Rethinking Reinforcement Learning for Recommendation: A Prompt Perspective

Xin Xin
Shandong University
China
xinxin@sdu.edu.cn

Tiago Pimentel
University of Cambridge
United Kingdom
tp472@cam.ac.uk

Alexandros Karatzoglou
Google Research
United Kingdom
alexkz@google.com

Pengjie Ren
Shandong University
China
jay.ren@outlook.com

Konstantina Christakopoulou
Google
United States
konchris@google.com

Zhaochun Ren*
Shandong University
China
zhaochun.ren@sdu.edu.cn

ABSTRACT

Modern recommender systems aim to improve user experience. As reinforcement learning (RL) naturally fits this objective—maximizing an user's reward per session—it has become an emerging topic in recommender systems. Developing RL-based recommendation methods, however, is not trivial due to the *offline training challenge*. Specifically, the keystone of traditional RL is to train an agent with large amounts of online exploration making lots of 'errors' in the process. In the recommendation setting, though, we cannot afford the price of making 'errors' online. As a result, the agent needs to be trained through offline historical implicit feedback, collected under different recommendation policies; traditional RL algorithms may lead to sub-optimal policies under these offline training settings.

Here we propose a new learning paradigm—namely Prompt-Based Reinforcement Learning (PRL)—for the offline training of RL-based recommendation agents. While traditional RL algorithms attempt to map state-action input pairs to their expected rewards (e.g., Q-values), PRL directly infers actions (i.e., recommended items) from state-reward inputs. In short, the agents are trained to predict a recommended item given the prior interactions and an observed reward value—with simple supervised learning. At deployment time, this historical (training) data acts as a knowledge base, while the state-reward pairs are used as a prompt. The agents are thus used to answer the question: *Which item should be recommended given the prior interactions & the prompted reward value*? We implement PRL with four notable recommendation models and conduct experiments on two real-world e-commerce datasets. Experimental results demonstrate the superior performance of our proposed methods.

*Corresponding author.

SIGIR '22, July 11–15, 2022, Madrid, Spain

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3531714

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Retrieval models and ranking**; **Novelty in information retrieval**.

KEYWORDS

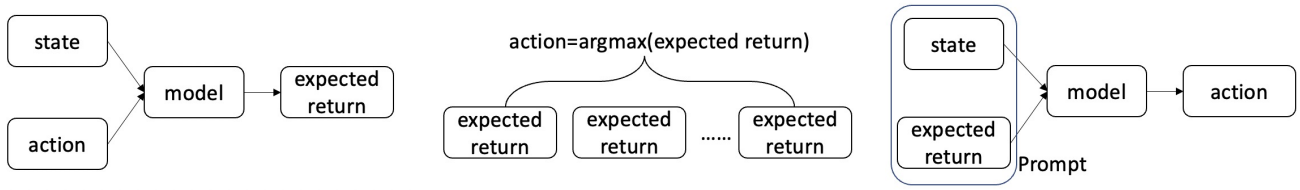
Next Item Recommendation; Reinforcement Learning; Recommender Systems; Session-based Recommendation

ACM Reference Format:
Xin Xin, Tiago Pimentel, Alexandros Karatzoglou, Pengjie Ren, Konstantina Christakopoulou, and Zhaochun Ren. 2022. Rethinking Reinforcement Learning for Recommendation: A Prompt Perspective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3477495.3531714

1 INTRODUCTION

Next item recommender systems are one of the core components of several modern online web services, including music or video streaming services [45], and e-commerce sites [17]. They are holistically ingrained into these applications, helping users navigate and find new content. As a general rule, these systems are modelled as sequence prediction tasks—they thus answer the question: *What is the next item the user would be interested to interact with given the past interactions*—and are typically implemented on top of recurrent neural networks or other generative sequential models. Conventional next item recommendation models are usually trained through an auto-regressive fashion, in which the model is trained to recover the historical interaction sequence [14, 22, 45]. Similar learning objectives are also used in language modeling in the field of natural language processing (NLP). Simply predicting the next item a user will interact with, however, may be a poor objective; one might prefer to instead maximise long-term engagement, for instance, or the diversity of the consumed items.

Reinforcement learning (RL) has been successfully employed in planning and controlling [29, 36]. An RL agent is trained to take actions which, given the observed state of the environment, maximize a pre-defined reward. Existing value-based RL algorithms usually involve policy evaluation and policy improvement, as shown in Figures 1a and 1b, respectively. Policy evaluation aims to learn a model which maps the state-action input pairs to the expected cumulative rewards (i.e., Q-values). Policy improvement selects the action with



(a) Policy evaluation. (b) Policy improvement. (c) PRL.

Figure 1: Traditional RL algorithms involve policy evaluation (a) to predict the expected return (i.e., cumulative reward) and then use policy improvement (b) to select actions with the highest return prediction. While PRL (c) aims to directly infer actions given the prompt of current state and expected return.

the maximum Q-value prediction. The long-term nature of planning in RL fits naturally with desirable properties in recommender systems (RS). The flexible reward setting in RL enables for flexible customization of recommendation objectives. As a result, the use of RL in recommendation has become an emerging topic [1, 9, 46].

However, developing RL-based recommendation methods is non-trivial. The learning paradigm of RL trains the agent by interacting with the environment and then observing the reward. The agents are reinforced towards taking actions with higher cumulative returns. This process needs a large amount of interactions taken by the agent itself. Although some existing RL methods are claimed to be "off-policy", they still need the agent to step over plenty of online interactions to refresh the replay buffer. Such a learning paradigm is feasible in fields like gaming [36], since conducting error-prone explorations does not come at a cost. In the field of RS, however, we cannot afford the price of making errors, since bad recommendation results will definitely affect user experience. As a result, we want to train the RL recommendation agent through fixed historical data without the agent being able to probe the environment. However, this historical data is not generated by the target agent itself, but from different or even unknown behavior policies. The expected quality of an estimated policy can be easily affected by this discrepancy in distributions. This problem is know as the *offline training challenge*.

Previous work attempted to address the offline training challenge through inverse propensity scores [7], model-based user simulation [8], or combining RL with supervised learning [43]. However, such methods still suffer from factors of unbounded high variances [30], biased user simulation [18] and Q-value estimation [10].

We propose *Prompt-Based Reinforcement Learning* (PRL), a new paradigm for effective offline training of RL-based recommendation agents. The concept of prompting is rooted in NLP [28], whereby large language models have been shown to learn new tasks with a single demonstration of an example. While PRL is not identical to this few-shot NLP concept, it is a similar concept in that we achieve the control of recommendation results by feeding the model with different prompt templates. PRL uses the offline historical data as a knowledge base while the state-reward pairs act as the prompt. The agents are trained to answer the question of *which item should be recommended if the prompted reward value is expected to be achieved under the given state*. In the training stage a generative sequential model is used to encode the users previous interactions into a hidden state, which can be regarded as the state of the environment in the RL setting. The current recommended item can be seen as the action. From the offline data we can compute the exact cumulative reward at each step of an interaction session. As a result, the historical data can be organized in the template of {state, cumulative reward}−>{observed action}. We then use a simple yet effective supervised self-attentive block [41] to learn and store such signals. During the inference stage, given the current state we feed the model an expected cumulative reward we want to obtain (e.g., twice the average cumulative reward of the current step), the model can directly infer actions through querying the historical knowledge base, as shown in Figure 1c. PRL enables the recommendation agent to adjust its actions conditioning on the prompt reward. For example, agent exploration can be effectively achieved by introducing random noise on the prompt reward during the inference stage. We verify the effectiveness of our approach by implementing PRL with four renowned recommendation models.

To summarize, this work makes the following contributions:

- We propose prompt-based reinforcement learning for the offline training of RL-based next item recommendation. We propose to use the state-reward pairs as the prompt to infer actions through querying the knowledge base of historical implicit feedback data.
- We propose to use a supervised self-attentive block to learn and store the signals between the input of state-reward pairs and the output of actions.
- We implement PRL with four renowned next item recommendation models as the state encoders, and conduct experiments on two real world e-commerce datasets. Experimental results demonstrate a generalized improvement of recommendation performance.

2 CHALLENGE INVESTIGATION

We first formulate the task of next item recommendation. Then we introduce reinforcement learning and analyse the offline training challenge. After that, the concept of prompting is described.

2.1 Next Item Recommendation

Let $\mathcal{I}$ denote the entire set of items in a specific system, then a user-item interaction sequence can be represented as $x_{1:t} = \{x_1, x_2, ..., x_{t-1}, x_t\}$, where $x_i \in \mathcal{I} (0 < i \leq t)$ is the interacted

item at timestamp i. Next item recommendation aims at recommending items that a user might be interested in at timestep $t+1$ given the sequence of past items $x_{1:t}$.

2.2 Reinforcement Learning and the Challenge

An RL agent is trained to take actions in an environment to get the maximum cumulative reward. This task is usually formulated as a Markov Decision Process (MDP) [11, 35, 36]. More precisely, for next item recommendation, users can be seen as the environment, while the MDP can be defined as tuples of $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \rho_0, \lambda)$ where

- $\mathcal{S}$: the space of all possible user states, which can be modeled through previous item interactions. Concisely, we can use a sequential model G to map the previous interaction sequence before timestamp t into a hidden state as $\mathbf{s}_t = G(x_{1:t}) \in \mathcal{S}$ $(t > 0)$. We will discuss prominent models for implementing $G(\cdot)$ in section 5.
- $\mathcal{A}$: the discrete action space which contains candidate items. An action a in the MDP represents the selection of a recommended item. In the offline training data, we can get the action at each timestamp t as $a_t = x_{t+1}$.
- $\mathbf{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the state transition probability, describing how the environment state changes when an action is performed in the environment.
- R: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function where r_t denotes the immediate reward at the t-th interaction step[1]. This is the key component of RL, which enables the agent to be trained in a customizable reward-driven fashion, such as promoting purchases [43], increasing diversity [38] or dwell time [7].
- ρ_0 describes the initial state distribution as $\mathbf{s}_0 \sim \rho_0$.
- λ is the discount factor for future rewards.

The goal of RL is to find a target policy $\pi_\theta(a|\mathbf{s})$ which maps the user's state $\mathbf{s} \in \mathcal{S}$ into a probability distribution over actions $a \in \mathcal{A}$, so that if the agent samples actions according to π_θ the system can obtain the maximum expected cumulative reward:

$$\max_{\pi_\theta} \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)], \text{ where } R(\tau) = \sum_{t=0}^{|\tau|} \lambda^t r_t(\mathbf{s}_t, a_t), \tag{1}$$

where θ denotes policy parameters and $\tau = (\mathbf{s}_0, a_0, \mathbf{s}_1, ...)$ is the sampled trajectory of the target policy with $\mathbf{s}_0 \sim \rho_0$, $a_t \sim \pi_\theta(\cdot|\mathbf{s}_t)$, $s_{t+1} \sim \mathbf{P}(\cdot|\mathbf{s}_t, a_t)$.

2.2.1 On-Policy Optimization. On-policy optimization, e.g. policy-gradient (PG) [42], is one of the most adopted methodologies to solve Eq.(1). PG aims at directly deriving the gradients of the expected cumulative rewards with respect to policy parameters θ as:

$$\nabla = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau) \nabla_\theta \log \pi_\theta(\tau)]. \tag{2}$$

Estimating this expectation requires a large amount of explorations taken by the agent itself, as shown in Figure 2a. However, for the offline training of RS from historical data, all we can estimate is:

$$\nabla' = \mathbb{E}_{\tau \sim \beta}[R(\tau) \nabla_\theta \log \pi_\theta(\tau)], \tag{3}$$

where β denotes the behavior data distribution of the historical data. Obviously, there is a difference between the distributions π_θ and β. [7] proposed to introduce an inverse propensity score (IPS) to correct the discrepancy at each timestamp as:

$$\nabla(t) = \frac{\pi_\theta(\tau(t))}{\beta(\tau(t))} \nabla'(t) \approx \frac{\pi_\theta(a_t|\mathbf{s}_t)}{\beta(a_t|\mathbf{s}_t)} \nabla'(t) \tag{4}$$

However, estimating a behavior policy β could be difficult [44]; and the computed IPS can have unbounded high variance [1].

2.2.2 Off-Policy Optimization. Off-policy optimization methods use a replay buffer to store past experiences and improve data efficiency. Deep Q-learning [36] (DQN) is one of the most typical off-policy methods. DQN utilizes policy evaluation (see Figure 1a) to calculate Q-values[2] and then policy improvement (see Figure 1b) to select actions with the highest Q-values. The model in policy evaluation is updated as follows:

$$\begin{aligned} \theta &\leftarrow \theta - \alpha \frac{\partial \mathbb{E}_{s,a \sim \pi'_\theta}(Q_\theta(\mathbf{s}, a) - Q_T(\mathbf{s}, a))^2}{\partial \theta}, \texttt{where} \\ Q_T(\mathbf{s}, a) &= r + \lambda \max_{a'} \mathbb{E}_{\mathbf{s}' \sim \mathbf{P}(\cdot|\mathbf{s},a)} Q_\theta(\mathbf{s}', a'). \end{aligned} \tag{5}$$

α is the learning rate, $Q_\theta(\mathbf{s}, a)$ denotes the Q-value calculated from the model while $Q_T(\mathbf{s}, a)$ is the target Q-value computed from time-difference (TD) learning [4], π'_θ denotes the policy used to build and refresh the replay buffer. This off-policy method requires π'_θ to be defined as the previous version of π_θ [10, 36]. In other words, the experiences stored in the replay buffer are still generated by the agent itself and new explorations are needed to refresh the replay buffer, as shown in Figure 2b.

However, when performing offline learning, historical training data comes from different and typically unknown agents. To avoid affecting user experience we anticipate that new explorations with user involvement are not needed until the agent is well trained, as shown in Figure 2c. In such setting, what we can update is:

$$\begin{aligned} \theta &\leftarrow \theta - \alpha \frac{\partial \mathbb{E}_{s,a \sim \beta}(Q_\theta(\mathbf{s}, a) - Q'_T(\mathbf{s}, a))^2}{\partial \theta}, \texttt{where} \\ Q'_T(\mathbf{s}, a) &= r + \lambda \max_{a'} \mathbb{E}_{\mathbf{s}' \sim \mathbf{P}_\beta(\cdot|\mathbf{s},a)} Q_\theta(\mathbf{s}', a'). \end{aligned} \tag{6}$$

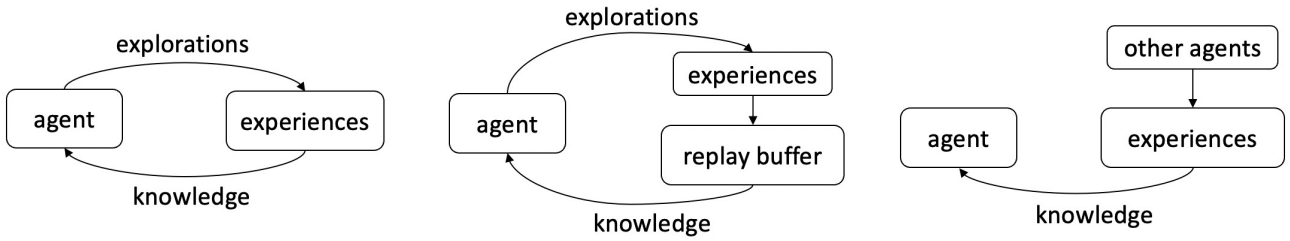
where $s' \sim \mathbf{P}_\beta(\cdot|\mathbf{s}, a)$ denotes that the next state s' is sampled from the offline data, rather than operating actions online and then observe the next state. Given that the state and action distribution in β can be different from the target policy, the parameter update in Eq.(6) can easily be biased. [10, 43] have shown that off-policy methods suffer from weak performance in the offline training setting.

2.2.3 Model-based RL. An alternative approach to train an RL recommendation agent is to use model-based reinforcement learning [8, 16]. Model-based RL is based on a model of the environment. The agents can then be trained through on-policy or off-policy methods with the data generated from interactions with the simulated environment rather than the real environment. As a result, users would not be involved directly during the training stage. However, model-based RL suffers from the following issues:

- The reward estimation of the simulator can be affected by various biases in the training data [5, 18].

[1] While in general a reward is not necessarily a deterministic function of a state–action pair, we will assume so to simplify our exposition.

[2] The Q-value for a state-action pair (i.e., $Q(\mathbf{s}, a)$) is defined as the expected cumulative reward gain if the action a is operated under the state $\mathbf{s}$.



(a) On-policy optimization. (b) Off-policy optimization. (c) Offline training for RS.

Figure 2: On-policy optimization methods (a) need to learn from large amount of experiences taken by the agent itself. Off-policy methods (b) improve the data efficiency through introducing a replay buffer to store the past experiences. However, the stored experiences still come from the agent itself and new interactions are needed to refresh the buffer. For the offline training of RS (c), the agent is expected to be trained from experiences of other agents without new explorations and users' involvement.

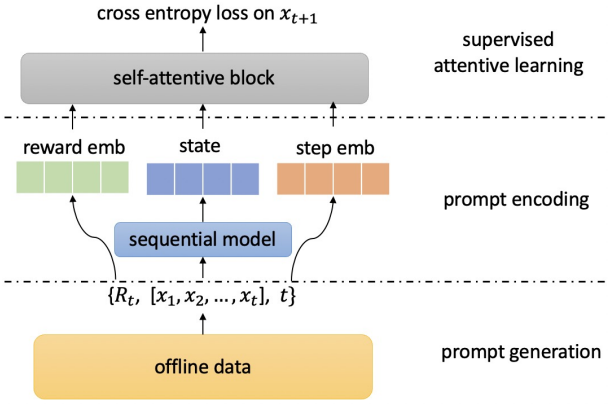


Figure 3: The training framework of PRL. "emb" is short for embedding. Prompt generation transforms the offline historical training data into tuples: {cumulative reward R_t, previous interactions $x_{1:t}$, interaction step t}. Then prompt encoding encodes the tuples into latent representations. Finally, a self-attentive block is used to learn the mapping function between the prompt and the action. The model parameters can be updated through a simple supervised cross-entropy loss function.

- The transition between (user) states is dynamic and difficult to model [25].
- The generalization ability of the constructed simulator is not well justified [2].

2.3 Prompt and Knowledge Base

A new learning paradigm namely prompt learning has become an emergent topic in the field of NLP. Different from the widely adopted "pre-training and fine-tuning"—which first pre-trains the model with tasks like language modeling and then fine-tunes learned parameters for downstream tasks—prompt learning aims to use the pre-trained model (which was pre-trained on a large training corpus) as a knowledge base and then formulates the downstream tasks as a prompt [28]. Since in the historical offline data we can compute the exact cumulative reward at each interaction step[3], the historical offline data can be formulated in the following way {state, cumulative reward}–>{observed action}, which can be intuitively interpreted as the signal for: *which action should be taken to obtain the cumulative reward given the user state*. During inference, we can expect the model to suggest actions which, if taken, should achieve the prompted reward in expectation, as shown in Figure 1c. Such a learning paradigm enables us to train the reward-driven RL recommendation agent in a much simpler supervised fashion. Note that, in this work, we exploit the concept of prompt to inspire our approach, but we don't investigate the complex prompt generation methods of NLP. The prompt used in this paper is the pair of state and cumulative reward. We leave more advanced prompt generation for recommendation as future work.

3 METHODOLOGY

In this section, we describe the detailed training and inference procedures for next item recommendation with PRL.

3.1 PRL Training

Training PRL consists of prompt generation, prompt encoding, and supervised attentive learning as shown in Figure 3.

3.1.1 Prompt Generation. Prompt generation aims to formulate the offline training data as a knowledge template which tells us which observed action x_{t+1} should be taken if we want to get the cumulative reward R_t with $x_{1:t}$ as the previous user-item interactions. In the offline training data, the cumulative reward R_t at each interaction step t of a session can be exactly computed as:

$$R_t = \sum_{t'=t}^{|\tau|} \lambda^{t'} r_{t'}, \tag{7}$$

where $|\tau|$ denotes the total steps of the interaction session. For each interaction session in the offline training data, we can inefficiently compute the cumulative reward R_t at every step as shown in Eq.(7) with a total time complexity of $O(|\tau|^2)$. A more efficient solution is to compute R_1 firstly and then R_{t+1} can be computed from R_t

[3]In this work, we do not consider the case of delayed rewards.

recursively by decreasing the reward $\lambda^t r_t$. With this recursive procedure, we can compute these rewards in $O(|\tau|)$ instead. Algorithm 1 shows the detailed procedure to reformulate the offline training sequences $\mathcal{D}_s$ as a prompt-based training set $\mathcal{D}_p$.

3.1.2 Prompt Encoding. Prompt encoding aims to use deep neural networks to map the generated prompt into latent representations. For the interaction sequence $x_{1:t}$, plenty of research has been proposed to capture the sequential signals, such as recurrent neural network (RNN)-based methods [14], convolutional neural network (CNN)-based methods [40, 45], and attention-based methods [22, 39]. We will give a more detailed description in section 5. The proposed PRL acts as a learning paradigm and all of these methods can be used as the sequential model shown in Figure 3.

Take the gated recurrent units (GRU) [14] as an example, we first embed each item x_i into a dense representation $\mathbf{x}_i \in \mathbb{R}^d$, where d denotes the embedding size. This can be done through a simple embedding table lookup operation. Then, the hidden state $\mathbf{s}_t$ for a given sequence of $x_{1:t}$ is defined as:

$$\begin{aligned} \mathbf{s}_t &= (1-\mathbf{z}_t)\mathbf{s}_{t-1} + \mathbf{z}_t\hat{\mathbf{s}}_\mathbf{t} \\ \mathbf{z}_t &= \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{s}_{t-1}) \\ \hat{\mathbf{s}}_\mathbf{t} &= tanh(\mathbf{W}_s\mathbf{x}_t + \mathbf{U}_s(\mathbf{g}_t \odot \mathbf{s}_{t-1})) \\ \mathbf{g}_t &= \sigma(\mathbf{W}_g\mathbf{x}_t + \mathbf{U}_g\mathbf{s}_{t-1}), \end{aligned} \tag{8}$$

where σ denotes the sigmoid function and $\odot$ is element-wise product. $\mathbf{W}_z, \mathbf{U}_z, \mathbf{W}_s, \mathbf{U}_s, \mathbf{W}_g, \mathbf{U}_g \in \mathbb{R}^{d\times d}$ are trainable parameters. In our experiments, we use four renowned sequential models to encode $x_{1:t}$. This allows us to verify the effectiveness and generalization ability of the proposed PRL. We don't elaborate on the details of all models, though, since this is not the key point of this work.

The representation for cumulative reward R_t is defined as:

$$\mathbf{e}_{R_t} = R_t \cdot \mathbf{e}_r, \tag{9}$$

where $\mathbf{e}_r \in \mathbb{R}^d$ is a trainable reward embedding. Another solution would be using different reward embeddings for discretized rewards. Besides, we also maintain a trainable embedding table $\mathbf{H}_T \in \mathbb{R}^{T\times d}$ to encode the step information. The final representation for the prompt $\{R_t, x_{1:t}, t\}$ is formulated as:

$$\mathbf{P}_t = [\mathbf{e}_{R_t}, \mathbf{s}_t, \mathbf{h}_t] \in \mathbb{R}^{3\times d}, \tag{10}$$

where $[\cdot]$ denotes the stack operation.

Algorithm 1 Prompt generation from offline training data

Input: user-item interaction sequence set $\mathcal{D}_s$, reward settings
Output: prompt-based training set $\mathcal{D}_p$

```
repeat
    Sample an interaction sequence x_{1:T} from D_s
    Compute R_1 according to Eq.(7)
    for t=1 : T − 1 do
        D_p.append({R_t, x_{1:t}, t}, x_{t+1})
        R_{t+1} = R_t − λ^t r_t
    end for
    D_s.remove(x_{1:T})
until D_s = ∅
return D_p
```

3.1.3 Supervised Attentive Learning. Given the encoded prompt representation $\mathbf{P}_t$, we need a model to learn to map the signal between $\mathbf{P}_t$ and observed action a_t (i.e., x_{t+1}). Self-attention [41] has been widely adopted in the field of NLP and has demonstrated impressive model capability. Recently, there are also works [6, 20, 31] attempting to introduce self-attention to RL. Inspired by this research, we propose to use a self-attentive block to learn the mapping signal. The dot-product based attention [41] is formulated as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}, \tag{11}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ denote the queries, keys and values, respectively. The attention computes a weighted addition of values according to the importance weights computed through the correlations between the query and the key [22, 41]. The scale factor $\sqrt{d}$ is used to normalize the computed values to avoid large inner products, especially when the dimension of the representations is large [22, 41].

Self-attention uses the same objects as queries, keys, and values. In the proposed PRL, we convert $\mathbf{P}_t$ to three representations, using linear projections, and then feed them to the attention layer. The residual connection [12] is introduced to incorporate the original $\mathbf{P}_t$ information. The final prompt representation $\tilde{\mathbf{P}}_t$ is defined as:

$$\tilde{\mathbf{P}}_t = \mathbf{P}_t + \text{Attention}(\mathbf{W}_q\mathbf{P}_t, \mathbf{W}_k\mathbf{P}_t, \mathbf{W}_v\mathbf{P}_t), \tag{12}$$

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d\times d}$ are trainable parameters. To avoid overfitting and enable a more stable learning without vanishing or exploding gradient issues, we include optional dropout layers and layer normalization [3].

Three attentive representations can then be extracted from $\tilde{\mathbf{P}}_t$:

$$\tilde{\mathbf{e}}_{R_t}, \tilde{\mathbf{s}}_t, \tilde{\mathbf{h}}_t = unstack(\tilde{\mathbf{P}}_t). \tag{13}$$

We feed the attentive state representation $\tilde{\mathbf{s}}_t$ to a fully connected layer to compute the classification logits on the candidate actions.

$$[y_1, y_2, ...y_n] = \delta(\mathbf{W}_i\tilde{\mathbf{s}}_t + \mathbf{b}), \tag{14}$$

where δ denotes the activation function and n is the number of candidate actions. $\mathbf{W}_i \in \mathbb{R}^{n\times d}$ can be seen as another trainable item embedding matrix and $\mathbf{b} \in \mathbb{R}^n$ is a bias vector.

Actor-Critic [24, 43] methods have achieved excellent results in recent research. The key idea is to use the predicted Q-values from the critic to re-weight the actor so that actions with higher cumulative reward would have more effect in the training stage. In PRL we re-weight the training samples with the immediate reward r_t for a more stable training. The weighted supervised cross-entropy loss is defined as:

$$L = -r_t \sum_{i=1}^{n} Y_i \log(p_i), \text{where } p_i = \frac{e^{y_i}}{\sum_{i'=1}^{n} e^{y_{i'}}}. \tag{15}$$

Y_i is an indicator function which is defined as $Y_i = 1$ if the user interacted with the i-th item in the next timestamp. Otherwise, $Y_i = 0$. Algorithm 2 shows a detailed training procedure of PRL.

3.2 PRL Inference

In training PRL, the cumulative reward can be computed from the offline data. At inference time, however, we need to provide the model with how much reward we want to obtain; the agent can then adjust its actions conditioning on the prompted reward. For

an interaction step t, a concise prompt reward can be set as the average cumulative reward of the offline training data at this step. To make the model more flexible, we extend the prompt reward for PRL inference as:

$$\tilde{R}_t = \mathcal{N}(\mu, \epsilon^2) \times \bar{R}_t, \tag{16}$$

where $\bar{R}_t$ denotes the average cumulative reward of step t in the training data. $\mathcal{N}(\mu, \epsilon^2)$ is a Gaussian distribution with μ as the mean and ϵ as the standard deviation. There are also various inference reward settings (e.g., according to the maximum cumulative reward in the training data). For offline inference and evaluation, we can prompt the model with such expected reward at each timestamp. Besides, for online inference, the proposed PRL can also support sequence-wise recommendation generation through promoting the model with the expected cumulative reward at the beginning (i.e., the first timestamp) and then decreasing the obtained reward according to the real user feedback. A more desired setting could be that the prompt inference reward can be automatically adjusted given the user state. We leave more advanced inference reward settings for future work.

4 EXPERIMENTS

In this section, we perform experiments on two e-commerce datasets to verify the effectiveness of the PRL learning paradigm. We aim to answer the following research questions:

RQ1: How does PRL perform when instantiated with different sequential recommendation models?

RQ2: What is the effect of the supervised attentive learning, including the self-attentive block and the weighted loss function?

RQ3: How do the prompt reward settings in the inference stage affect the PRL performance?

4.1 Experimental Settings

4.1.1 Datasets. Experiments are conducted on two public accessible datasets: Challenge15[4] and RetailRocket[5].

Challenge15. This dataset comes from the RecSys Challange 2015. In it, each user–item interaction session contains a sequence of user click or purchases behaviours. Sessions whose length are shorter than 3 items or longer than 50 are removed. Then 200k sessions are randomly sampled to obtain a dataset containing 1,110,965 clicks and 43,946 purchases upon 26,702 items.

RetailRocket. This dataset contains sequential data of user's behaviour in a e-commerce website; where users view and add items to a shopping cart. For simplicity, we treat views as clicks and adding to a cart as a purchase. Items which are interacted less than 3 times are removed. Sequences whose length is shorter than 3 or longer than 50 items are also removed. The processed dataset contains 1,176,680 clicks and 57,269 purchases over 70,852 items. Table 1 presents these datasets' detailed statistics.

[4]https://recsys.acm.org/recsys15/challenge/
[5]https://www.kaggle.com/retailrocket/ecommerce-dataset

Algorithm 2 Overall Training procedure of PRL

Input: user-item interaction sequence set $\mathcal{D}_s$, reward settings
Output: all parameters in the learning space θ

```
Initialize all trainable parameters
Generate D_p according to Algorithm 1
repeat
    Draw a mini-batch of {R_t, x_{1:t}, t}, x_{t+1} from D_p
    Compute P_t according to Eq.(8)-Eq.(10)
    Compute s̃_t according to Eq.(12)-Eq.(13)
    Compute loss function L according to Eq.(14)-Eq.(15)
    for each parameter ϑ ∈ θ do
        Compute ∂L/∂ϑ on the mini-batch by back-propagation
        Update ϑ ← ϑ − η · ∂L/∂ϑ
    end for
until converge
return all parameters in θ
```

Table 1: Dataset statistics.

Dataset	Challenge15	RetailRocket
#sequences	200,000	195,523
#items	26,702	70,852
#clicks	1,110,965	1,176,680
#purchase	43,946	57,269

4.1.2 Evaluation protocols. PRL implements offline training of a RL-based recommendation agent. As a result, the experiments focus is offline evaluation of the PRL agent. The ratio of training, validation, and test set is 8:1:1. We use the same data splits as [43]. For validation and testing, the evaluation is performed by providing the agent with previous user-item interaction sequences and the generated prompt reward from Eq.(16). Then we check the rank of the ground-truth action (i.e., interacted items) for the next step. The ranking is performed among the whole item set. Each experiment is repeated 3 times, and the average performance is reported.

For the main results, the recommendation performance is measured by hit ratio (HR) and normalized discounted cumulative gain (NDCG). HR@k is a recall-based metric, measuring whether the ground-truth action is in the top-k positions of the recommendation list [43]. We can define HR for clicks as:

$$\text{HR}(\texttt{click}) = \frac{\#\text{hits among clicks}}{\#\text{clicks}} \tag{17}$$

HR(purchase) is then defined similarly to HR(click), except that we replace numbers of clicks with purchases [43]. NDCG is a rank weighted metric which assigns higher scores to top ranked positions in the recommendation list [21].

4.1.3 Baselines. We instantiate PRL with four renowned deep learning-based sequential recommendation models, including RNN-based models, CNN-based models, and attention-based models to verify the generalization ability of the proposed PRL.

- GRU [14]: This method utilizes a GRU to model user–item interactions. The hidden state of the final timestamp is regarded as the environment state, as shown in Eq. (8).
- Caser [40]: This is a CNN-based method which applies convolutions on the item embedding sequence. Caser is effective at capturing skipping signals between interactions.
- NItNet [45]: This method uses a dilated CNN to enlarge the receptive field to learn long sequences. Besides, residual

Table 2: Top-k recommendation performance comparison of different models ($k = 5, 10, 20$) on the Challenge15 dataset. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance p-value < 0.1 compared with the best baseline which is marked with _. The values for normal training, SQN and SAC come from [43], since we use the same data splits and hyperparameter settings.

Models	purchase						click					
	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20
GRU	0.3994	0.2824	0.5183	0.3204	0.6067	0.3429	0.2876	0.1982	0.3793	0.2279	0.4581	0.2478
GRU-SQN	0.4228	0.3016	0.5333	0.3376	0.6233	0.3605	0.3020	**0.2093**	0.3946	0.2394	0.4741	0.2587
GRU-SAC	0.4394	0.3154	0.5525	0.3521	0.6378	0.3739	0.2863	0.1985	0.3764	0.2277	0.4541	0.2474
GRU-PRL	**0.4514***	**0.3214**	**0.5673***	**0.3593**	**0.6525***	**0.3809***	**0.3027**	0.2086	**0.3967**	**0.2398**	**0.4755**	**0.2598**
Caser	0.4475	0.3211	0.5559	0.3565	0.6393	0.3775	0.2728	0.1896	0.3593	0.2177	0.4371	0.2372
Caser-SQN	0.4553	0.3302	0.5637	0.3653	0.6417	0.3862	0.2742	0.1909	0.3613	0.2192	0.4381	0.2386
Caser-SAC	0.4866	0.3527	0.5914	0.3868	0.6689	0.4065	0.2726	0.1894	0.3580	0.2171	0.4340	0.2362
Caser-PRL	**0.4938***	**0.3555**	**0.6052***	**0.3920***	**0.6914***	**0.4138***	**0.3074***	**0.2121***	**0.4028***	**0.2431***	**0.4838***	**0.2637***
NItNet	0.3632	0.2547	0.4716	0.2900	0.5558	0.3114	0.2950	0.2030	0.3885	0.2332	0.4684	0.2535
NItNet-SQN	0.3845	0.2736	0.4945	0.3094	0.5766	0.3302	0.3091	0.2137	0.4037	0.2442	0.4835	0.2645
NItNet-SAC	0.3914	0.2813	0.4964	0.3155	0.5763	0.3357	0.2977	0.2055	0.3906	0.2357	0.4693	0.2557
NItNet-PRL	**0.4295***	**0.3098***	**0.5405***	**0.3460***	**0.6351***	**0.3701***	**0.3280***	**0.2273***	**0.4248***	**0.2588***	**0.5028***	**0.2786***
SASRec	0.4228	0.2938	0.5418	0.3326	0.6329	0.3558	0.3187	0.2200	0.4164	0.2515	0.4974	0.2720
SASRec-SQN	0.4336	0.3067	0.5505	0.3435	0.6442	0.3674	**0.3272**	**0.2263**	**0.4255**	**0.2580**	**0.5066**	**0.2786**
SASRec-SAC	0.4540	0.3246	0.5701	0.3623	0.6576	0.3846	0.3130	0.2161	0.4114	0.2480	0.4945	0.2691
SASRec-PRL	**0.4681***	**0.3360***	**0.5927***	**0.3768***	**0.6893***	**0.4013***	0.3239	0.2246	0.4219	0.2565	0.5029	0.2770

Table 3: Top-k recommendation performance comparison of different models ($k = 5, 10, 20$) on the RetailRocket dataset. NG is short for NDCG. Boldface denotes the highest score. $*$ denotes the significance p-value < 0.1 compared with the best baseline which is marked with _. The values for normal training, SQN and SAC come from [43], since we use the same data splits and hyperparameter settings.

Models	purchase						click					
	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20
GRU	0.4608	0.3834	0.5107	0.3995	0.5564	0.4111	0.2233	0.1735	0.2673	0.1878	0.3082	0.1981
GRU-SQN	0.5069	0.4130	0.5589	0.4289	0.5946	0.4392	0.2487	0.1939	0.2967	0.2094	0.3406	0.2205
GRU-SAC	0.4942	0.4179	0.5464	0.4341	0.5870	0.4428	0.2451	0.1924	0.2930	0.2074	0.3371	0.2186
GRU-PRL	**0.5486***	**0.4640***	**0.5972***	**0.4798***	**0.6284***	**0.4879***	**0.2805***	**0.2165***	**0.3325***	**0.2336***	**0.3821***	**0.2462***
Caser	0.3491	0.2935	0.3857	0.3053	0.4198	0.3141	0.1966	0.1566	0.2302	0.1675	0.2628	0.1758
Caser-SQN	0.3674	0.3089	0.4050	0.3210	0.4409	0.3301	0.2089	0.1661	0.2454	0.1778	0.2803	0.1867
Caser-SAC	0.3871	0.3234	0.4336	0.3386	0.4763	0.3494	0.2206	0.1732	0.2617	0.1865	0.2999	0.1961
Caser-PRL	**0.5277***	**0.4403***	**0.5742***	**0.4554***	**0.6124***	**0.4653***	**0.2770***	**0.2158***	**0.3296***	**0.2328***	**0.3774***	**0.2450***
NItNet	0.5630	0.4630	0.6127	0.4792	0.6477	0.4881	0.2495	0.1906	0.2990	0.2067	0.3419	0.2175
NItNet-SQN	0.5895	0.4860	**0.6403**	0.5026	**0.6766**	0.5118	0.2610	0.1982	0.3129	0.2150	0.3586	0.2266
NItNet-SAC	0.5895	0.4985	0.6358	0.5162	0.6657	0.5243	0.2529	0.1964	0.3010	0.2119	0.3458	0.2233
NItNet-PRL	**0.5976***	**0.5095***	0.6386	**0.5229**	0.6674	**0.5302**	**0.2812***	**0.2180***	**0.3343***	**0.2353***	**0.3825***	**0.2475***
SASRec	0.5267	0.4298	0.5916	0.4510	0.6341	0.4618	0.2541	0.1931	0.3085	0.2107	0.3570	0.2230
SASRec-SQN	**0.5681**	0.4617	**0.6203**	0.4806	**0.6619**	0.4914	0.2761	0.2104	0.3302	0.2279	0.3803	0.2406
SASRec-SAC	0.5623	0.4679	0.6127	0.4844	0.6505	0.4940	0.2670	0.2056	0.3208	0.2230	0.3701	0.2355
SASRec-PRL	0.5612	**0.4737***	0.6127	**0.4905***	0.6564	**0.5016***	**0.2867***	**0.2201***	**0.3415***	**0.2379***	**0.3952***	**0.2515***

connections are introduced to increase the network depth. NItNet achieves good performances with high efficiency.

- SASRec [22]: This baseline is attention-based and uses the Transformer [41] decoder. The output of the Transformer is treated as the state for the previous sequence.

Each model is trained with the following approaches:

- Normal: Train the model with the normal cross-entropy loss.
- SQN [43]: Self-supervised Q-learning is a recently proposed offline RL learning method which combines supervised learning with Q-learning through a shared base model.

Table 4: Effect of the self-attentive block. Boldface is the highest score. $*$ denotes p-value < 0.1 compared with PRL.

	Methods	purchase						click					
		HR@5	NG@5	HR@10	NG@10	HR@20	NG@20	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20
Challenge15	PRL-mean	0.4389*	0.3128*	0.5477*	0.3483*	0.6302*	0.3692*	0.2914*	0.2028*	0.3783*	0.2310*	0.4520*	0.2497*
	PRL-MLP	0.4481	0.3198	0.5618	0.3568	0.6504	**0.3812**	0.2840*	0.1962*	0.3761*	0.2261*	0.4584*	0.2469*
	PRL	**0.4514**	**0.3214**	**0.5673**	**0.3593**	**0.6525**	0.3809	**0.3027**	**0.2086**	**0.3967**	**0.2398**	**0.4755**	**0.2598**
RetailRocket	PRL-mean	0.5176*	0.4480*	0.5591*	0.4615*	0.5924*	0.4699*	0.2469*	0.1940*	0.2907*	0.2082*	0.3337*	0.2191*
	PRL-MLP	0.4890*	0.4090*	0.5373*	0.4247*	0.5817*	0.4360*	0.2439*	0.1899*	0.2912*	0.2052*	0.3362*	0.2166*
	PRL	**0.5486**	**0.4640**	**0.5972**	**0.4798**	**0.6284**	**0.4879**	**0.2805**	**0.2165**	**0.3325**	**0.2336**	**0.3821**	**0.2462**

- SAC [43]: Self-supervised actor-critic further extends SQN by using the Q-learning part as a critic to re-weight the supervised learning-based actor.
- PRL: Our proposed method.

4.1.4 Parameter settings. On our experiments using both datasets, we limit the model's input to only use the last 10 interacted items at a time, i.e. using only $x_{t-10:t}$ as our model's input. For sequences whose lengths are less than 10, we pad these sequences with a padding token. The Adam optimizer [23] is used to train all models, with batches of size 256. The learning rate is set as 0.01 for RC15 and 0.005 for RetailRocket. For a fair comparison, the item embedding size is set as 64 for all models and all training methods. For GRU, the size of the hidden state is set as 64. For Caser, we use 1 vertical convolution filter and 16 horizontal filters whose heights are set from {2,3,4} according to the original paper [40]. For NextItNet, we use the code published by its authors and keep the settings unchanged. For SASRec, the number of heads in self-attention is set to 1, following the original paper [22]. The drop-out ratio is tuned among [0,0.1,0.2,0.3,0.4,0.5] since we observed that continuing increasing the drop ratio would affect the model performance. For SQN and SAC, we use the exact same setting with their original paper [43]. For PRL, the prompt's reward setting used at inference time is set as $\mu = 2$ and $\epsilon = 0$. The reward for purchases is set as $r_p = 1.0$ and the reward for clicks is set as $r_c = 0.2$. Note that the hyperparameters of recommendation models are kept exactly the same across all training approaches for a fair comparison. Further, by keeping PRL's hyperparameters constant across our experiments, we show that it can be instantiated with different models without exhaustive hyperparameter refinement.

4.2 Performance Comparison (RQ1)

Tables 2 and 3 show a comparison of the top-k recommendation performances on Challenge15 and RetailRocket, respectively. The values for normal training, SQN and SAC come from [43], since we use the same data splits and hyperparameter settings.

We observe that on the Challenge15 dataset, the proposed PRL method achieves the best performance in almost all cases except for the click prediction when integrating with the SASRec model, in which case SQN achieves the highest scores. However the performance gap between SASRec-SQN and SASRec-PRL for click prediction is very small and both of the two methods over-perform normal training. The reason of similar click prediction performance between SASRec-SQN and SASRec-PRL could be that the self-attention based SASRec itself is a powerful base model and the candidate item set in the Challenge15 dataset (i.e., 26,702) is relatively small, so both PRL and SQN have been pushed to the similar almost optimal performance. However, we can see that for purchase predictions, SASRec-PRL still achieves significant performance improvement. It demonstrates that PRL effectively improves the offline training performance of RL-based recommender systems.

On the RetailRocket dataset, we can see that PRL also achieves the highest scores in almost all situations. PRL achieves the highest NDCG in all cases. This demonstrates that PRL tends to push the items which have a higher purchase reward to the top ranking positions of the recommendation list. The biggest performance improvement on RetailRocket is achieved when PRL is instantiated with the Caser model. This further verifies the effectiveness and the generalization ability of PRL.

To conclude, PRL consistently and significantly improves the offline learning performance for RL-based recommendation tasks and can be applied for various sequential recommendation models.

4.3 Ablation Study (RQ2)

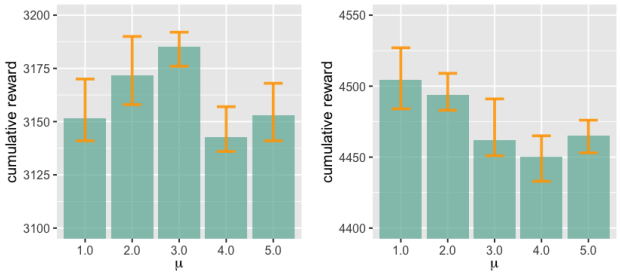
4.3.1 Effect of the self-attentive block. PRL uses a self-attentive block to map a prompt to a corresponding action. In this section, we conduct experiments to verify the effect of this block. We replace the self-attentive block with mean-pooling (i.e., PRL-mean) or a multi-layer perceptron (MLP) (i.e.,PRL-MLP). Table 4 shows the performance comparison when using GRU as the base sequential model. Results of other models lead to the same conclusion. We can see that PRL with the self-attentive block achieves the best performance with significant improvement. This demonstrates the effectiveness of the self-attentive block. Besides, comparing the results with Table 2 and Table 3, we can see that PRL-mean and PRL-MLP achieve better performance than the naive GRU. It further demonstrates the involvement of reward prompt-based learning is effective to improve the recommendation performance.

4.3.2 Effect of the weighted loss. PRL uses the immediate reward r_t to re-weight the supervised training loss so that actions with higher reward would account for larger weights. In this subsection, we conduct experiments to illustrate the effect of this re-weighting schema. We compare the results of PRL without any re-weighting (i.e., PRL-w/o) and PRL re-weighted by the cumulative reward (i.e., PRL-cumu). Table 5 shows the performance comparison when using GRU as the base sequential model. We can see that on the

Table 5: Effect of the weighted loss. PRL-w/o denotes training the agent with PRL but without any re-weighting. PRL-cumu means using the cumulative reward to re-weight the loss. $*$ denotes p-value < 0.1 compared with PRL.

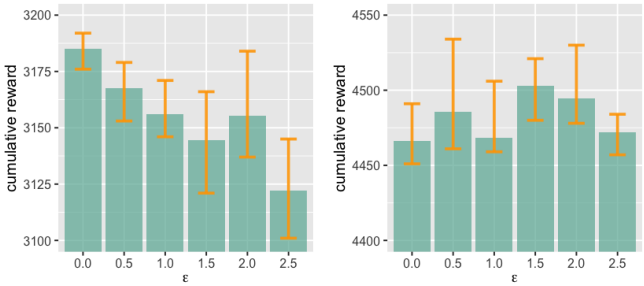
	Methods	purchase						click					
		HR@5	NG@5	HR@10	NG@10	HR@20	NG@20	HR@5	NG@5	HR@10	NG@10	HR@20	NG@20
Challenge15	PRL-w/o	0.4143*	0.2905*	0.5227*	0.3259*	0.6141*	0.3491*	**0.3135***	**0.2166***	**0.4080***	**0.2473***	**0.4889***	**0.2678***
	PRL-cumu	0.4026*	0.2769*	0.5051*	0.3103*	0.5955*	0.3332*	0.2887*	0.1984*	0.3792*	0.2278*	0.4551*	0.2471*
	PRL	**0.4514**	**0.3214**	**0.5673**	**0.3593**	**0.6525**	**0.3809**	0.3027	0.2086	0.3967	0.2398	0.4755	0.2598
RetailRocket	PRL-w/o	0.5193*	0.4267*	0.5717*	0.4438*	0.6154*	0.4548*	0.2780	0.2139	**0.3350**	0.2324	**0.3841**	0.2448
	PRL-cumu	0.4951*	0.4017*	0.5555*	0.4214*	0.5954*	0.4316*	0.2596*	0.1964*	0.3130*	0.2137*	0.3611*	0.2259*
	PRL	**0.5486**	**0.4640**	**0.5972**	**0.4798**	**0.6284**	**0.4879**	**0.2805**	**0.2165**	0.3325	**0.2336**	0.3821	**0.2462**

Challenge15 dataset, PRL-w/o achieves the highest scores for click prediction while PRL achieves the best purchase prediction. On RetailRocket, PRL-w/o and PRL have similar performance for click prediction but PRL performs better for purchase prediction. The results demonstrate that the re-weighting schema of PRL successfully helps the model to recommend more purchased items with higher immediate reward. Regarding PRL-cumu, we can see that its performance is worse than PRL-w/o and PRL. The reason could be that the cumulative reward has much higher variance. So directly using the cumulative reward to re-weight the loss cannot boost the agent performance. Besides, comparing these results to Tables 2 and 3, we can see that even PRL-w/o achieves better recommendation performance than the naive GRU. This further demonstrate the effectiveness of the proposed prompt-based learning.



(a) Challenge15. **(b) RetailRocket.**

Figure 4: Effect of the inference reward expectation μ.



(a) Challenge15. **(b) RetailRocket.**

Figure 5: Effect of the inference reward deviation ϵ.

4.4 Prompt Reward Investigation (RQ3)

In this subsection, we conduct experiments to see how the inference reward settings affect the model performance. We report the cumulative reward@1 in the test set, which measures how much cumulative reward we can get from the top-1 position of our recommendation list. Figure 4a and Figure 4b show the effect of reward expectation μ on Challenge15 and RetailRocket, respectively. We can see that on the Challenge15 dataset, the cumulative reward increases at beginning and then decreases. While on the RetailRocket dataset, the cumulative reward keeps decreasing with higher reward expectations. This demonstrates that a larger reward expectation sometimes improves the inference performance, but a reward expectation too large can be harmful. Figures 5a and 5b illustrate the effect of the reward deviation ϵ, which can be seen as an exploration factor. We can see that on Challenge15, a large ϵ reduces the recommendation performance. While on RetailRocket, a larger ϵ can slightly improve the inference performance. Combined, the results of Figures 4 and 5 suggest that Challenge15's users may prefer items with high reward expectation and little exploration. On the other hand, RetailRocket's users may tend to prefer more exploration.

5 RELATED WORK

Markov Chain (MC) models [13, 26, 33] and factorization-based methods [15, 32] were widely used for next item recommendation tasks in the past. However, such shallow models cannot effectively capture complex sequential signals [40, 45]. Recently, deep learning-based sequential models have been widely investigated for next item recommendation. [14] proposed to model user previous interactions by GRU. [40] and [45] are based on CNNs. Besides, [22, 39] exploited self-attention and Transformer-based architectures [41]. Generally speaking, all we need is a model G (described in section 2.1), whose input is a sequence of previous user–item interactions, while the output is a hidden state $\mathbf{s}$ that describes the user's state. The proposed PRL thus serves as a general learning paradigm, and its model G can be instantiated with any of a diverse pool of sequential models.

RL has been previously applied for recommendation. To perform offline learning from historical data, existing works mainly focus on using IPS score [7] and model-based simulation [8, 16, 47]. Besides, [43] proposed self-supervised reinforcement learning for recommendation; given a standard supervised generative sequential model, they introduce an additional output layer which is trained with standard Q-learning to bias the model towards the desired reward expectation. Furthermore, offline RL algorithms are attracting more and more research efforts. [10] proposed batch constrained Q-learning, forcing the agent to generate in-distribution actions. [27] proposed conservative Q-learning to avoid the over-estimation of Q-values. [34, 37] proposed upside-down RL to transform RL into a form of supervised learning. Recently, [6, 20] proposed to use Transformers to model the RL problem as a big sequence modeling task. While similar to our proposed prompt-inspired methodology, these methods were not tailored to perform next item recommendation, and could encounter difficulties if applied to a setting with highly dynamic user states [25] and a large action space [19].

6 CONCLUSION AND FUTURE WORK

We propose prompt-based reinforcement learning for the offline training of RL-based next item recommendation agents. We theoretically analyse the offline training challenge when exploiting RL for recommendation. Then we propose to use the historical offline data as a knowledge base and then formulate the recommendation task as a question of which action should be taken if the prompt reward is expected to be achieved under the state observation. The proposed PRL can be trained through a simple supervised fashion. We implement PRL with four renowned sequential recommendation models and conduct experiments on two real-world datasets. Experimental results demonstrate the effectiveness of our proposed method. Future work includes online tests and more advanced prompt generation. Besides, we are also interested in investigating adaptive prompt reward settings for model inference.

ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China (2020YFB1406704), the Natural Science Foundation of China (61902219, 61972234, 62072279, 62102234), the Key Scientific and Technological Innovation Program of Shandong Province with grant No.2019JZZY010129, the Natural Science Foundation of Shandong Province (ZR2021QF129), the Tencent WeChat Rhino-Bird Focused Research Program (JR-WXG-2021411), the Fundamental Research Funds of Shandong University, the Shandong University multidisciplinary research and innovation team of young scholars (2020QNQT017), and Meituan. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] M Mehdi Afsar, Trafford Crump, and Behrouz Far. 2021. Reinforcement learning based recommender systems: A survey. *arXiv preprint arXiv:2101.06286*.
[2] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. 2017. Generalization and equilibrium in generative adversarial nets (gans). In *International Conference on Machine Learning*. PMLR, 224–232.
[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.
[4] Richard Bellman. 1966. Dynamic programming. *Science* 153, 3731, 34–37.
[5] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2020. Bias and debias in recommender system: A survey and future directions. *arXiv preprint arXiv:2010.03240*.
[6] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*.
[7] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 456–464.
[8] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2019. Generative Adversarial User Model for Reinforcement Learning Based Recommendation System. In *International Conference on Machine Learning*. 1052–1061.
[9] Xiaocong Chen, Lina Yao, Julian McAuley, Guanglin Zhou, and Xianzhi Wang. 2021. A survey of deep reinforcement learning in recommender systems: A systematic review and future directions. *arXiv preprint arXiv:2109.03540*.
[10] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 2052–2062.
[11] Hado V Hasselt. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems*. 2613–2621.
[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[13] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining*. IEEE, 191–200.
[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *4th International Conference on Learning Representations*.
[15] Balázs Hidasi and Domonkos Tikk. 2016. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery* 30, 2, 342–371.
[16] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in neural information processing systems*. 4565–4573.
[17] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 368–377.
[18] Jin Huang, Harrie Oosterhuis, Maarten de Rijke, and Herke van Hoof. 2020. Keeping dataset biases out of the simulation: A debiased simulator for reinforcement learning based recommender systems. In *Fourteenth ACM Conference on Recommender Systems*. 190–199.
[19] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 2019. SlateQ: A tractable decomposition for reinforcement learning with recommendation sets. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. ijcai.org, 2592–2599.
[20] Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline Reinforcement Learning as One Big Sequence Modeling Problem. In *Advances in Neural Information Processing Systems*.
[21] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4, 422–446.
[22] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining*. IEEE, 197–206.
[23] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, 2015*.
[24] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.
[25] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 447–456.
[26] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8, 30–37.
[27] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*.
[28] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.
[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540, 529.

[30] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. 2016. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*. 1054–1062.

[31] Emilio Parisotto, H Francis Song, Jack W Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. 2020. Stabilizing Transformers for Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning,2020 (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 7487–7498.

[32] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[33] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 811–820.

[34] Juergen Schmidhuber. 2019. Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions. *arXiv preprint arXiv:1912.02875*.

[35] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep, 1265–1295.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587, 484.

[37] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. 2019. Training agents using upside-down reinforcement learning. In *NeurIPS Deep Reinforcement Learning Workshop*.

[38] Dusan Stamenkovic, Alexandros Karatzoglou, Ioannis Arapakis, Xin Xin, and Kleomenis Katevas. 2021. Choosing the Best of Both Worlds: Diverse and Novel Recommendations through Multi-Objective Reinforcement Learning. In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining*.

[39] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[40] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 565–573.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[42] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4, 229–256.

[43] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. 2020. Self-supervised reinforcement learning for recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 931–940.

[44] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. 2021. Supervised Advantage Actor-Critic for Recommender Systems. In *Proceedings of the 15th ACM International Conference on Web Search and Data Mining*.

[45] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 582–590.

[46] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. " Deep reinforcement learning for search, recommendation, and online advertising: a survey" by Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator. *ACM SIGWEB Newsletter* Spring, 1–15.

[47] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2810–2818.