | .4044 ±.0089 |
| 10K | DQN | .2233 ±.0086 | .4451 ±.0095 | .6773 ±.0056 | .3632 ±.0089 | .4793 ±.0051 | .3342 ±.0088 | .4116 ±.0060 |
| | DQN-Sim | .2318 ±.0132 | .4498 ±.0163 | .6767 ±.0072 | .3693 ±.0145 | .4828 ±.0101 | .3408 ±.0140 | .4164 ±.0112 |
| | DQN-Both | **.2369** ±.0085 | **.4536** ±.0060 | **.6843** ±.0058 | **.3736** ±.0060 | **.4890** ±.0053 | **.3452** ±.0063 | **.4222** ±.0057 |

Table 3: The performance of different agents at training step = {2K, 4K, 10K} on testing set (only transitions with click = 1). The difference of Recall@1 between DQN and DQN-Sim/DQN-Both evaluated at the same training step is statistically significant (The one-tailed $p < 0.1$).

| # of rollouts | 1 | 20 | 50 | 100 |
|---|---|---|---|---|
| Percentage | 3.9% | 9.4% | 16.6% | 25.7% |

Table 4: Percentage of action space explored by the agent in terms of # of rollouts. # of rollouts is the repeated times the agent takes an action with $\epsilon$-greedy strategy for each turn-level transition. And the percentage is the average percentage of the number of different actions taken by the agent.

| Sub-task | Precision | Recall | F1-Score |
|---|---|---|---|
| Click | 0.6853 | 0.6741 | 0.6693 |
| Terminate | 0.7170 | 0.6130 | 0.6386 |
| Transition | 0.3377 | 0.2742 | 0.2648 |

Table 5: The performance of our simulator on three tasks.

We perform an experiment to reveal how the exploration can be enforced with the assistance of our simulator. We present the percentage of action space explored by the agent in terms of number of rollouts. When the number of rollouts equals to 1, it is the same as DQN with fixed real experiences. As shown in Table 4, we can see that only 3.9% of actions can be visited without experiences generated by the simulator. While the percentage of visited actions increases greatly when the simulator generated experiences are involved. The number increases with the number of rollouts.

**Performance of environment simulator.** The performance of our simulator on three sub-tasks are reported in Table 5. We use three metrics including recall, precision and F1 score.

## 6 Related Work

RL-based algorithms have been applied to different scenarios of recommendation. [Zhao *et al.*, 2018] applies actor-critic algorithm to solve items recommendation involving the 2-D order between items in one page. [Zheng *et al.*, 2018] integrates user behavior and profile information into state representation for online news recommendation with RL-based agent. [Choi *et al.*, 2018] introduces an RL-based framework for recommendation where states are represented as grid-world obtained from bi-clustering to reduce the state space and action space. [Chen *et al.*, 2018] employed stratified sampling and regret approximation to stabilize the learning of agent for recommendation.

To our knowledge, the agenda-based simulator [Schatzmann *et al.*, 2007] is the first simulator for RL-based tasks, and have been widely applied in dialogue system [Wei *et al.*, 2018; Li *et al.*, 2017; Peng *et al.*, 2017]. It is rule-based, where the handcrafted action space of environment simulator should be pre-defined and cannot be generalized to complex tasks (e.g., online shopping). From the perspectives of simulator and agent, the interacting process can both be treated as a sequential decision problem respectively. Thus, a simulator could be learned via inverse reinforcement learning based on experts' (users') demonstrations [Chandramohan *et al.*, 2011]. What's more, the sequence-to-sequence architecture could also be employed to model the user simulator [Asri *et al.*, 2016]. [Peng *et al.*, 2018] propose a multi-task user simulator to interact with the agent. Instead of transiting into next state directly, it selects one of the pre-defined user actions and an additional state tracker is needed to update the state.

How to train the agent with collected data efficiently has attracted a lot of research. [Lu *et al.*, 2018] introduce two approaches for data augmentation in task-completion tasks so that the agent could be trained more efficiently. Considering the model-based RL algorithms could learn a better policy from less data, state abstraction have been used to reduce the complexity of model-based RL methods [Serban *et al.*, 2018; Jiang *et al.*, 2015].

## 7 Conclusion and Future Work

In this work, we introduce a multi-task model-based environment simulator for online shopping platform to train the RL-based agent via interactions. Experimental results on real-world dataset demonstrate that the agent can converge faster with the assistance of the simulator. Further analysis shows that agent is able to explore larger action space with experiences generated by the simulator. In future, we will explore to improve the performance of the simulator to provide a better simulation of the environment.

## Acknowledgments

This work is partially funded by National Natural Science Foundation of China (No. 61751201), National Natural Science Foundation of China (No. 61702106) and Shanghai Science and Technology Commission (No. 17JC1420200, No. 17YF1427600 and No. 16JC1420401).

## References

[Asri *et al.*, 2016] Layla El Asri, Jing He, and Kaheer Suleman. A sequence-to-sequence model for user simulation in spoken dialogue systems. *CoRR*, abs/1607.00070, 2016.

[Chandramohan *et al.*, 2011] Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefevre, and Olivier Pietquin. User simulation in dialogue systems using inverse reinforcement learning. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[Chen *et al.*, 2018] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1187–1196. ACM, 2018.

[Choi *et al.*, 2018] Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. Reinforcement learning based recommender system using biclustering technique. *CoRR*, abs/1801.05532, 2018.

[Clark, 2015] Jack Clark. Google turning its lucrative web search over to ai machines. *Bloomberg Technology*, 26, 2015.

[He *et al.*, 2016] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1621–1630, 2016.

[Jiang *et al.*, 2015] Nan Jiang, Alex Kulesza, and Satinder Singh. Abstraction selection in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 179–188, 2015.

[Li *et al.*, 2011] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297–306. ACM, 2011.

[Li *et al.*, 2017] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. End-to-end task-completion neural dialogue systems. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 733–743, 2017.

[Liu *et al.*, 2017] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. Cascade ranking for operational e-commerce search. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1557–1565. ACM, 2017.

[Lu *et al.*, 2018] Keting Lu, Shiqi Zhang, and Xiaoping Chen. Learning to dialogue via complex hindsight experience replay. *arXiv preprint arXiv:1808.06497*, 2018.

[Peng *et al.*, 2017] Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2231–2240, 2017.

[Peng *et al.*, 2018] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2182–2192, 2018.

[Schatzmann *et al.*, 2007] Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152. Association for Computational Linguistics, 2007.

[Serban *et al.*, 2018] Iulian Vlad Serban, Chinnadhurai Sankar, Michael Pieper, Joelle Pineau, and Yoshua Bengio. The bottleneck simulator: A model-based deep reinforcement learning approach. *arXiv preprint arXiv:1807.04723*, 2018.

[Taghipour and Kardan, 2008] Nima Taghipour and Ahmad Kardan. A hybrid web recommender system based on q-learning. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1164–1168, New York, NY, USA, 2008. ACM.

[Wei *et al.*, 2018] Zhongyu Wei, Qianlong Liu, Baolin Peng, Huaixiao Tou, Ting Chen, Xuanjing Huang, Kam-Fai Wong, and Xiangying Dai. Task-oriented dialogue system for automatic diagnosis. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 201–207, 2018.

[Yin *et al.*, 2016] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. Ranking relevance in yahoo search. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 323–332. ACM, 2016.

[Zhao *et al.*, 2018] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 95–103, New York, NY, USA, 2018. ACM.

[Zheng *et al.*, 2018] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 167–176, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee.