SARDINE: A Simulator for Automated Recommendation in Dynamic and Interactive Environments

ROMAIN DEFFAYET, Naver Labs Europe, Meylan, France and University of Amsterdam, Amsterdam, Netherlands
THIBAUT THONET, Naver Labs Europe, Meylan, France
DONGYOON HWANG, Korea Advanced Institute of Science and Technology, Daejeon, South Korea
VASSILISSA LEHOUX, Naver Labs Europe, Meylan, France
JEAN-MICHEL RENDERS, Naver Labs Europe, Meylan, France
MAARTEN DE RIJKE, University of Amsterdam, Amsterdam, The Netherlands

Simulators can provide valuable insights for researchers and practitioners who wish to improve recommender systems, because they allow one to easily tweak the experimental setup in which recommender systems operate, and as a result lower the cost of identifying general trends and uncovering novel findings about the candidate methods. A key requirement to enable this accelerated improvement cycle is that the simulator is able to span the various sources of complexity that can be found in the real recommendation environment that it simulates.

With the emergence of interactive and data-driven methods – e.g., reinforcement learning or online and counterfactual learning-to-rank – that aim to achieve user-related goals beyond the traditional accuracy-centric objectives, adequate simulators are needed. In particular, such simulators must model the various mechanisms that render the recommendation environment dynamic and interactive, e.g., the effect of recommendations on the user or the effect of biased data on subsequent iterations of the recommender system. We therefore propose SARDINE, a flexible and interpretable recommendation simulator that can help accelerate research in interactive and data-driven recommender systems. We demonstrate its usefulness by studying existing methods within nine diverse environments derived from SARDINE, and even uncover novel insights about them.

CCS Concepts: • **Information systems** → **Recommender systems**.

Additional Key Words and Phrases: Interactive recommender systems, Simulator, Reinforcement learning

1 INTRODUCTION

Recommender systems must match users and items based on item content and user preferences, so as to provide users with content that fulfills a consumption need or carries relevant information given user preferences [35]. In other words, they need to learn the semantic information [45] that explains why a certain user is attracted to a certain item, usually by leveraging user features, item content or logged interactions. However, by restricting the scope of recommender systems to a static semantic matching task one would ignore a crucial part of the recommendation task: converting semantic understanding of users and items into increased value for the user, as

Authors' addresses: Romain Deffayet, Naver Labs Europe, Meylan, France, IRLab and University of Amsterdam, Amsterdam, Netherlands, romain.deffayet@naverlabs.com; Thibaut Thonet, Naver Labs Europe, Meylan, Auvergne-Rhône-Alpes, France, thibaut.thonet@naverlabs.com; Dongyoon Hwang, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, godnpeter@kaist.ac.kr; Vassilissa Lehoux, Naver Labs Europe, Meylan, Auvergne-Rhône-Alpes, France, vassilissa.lehoux@naverlabs.com; Jean-Michel Renders, Naver Labs Europe, Meylan, Auvergne-Rhône-Alpes, France, jean-michel.renders@naverlabs.com; Maarten de Rijke, University of Amsterdam, Amsterdam, The Netherlands, m.derijke@uva.nl.

ACM 2770-6699/2024/4-ART
https://doi.org/10.1145/3656481

well as for content providers and other potential stakeholders. Such value may be measured by, e.g., click-through rate, user satisfaction, retention rate, or fairness metrics.

This concern has led to the emergence of methods that consider beyond-accuracy goals [22, 33] and that often view recommendation as a dynamic and interactive task [15]. First, recommender systems are often trained from user interaction data, either in an online [47] or offline [13] fashion. As a result, recommender systems must learn to deal with noisy user feedback [53], limited knowledge about new users in the cold start scenario [26], as well as potential biases in user behavior that may impact the training data [13]. Second, the items consumed by a user may have an effect on the user state [2, 11, 12]. They could alter user preferences – by developing a user's interest in a topic, by educating users about a topic in a way that encourages them to explore more advanced content, or by changing their perspective on other items, for instance by sparking their interest or instead by reducing it. Items could also temporarily affect user behaviors, for instance by causing boredom, which subsequently reduces user interest and engagement in the platform [2]. Third, exogenous factors may change the value of items and the preferences of users, yielding an ever-changing dynamic environment [19].

1.1 The role of simulators in recommender systems research

In order to account for the dynamic and interactive aspects of recommendation, various approaches have been proposed, including contextual bandits [1, 28], reinforcement learning (RL) [5, 11, 12], active learning [40], counterfactual learning-to-rank [13, 24], and click modeling [6, 8, 9]. These approaches are trained from user data, and it has been shown that they should not be evaluated solely on accuracy-centric benchmarks [10, 22, 48] as these miss the potential benefits brought by beyond-accuracy methods.

While online evaluation [44, 55] remains a gold standard – when done right [23] – to evaluate the impact of recommendation models on user-related metrics, most researchers do not have access to a live recommendation system. Moreover, the potential degradation in user satisfaction and revenue induced by online experiments may limit the possibility to conduct such an evaluation, especially in a research setting where many experiments are needed to improve on the current version of the recommender system.

In that case, prior work [10, 25, 49] has advised to either resort to off-policy evaluation (OPE), which consists in evaluating the target system using data collected with the original system, or otherwise to conduct experiments in a simulated environment. Simulators are by definition synthetic, at least partially, and good performance obtained in a simulator is therefore no guarantee of success in the live system. However, their value lies in the ability to control relevant parameters in a way that spans the potential dynamics encountered in the real environment. Indeed, tweaking parameters and observing their effect on candidate methods allows one to identify general trends and study important research topics: regimes of success and failure (e.g., low data, high bias), robustness to environment features that may be observed in the real world (e.g., noise, distribution shifts), generalizability of the results, etc.

In that sense, simulated evaluation can even be less opaque than OPE and online evaluation, as observing variables that are normally not accessible to the practitioner can help better interpret the observed performance of the candidate systems. In order to deliver these benefits, we argue that simulators should be:

(1) Configurable in a way that is easily interpretable to the practitioner,
(2) Able to span a large part of the various forms of complex behavior commonly found in the real environment.

In practice, we draw up a list of specifications that we use as a goalpost for designing our simulator:

Specifications 1.1 – Our simulator should satisfy the following requirements:

- **Comprehensiveness:** Most of the important research questions for interactive recommender systems can be studied in one core simulated engine;
- **Interpretability:** One or a few well-defined parameters can control a specific aspect of interest in recommender system research, i.e., the simulator should be interpretable and controllable;

- **Effect isolation:** The effect of individual parameters or individual algorithmic modules can be singled out, so as to allow the focused study of one aspect of the environment (e.g., noise, user drift, etc.) or one part of the method (e.g., user and item representation, decision-making module, etc.);
- **Non-triviality:** The simulated task should not be trivially solved by off-the-shelf baselines; and
- **Configurability:** Additions and changes to the existing simulator should be easy enough to enable deeper studies or new research questions.

In order to fulfill the specifications, and before engaging with simulator design, we must define the scope of the research we wish to enable with such a simulator. We therefore define the research agenda our simulator addresses in the next section.

1.2 A research agenda for interactive recommender systems

We identify four overarching research topics (RTs) that we believe to be crucial for interactive recommender systems (RSs) research, and that can be studied in our simulator. We also connect them to variants of our simulator that are particularly well-suited to study them:

(RT1) **How to enable multi-step reasoning and control user-related metrics in the long run?** In a dynamic and interactive environment, shifting dynamics and delayed consequences of actions prompt RS designers to adopt a control paradigm, where target variables such as user satisfaction, revenue, or fairness-related variables must be optimized and kept at a desired value in the long run. This requires multi-step reasoning, i.e., thinking ahead of time about future consequences of recommendations formulated at the present time. Many approaches have been proposed to tackle multi-step reasoning, notably with reinforcement learning [5, 11, 12, 54].This research topic can be studied thanks to the interactive environments we release, i.e., `SingleItem-Bored`, `SlateTopK-Bored`, `SlateTopK-BoredInf`, `SlateTopK`, `SlateTopK-Uncertain`, `SlateRerank-Bored`.

(RT2) **How to learn from biased data?** As online learning is often not possible in a large commercial platform, it is common to resort to offline or off-policy learning, by first collecting data in the live environment, and then learning from this data. However, multiple biases arise in the logged data. Due to selection bias, the distribution of items observed in the data is highly imbalanced, including many items that are never or almost never shown to certain users. Additionally, even when feedback is observed, biases in user behavior favor certain items above others, e.g., due to position bias. As a result, training models that do not account for these biases leads to the unfair promotion of already well-exposed items. Learning from data despite these biases is a very active area in information retrieval research, with techniques such as offline reinforcement learning [5, 12, 54], counterfactual learning-to-rank [13, 24], or click modeling [6, 9]. All of our simulated environments can be used for off-policy training, but we notably study this research topic with our `SlateRerank-Static` and `SlateRerank-Bored` environments.

(RT3) **How to make sure that interactive recommender systems are robust to uncertainties of the real-world?** Recommender systems must operate under large amounts of uncertainty coming from multiple sources: in the user feedback and in their evolution after consuming items (e.g., varying mood and personal traits, light scanning of the results), about exogenous factors influencing user behavior and item value (e.g., world events, current context when accessing the platform), about user preferences (e.g., cold start, changing users) and in the policy itself (e.g., business rules, stochastic amortization). Large amounts of uncertainty may hurt the performance of recommender systems and yield disappointing results during the deployment of these models, which has prompted the development of uncertainty-aware methods [26, 37]. Our `SlateTopK-Uncertain`, `SlateTopK-PartialObs` and `SingleItem-PartialObs` allow to study such uncertainties.

(RT4) **How to effectively and efficiently recommend slates (e.g., lists or grids) of items to users?** The

interface of many recommendation platforms requires showing multiple recommendations to users on the same page. This comes with additional challenges as different combinations of items may lead to different short and long-term outcomes. The problem thus becomes combinatorial in nature, which makes the task intractable for most applications. The existing literature discusses slate-specific methods for both training and evaluation of slate recommendation policies [5, 20, 49], including methods that improve on the efficiency of slate recommender systems [36, 42]. It is possible to train slate recommender systems on all our `SlateTopK` and `SlateRerank` environments.

1.3 Our contributions

Our contributions can be summarized as follows:

- We introduce a simulator for automated recommendation in dynamic and interactive environments (SARDINE), which can be used as a flexible core engine for multiple types of simulated experiments in recommender systems research, allowing quicker iterations towards studying, among others, the research topics (RT1–4) mentioned in Section 1.2, i.e., multi-step reasoning, biased data, uncertain dynamics, and slate recommendation.
- We additionally provide nine different environments derived from this simulator, in the form of gymnasium [50] environments, that are already tailored for studying important aspects of recommendation in dynamic and interactive settings.[1]
- We conduct experiments on the nine proposed environments, in order to (i) better describe the main dynamics of the simulator, (ii) provide a testbed for some existing approaches and baselines, and (iii) uncover some novel findings about existing approaches, thereby restating the value of our simulator for effective recommender system research.[2]

Furthermore, we now summarize the expected benefits of releasing our simulator. Indeed, we seek to help accelerate future research, by: (i) providing a playground for researchers to create and test prototypes and therefore iterate more quickly; (ii) enabling quickly building experimental set-ups in order to gain knowledge on specific research questions related to the topics RT1–4 we describe in the paper; and (iii) providing a set of not-yet-solved simulated tasks that trace a path towards progress in recommender systems research (e.g., as Atari games or Go have been for multi-step visual control).

In contrast, we have no intention to: (i) create a realistic simulator of the human mind – besides clearly being an unattainable goal, we argue that it is not necessary to gain perfect knowledge of the actual underlying user model to effectively optimize the target variables (e.g., user engagement). Instead, we propose to study the adaptability and robustness of recommendation agents, with the help of a large array of different simulated settings. (ii) Provide guarantees of live performance. Simulators, whether they are fully- or semi-synthetic, cannot provide guarantees of performance in the live recommender system. They are nonetheless valuable for making progress in recommender systems research, e.g., by studying the robustness of agents and the edge cases where they might struggle, by quickly iterating on simulated tasks that robust recommenders should be able to solve, or even by detecting poorly robust methods before conducting A/B testing in a live system and potentially negatively impacting real users. And (iii) replace offline evaluation on traditional metrics. While a set of diverse simulated experiments offers a unique perspective on the inner workings of recommender systems, simulations must always be complemented with offline and online real-world experiments in order to build a well-rounded assessment of the progress in recommender systems research.

The remainder of the paper is organized as follows. We formally define the recommendation problem of interest in Section 2. We then describe the technical details of the SARDINE simulator in Section 3. Section 4 covers the

[1]The core simulator as well as the proposed environments can be found at https://github.com/naver/sardine.
[2]Our experiments are open-source and can be found at https://github.com/RomDeffayet/SARDINE_Experiments.

details about our experimental setup, which includes the description of the SARDINE environments tested in our experiments as well as the compared approaches. The experiment results are presented and discussed in Section 5. Finally, we compare our proposed SARDINE to existing recommendation simulators in Section 6, and conclude the paper in Section 7.

2 PROBLEM DEFINITION

The problem studied in this paper can be defined as slate recommendation[3] in a dynamic environment. In this scenario, we consider that a user interacts with a recommender system over a session of L steps. In each step, the recommender system presents a slate containing S items from a predefined set $\mathcal{I}$ of cardinal $n_{\mathcal{I}}$ to the user. Based on the affinity between the recommended items and the user preferences, the user decides to click on some or none of the slate items. Information about the interaction and the current user state is then returned to the agent and, based on this, the recommender determines the next slate to recommend. This process can be formulated as a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$ defined as follows:

- A set of states $s \in \mathcal{S}$, which represent the user state and summarize information about the past interactions.
- A set of actions $a \in \mathcal{A}$ corresponding to the possible slates presented by the recommender to the user. This set covers all slates combining items from $\mathcal{I}$, so that $|\mathcal{A}| = \frac{n_{\mathcal{I}}!}{(n_{\mathcal{I}}-S)!}$ for a slate of size S.
- A set of transition probabilities $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, which define the dynamics in the process, i.e., how likely a state $s' \in \mathcal{S}$ is if the recommender takes action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$.
- A (potentially stochastic) reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which we define as the sum of clicks over the recommended slate.

We also define a possibly stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ whose role is to decide what slate a the recommender system should return in a given state s. A trajectory τ is defined as the set of successive states, actions and rewards collected in a session of interactions between a user and a recommender. We denote as $\tau \sim \pi$ the fact that trajectory τ is generated by following the actions provided by policy π. The problem of slate recommendation in a dynamic environment can then be summarized as identifying a policy π^* that maximizes the cumulated reward (also known as *return*) in expectation over possible trajectories, i.e., $\pi^* \in \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{(s,a)\in\tau} R(s, a)\right]$.

In this paper, we introduce a simulator that instantiates the MDP described above to provide a testbed for developping recommendation policies and studying their characteristics in various settings. The proposed simulator is further described in Section 3.

3 SIMULATOR

In this section, we detail the components of our proposed simulator for automated recommendation in dynamic and interactive environments, or SARDINE in short. In SARDINE, we consider a cold-start scenario where each new session corresponds to a new user, generated on-the-fly. This means that we assume no prior knowledge on user profiles before a session starts and that the agent must do some exploration to discover user interests. This design choice is realistic for many recommendation platforms, e.g., when a single device or profile regroups several users – who exhibit diverse preferences over different sessions – or when the platform does not track a user ID for privacy reasons [16].

First, our simulator is initialized by forming synthetic embeddings for the set of recommendable items (Section 3.1). Then, each user session is generated by following these successive steps:

(1) Sample a user embedding for the current session's user (Section 3.1);
(2) Provide an initial recommendation (Section 3.2) or prompt the agent to recommend a slate to the user;

[3]We consider that single-item recommendation is just a special case of slate recommendation with a slate of size one. Therefore our problem formulation also covers this case.

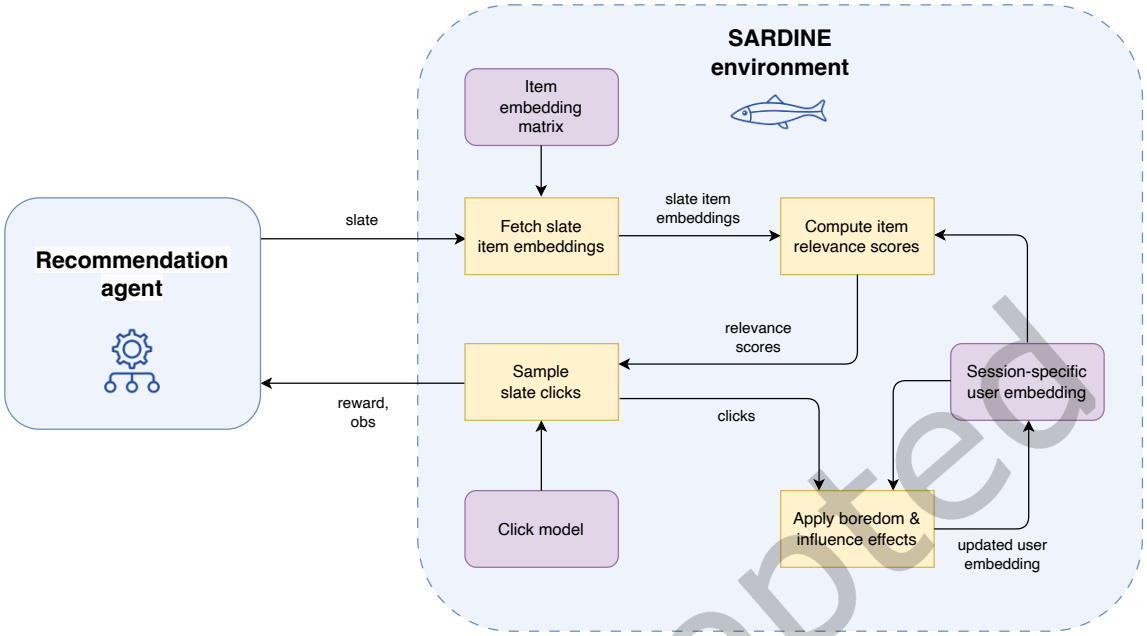


Fig. 1. Diagram summarizing the different components of the proposed SARDINE simulator, and its interaction with the recommendation agent.

(3) Compute the relevance of the items in the slate with respect to the user and sample the clicks on the slate based on items' relevance and rank (Section 3.3);
(4) Update the user embedding to account for the effects of boredom and clicked item influence, if those mechanisms are included in the simulator (Section 3.4);
(5) Repeat steps (2) to (4) until the number of interaction steps reaches the session length L.

We define both a fully observable variant and a partially observable variant for SARDINE, whose differences are detailed in Section 3.5. Moreover, we use the main engine described in this section with different sets of hyperparameters so as to create nine different environments with various characteristics, and targeting various research outcomes. We introduce these environments in Section 4.1.

Fig. 1 illustrates the different components of our simulator and its interactions with the recommendation agent. In Table 1 we additionally provide a description for the hyperparameters of the simulator, which are further defined in the remainder of this section.

3.1 Item and user embeddings

Items and users are assigned randomly-generated sparse embeddings of size $n_{\mathcal{T}} = |\mathcal{T}|$, where $\mathcal{T}$ is the set of topics associated to items and users (defined below). The sparsity enforces a coverage of only a limited number of topics per item and user. The generative process to define the embedding[4] for each item i in the set of items $\mathcal{I}$ is the following:

(1) Sample the item embedding components from a uniform distribution over $[0, 1]$: $\mathbf{e}_i = (\mathbf{e}_{i,1}, \ldots, \mathbf{e}_{i,n_{\mathcal{T}}}) \in \mathbb{R}^{n_{\mathcal{T}}}$ with $\mathbf{e}_{i,j} \sim \text{Unif}([0, 1])$;

[4]To distinguish the item embeddings used in the simulator from the item embeddings that may be learned by an agent, we refer to the former as *ideal item embeddings* when disambiguation is needed.

Table 1. List of the hyperparameters used in the proposed SARDINE simulator, with their description.

Hyperparameter	Description
L	Session length (in time steps).
S	Slate size (in number of items).
$n_{\mathcal{I}}$	Number of items.
$n_{\mathcal{T}}$	Number of topics (and user/item embedding dimension).
λ	Scale hyperparameter for the relevance function.
μ	Shift hyperparameter for the relevance function.
α	Range hyperparameter for item attractiveness.
ϵ	Click propensity for examination probability.
n_{b}	Number of items considered for boredom computation.
t_{b}	Click recency (in time steps) for boredom computation.
τ_{b}	Threshold on topic occurrence for boredom computation.
ω	Weight controlling the influence of clicked items on user.
O	Hyperparameter indicating full or partial state observability.

(2) Sample a number of topics associated to the item (i.e., the number of non-zero components to retain in $\mathbf{e}_i$) equal to either 2 or 3: $n_{\mathcal{T}_i} \sim \text{Unif}(\{2, 3\})$;
(3) Sample the $n_{\mathcal{T}_i}$ topics associated to the item from the topic set $\mathcal{T}$: $\mathcal{T}_i = \{T_{i,1}, \ldots, T_{i,n_{\mathcal{T}_i}}\} \subset \mathcal{T}$ with $T_{i,j} \sim \text{Unif}(\mathcal{T})$ without replacement;
(4) Zero out the components of the item embedding that correspond to non-selected topics (i.e., outside of $\mathcal{T}_i$): $\mathbf{e}_{i,j} := 0$ if $j \in \mathcal{T} \setminus \mathcal{T}_i$;
(5) Normalize the components to have an embedding with unitary Euclidean norm: $\mathbf{e}_i := \frac{\mathbf{e}_i}{\|\mathbf{e}_i\|_2}$.

We denote the *main topic* of item i as T_i^* which corresponds to the dominating component in the item embedding $\mathbf{e}_i$, i.e., $T_i^* = \arg\max_{j \in \mathcal{T}} \mathbf{e}_{i,j}$.

The process to generate a user embedding $\mathbf{e}_u$ for each new session is similar to that of generating an item embedding, with the difference that we allow a user embedding to cover 3, 4, or 5 topics instead of 2 or 3 for items. The rationale for this choice is that a user may be interested in a broad selection of topics whereas an item usually spans a more narrow set of topics (e.g., the number of movie genres a user likes vs. the number of genres a movie belongs to).

3.2 Initial recommendation

The first recommendation the user receives at the beginning of a session is independent of the agent and done directly in the simulator. Given our cold-start setting, i.e., we have no prior knowledge of the user profile, we wish to start the session by probing the preferences of the user. For that purpose, our initial recommendation is simply a slate containing random items. In a real-life scenario, other alternatives could be considered, e.g., by exploiting the popularity of items [3], prior user profile [4] as well as user metadata [31]. We leave the investigation of alternative initial recommendations for future work.

3.3 Relevance computation & click model

In this section, we describe how relevance, i.e., the matching score, is computed for a (user, item) pair. Then we detail how this relevance score is used to sample the clicks and skips on a slate recommended to the user.

Relevance score. The relevance of items presented to a user is calculated based on the dot-product between the item embedding and the user embedding:

$$\text{rel}(i, u) = \mathbf{e}_i^T \mathbf{e}_u. \tag{1}$$

Item attractiveness. To the relevance score we then apply a sigmoid function that is rescaled and shifted to account for the range of values and the desired level of saturation for the function, resulting in an attractiveness score. Compared to the relevance score, the attractiveness of an item reflects click behavior specified by the hyperparameters of the sigmoid; their role is explained below. Formally, the attractiveness of item i for user u is defined as follows:

$$A_{u,i} = \alpha \cdot \sigma(\text{rel}(i, u)) \text{ where } \sigma(x) = \frac{1}{1 + \exp(-\lambda(x - \mu))}. \tag{2}$$

The hyperparameter α is introduced to adjust the range of the attractiveness score. The shift hyperparameter μ ensures that the function outputs a value close to 1 for a highly matching (user, item) pair, and close to 0 for an item totally unrelated to the user. The scale hyperparameter λ controls how steep the sigmoid will be (i.e., how easily the output of the function saturates to 0 or 1). This latter hyperparameter plays a key role for the level of uncertainty in the simulator. Indeed, a lower value of λ implies that the sigmoid σ will be less steep, leading to smaller differences in attractiveness (and, in turn, in click probabilities as detailed below) between relevant and irrelevant items. In other words, the user feedback is more uncertain when λ is low.

In practice, we set the values of λ and μ using the following rules:

(1) A random recommendation policy should almost always propose irrelevant items, i.e., such that $\sigma({\mathbf{e}_i}^T \mathbf{e}_u)$ is generally close to 0 for a randomly selected item i;
(2) An oracle recommendation policy should always propose relevant items, i.e., such that $\sigma({\mathbf{e}_i}^T \mathbf{e}_u)$ is close to 1 for the few top items i that best match user u;
(3) A bored[5] user cannot be satisfied most of the time, even by an oracle recommendation policy, i.e., when user u is in a bored state, $\sigma({\mathbf{e}_i}^T \mathbf{e}_u)$ is much smaller than 1 even for the few top items i that best match u.

Click model. After the attractiveness score for a (user, recommended item) pair has been computed for each item of the slate, the simulator has to decide if this pair leads to a click or not. For that purpose, we consider a position-based click model, i.e., the probability of click is defined by the product of item-specific attractiveness and rank-specific examination probability. More complex click models could be considered and added to the simulator, but we do not wish to provide a catalog of all existing models. Instead, we want to show that the impact of biased data in general is visible in our simulator, taking the position-based model as an example.

Formally, this click probability is expressed as $\mathbb{P}(c \mid u, i, r) = A_{u,i} \times E_r$ for an item i positioned at rank $r \in \{1, \ldots, S\}$ in the slate, with S the slate size. $A_{u,i}$ is the attractiveness of item i for user u defined in Eq. 2 and E_r is the probability that the user examines the items in the slate down to rank r. By default, the examination probability is set to $E_r = \varepsilon^{r-1}$, where the hyperparameter ε defines the rate of decay of the examination probability. The click (or skip) from user u on item i at rank r in the slate is then sampled from the Bernoulli distribution $\text{Bern}(A_{u,i} \times E_r)$.

3.4 Boredom and influence mechanisms

Our SARDINE simulator introduces two long-term mechanisms in the recommendation, which penalize myopic strategies and thus require the agent to consider the consequences of its actions several steps after taking them. The rationale for this choice is to be able to generate benchmarks where reinforcement learning-based agents are a better choice than bandit approaches, which would otherwise be more suitable for a greedy sequential recommendation task as shown in [27]. This goal is motivated by empirical evidence of the limits of greedy methods with respect to, e.g., diversity, and thus their detrimental impact on long-term metrics such as churn

[5]The notion of boredom introduced in this simulator is further detailed in Section 3.4.

rate [2, 12, 34]. The first mechanism we define is referred to as *boredom* and intuitively reflects the fact that a user may become less interested in consuming content (i.e., clicking on items) when the items recommended in successive slates are too similar, similarly to [11, 12]. The second mechanism we consider is the *influence of the clicked items* on the future user behavior: when a user consumes an item, this may shift the user's interest towards the item's topics, as in [e.g., 7]. These two mechanisms are described in more detail below.

Boredom. To determine if a user u gets bored during a session, we consider the items clicked in the last t_b time steps. If there are more than n_b such items, we keep only the n_b most recently clicked items and we record a list of their main topics. Then, if a topic $T \in \mathcal{T}$ occurs more than a threshold of $\tau_\text{b} \leq n_\text{b}$ times in this list, we consider that the user u is bored with respect to topic T. We define two boredom variants that specify the impact on the bored user's behavior: *temporary loss-of-interest boredom* and *churn-and-return boredom*. For the temporary loss-of-interest boredom, the user u who is bored with respect to topic T has their user embedding component $\mathbf{e}_{u,T}$ set to 0 (i.e., this simulates a loss of interest for topic T) for t_b time steps. After this period, we consider that the boredom effect has timed out and the user may be willing to click again on items with main topic T, so the component $\mathbf{e}_{u,T}$ is restored to its previous value. The churn-and-return boredom operates in a similar fashion with the difference that all components of $\mathbf{e}_u$ are set to 0 until the boredom effect times out: this simulates the fact that the user churns the platform (as an all-zero user embedding implies an absence of clicks in our simulator) and then returns after t_b time steps.

Clicked item influence. At each interaction step, the user u is recommended a slate and potentially clicks on some of this slate's items. We denote as $\mathcal{I}_c$ this set of clicked items. We transcribe the influence of clicked items on u's future behavior by updating the user embedding $\mathbf{e}_u$ as a weighted average of the previous user embedding and the mean of the clicked item embeddings: $\mathbf{e}_u := \omega\, \mathbf{e}_u + (1-\omega)\frac{1}{|\mathcal{I}_c|}\sum_{i\in\mathcal{I}_c} \mathbf{e}_i$, where ω is a hyperparameter controlling the amount of influence clicked items have on the user. Intuitively, the influence mechanism causes a drift in user interests and thus makes the recommendation process more dynamic.

3.5 Full observability vs partial observability

Our SARDINE simulator can be used in two modes: either with *full* state observability or with *partial* state observability, which is recorded in the hyperparameter O. The former simulates a Markov decision process (MDP) setting while the latter defines a partially observable Markov decision process (POMDP) setting. In this section, we define the state/observation used in these two cases.

Full observability. In the full observability case, agents have access to the entire information about the user state. Here, the state fed to the agent is defined as the concatenation of 3 vectors:

- The current user embedding, i.e., $\mathbf{e}_u$, which corresponds to the user embedding at the current time step and thus includes the effects of boredom and influence (if those mechanisms are included in the simulator). Size: $n_\mathcal{T}$.
- A histogram indicating the number of times each topic was the main topic of an item among the n_b last clicked items in the most recent t_b time steps. The histogram is normalized by dividing click numbers by the threshold τ_b and by clipping between 0 and 1. Size: $n_\mathcal{T}$.
- A vector indicating the boredom timeout duration (in number of steps) left for each topic. If a topic is not in a bored state for the user, then its default timeout duration is t_b. For topics that triggered boredom in previous steps and whose boredom is still ongoing, the duration will be between 0 and t_b (excluded). This vector is also normalized between 0 and 1 by dividing it by t_b. Size: $n_\mathcal{T}$.

In the state, the current user embedding is used to keep track of the dynamic user preferences, while the histogram and timeout vectors maintain the information about recent item consumption and boredom. The current item embeddings – which represent the actual preferences of a user at a given time – are normally not available in a real-life recommendation scenario. However, studying this fully observable setting enables the practitioner

to single out the impact of the recommendation algorithm, contrarily to the partially observable setting which compounds the effects of algorithm effectiveness and user embedding estimation quality. As we will show in our experiments (Sections 4 and 5), the fully observable case already leads to challenging environments which justifies our choice to include this less realistic scenario.

Partial observability. For the partial observability setting, the agent cannot access the inner workings of the simulator and is only provided a set of observations about the last interaction. The observation returned to the agent is the concatenation of 3 vectors:

- The slate that was recommended by the agent, with the item ID for each slot. Size: S.
- The clicks that the user did on the recommended slate, with 1 or 0 at each slot to indicate a click or a skip, respectively. Size: S.
- The histogram of recent clicked topics, as in the fully observable case. It is realistic to consider this information accessible to the agent as item categories in recommender systems are generally public. Size: $n_{\mathcal{T}}$.

Based on these 3 pieces of information, the agent is able to identify which recommended items led to a click and exploit recently clicked topics to better infer user preferences. However, they are not enough to perfectly determine the user state and the agent may need to incorporate the history of observations in the same session in order to improve its estimation of the user state (which is usually done through state encoders).

4 EXPERIMENTAL SETUP

Now that we have detailed the main components of our simulator in the previous section, we can describe some of its possible instantiations for conducting experiments related to the research agenda of Section 1.2. This section therefore aims (i) to provide guidance for the usage of the simulator, (ii) to define a testbed for studying existing methods along the research topics defined in Section 1.2, and (iii) to demonstrate the simulator's utility for recommendation research by uncovering some novel insights about these methods.

First, Section 4.1 introduces nine recommendation environments instantiated from our simulator, that we use in our experiments. Then, Section 4.2 describes the recommendation agents we seek to compare on the environments. Finally, Section 4.3 summarizes the simulator hyperparameters adopted by the different environments, as well as the agent hyperparameters used in our experiments.

4.1 Simulated environments

To demonstrate possible use cases enabled by SARDINE, we defined nine different environments – each being a variant of our simulator. The characteristics of these nine environments are detailed in Table 2. They are characterized along six dimensions, which are directly linked to the research topics defined in Section 1.2:

(1) The type of recommendation made to the user: single-item recommendation (corresponding to the case where $S = 1$) or slate recommendation ($S > 1$) — **RT4**;
(2) The presence of a boredom mechanism, i.e., users get bored when being presented repeatedly with a similar content, and thus become less likely to click on the related items — **RT1**;
(3) The presence of an influence mechanism, i.e., users are influenced by clicked items in future interaction steps — **RT1**;
(4) The level of click uncertainty, i.e., the degree of stochasticity in the click probabilities, which is controlled by the scale hyperparameter λ in the relevance sigmoid: lowering λ increases the click likelihood on less relevant items (see Section 3.3 for more details) — **RT3**;
(5) The observability, i.e., whether the agent has access to full or partial user state information (MDP or POMDP setting, respectively) as detailed in Section 3.5 — **RT3**;
(6) Whether the task is reranking, in which case there is a limited number of items that are all presented to the

user (i.e., $n_I = S$) and the recommendation agent has to find the best permutation of those items[6] — **RT2**.

Below, we summarize the purpose of each of the nine environments we introduce:

- `SingleItem-Static`: This single-item recommendation environment with static user behavior and full state observability was chosen to showcase an "easy" environment where learned agents should be able to reach optimal performance without difficulty. This environment also provides a good sanity check to validate that a learned agent is working as expected.
- `SingleItem-BoredInf`: This environment augments `SingleItem-Static` with boredom and influence long-term mechanisms, which require the agent to consider multi-turn dynamics to provide effective recommendations. Therefore, this corresponds to a typical RL-based recommendation environment, in an MDP setting.
- `SingleItem-PartialObs`: This is another variant of `SingleItem-Static` that increases the environment's difficulty through partial observability, i.e., the true state is not directly accessible and the agent is only provided with partial observations at each interaction step. This simulates typical sequential recommendation environments based on offline feedback, where the state (i.e., the user embedding) is unknown and recommendations have no causal effect on future user interactions [10].
- `SlateTopK-Bored`: This variant of the simulator includes slate recommendation (as opposed to the single-item recommendation from the previous environments) and a boredom mechanism, with full state observability. It makes this environment suitable to evaluate RL-based slate recommendation methods in an MDP setting.
- `SlateTopK-BoredInf`: This environment is based on `SlateTopK-Bored` with an additional influence mechanism, making the dynamics more complex as clicked items' influence causes a drift in user interests.
- `SlateTopK-PartialObs`: This challenging environment derived from `SlateTopK-BoredInf` includes boredom and influence mechanisms, but also partial observability. The POMDP setting along with the need for RL-based agents to tackle the effects of the long-term mechanisms make this environment a good choice to investigate state encoders as well as RL-based slate recommendation agents.
- `SlateTopK-Uncertain`: In this environment, we start from `SlateTopK-PartialObs` and increase the uncertainty through greater stochasticity in the clicking process. In practice, this is done by reducing the value of the relevance scale hyperparameter λ. We vary λ from its standard value 100 (used in `SlateTopK-PartialObs`) to 10, 5 or 2 to study different levels of click uncertainty.
- `SlateRerank-Static`: This environment is focused on the reranking task described previously and includes static users. Its main purpose is to enable us to study the effect of the ranking order (i.e., the presentation bias) as opposed to the mere effect of including items in the ranking (i.e., the selection bias), as done in `SlateTopK` environments. This environment and its potential variants are therefore particularly suited for click modeling and counterfactual learning-to-rank research.
- `SlateRerank-Bored`: Similarly to `SlateRerank-Static`, this environment provides a testbed for research on presentation biases such as position bias. However, it adds a boredom mechanism so that greedy agents, even with perfectly alleviated position bias, are not optimal. It thus constitutes a way to conduct research on the effect of data biases on, e.g., RL agents.

The set of environments introduced above is not intended to give an exhaustive coverage of all possible combinations allowed by our simulator, but rather to provide a sample of relevant environments highlighting its various possibilities. In particular, we chose these environments to reflect the four research topics introduced in Section 1.2: the inclusion of multi-step mechanisms (in `SingleItem-BoredInf`, `SlateTopK-Bored`,

[6]An example of such a scenario in a real-life recommender system is in a two-stage setting where the recommender first reranks the (limited) set of item categories for the user. Then, in a second step, the recommender identifies the best item to present for each ranked category slot. What we are interested in here is the first reranking step where items correspond to categories.

Table 2. Description of the nine environments studied in our experiments, each corresponding to a variant of our simulator.

Environment name	Rec. type	Boredom	Influence	Click uncertainty	Observability	Reranking
`SingleItem-Static`	Single item	No	No	Low	Full	No
`SingleItem-BoredInf`	Single item	Yes	Yes	Low	Full	No
`SingleItem-PartialObs`	Single item	No	No	Low	Partial	No
`SlateTopK-Bored`	Slate	Yes	No	Low	Full	No
`SlateTopK-BoredInf`	Slate	Yes	Yes	Low	Full	No
`SlateTopK-PartialObs`	Slate	Yes	Yes	Low	Partial	No
`SlateTopK-Uncertain`	Slate	Yes	Yes	Medium to v. high	Partial	No
`SlateRerank-Static`	Slate	No	No	High	Full	Yes
`SlateRerank-Bored`	Slate	Yes	No	High	Full	Yes

`SlateTopK-BoredInf`, `SlateTopK-PartialObs`, and `SlateRerank-Bored`), the biases induced by the item presentation order (in `SlateRerank-Static` and `SlateRerank-Bored`), the uncertainty in the clicks (in `SlateTopK-Uncertain`) and in the user state (in `SingleItem-PartialObs`, `SlateTopK-PartialObs`, and `SlateTopK-Uncertain`), and the recommendation of slates as opposed to single items (`SlateTopK` and `SlateRerank` environments vs. `SingleItem` environments). The precise set of hyperparameters used in each environment are detailed in Section 4.3.

As a side note, in Section 3.4, we defined two types of boredom mechanism: the temporary loss-of-interest boredom and the churn-and-return boredom. In our experiments, we only use the churn-and-return boredom. Indeed, the experiments done in our pilot studies with the two boredom mechanisms lead to similar conclusions on the approaches' relative performance. Therefore, we omit results with the temporary loss-of-interest boredom for the sake of brevity.

4.2 Compared methods

This section presents the different baseline recommendation methods that we re-implemented in SARDINE[7] and tested in our experiments. We sought to include both simple, naive baselines as well as recent and state-of-the-art approaches to highlight the different characteristics and difficulty levels of the environment presented in Section 4.1. Our compared methods include the following:

- **Random:** This simple baseline simply consists in recommending a random slate (or item in the case of `SingleItem` environments) at each interaction step.
- **Greedy Oracle:** This baseline recommends at each step the optimal slate (or item in the case of `SingleItem` environments) based on the current user embedding. The optimal slate contains the S items that maximize the relevance function defined in Section 3.3, ordered by relevance in a top-down fashion. This approach is optimal in a static setting (without boredom and influence). However, it is unable to perform multi-step reasoning in a dynamic setting (with boredom and/or influence) due to its myopic behavior, hence the name Greedy Oracle.
- **REINFORCE + Top-K:** This approach proposed in [5] extends the REINFORCE policy-gradient agent to the slate recommendation problem. It estimates the value of individual items rather than the full slate, thereby making the problem tractable. However, it requires certain assumptions, for instance that the slate receives at most one click and that the items' returns are mutually independent. Since slates can have several clicks in SARDINE, we simply use the first click in the slate for this method. For the `SingleItem` environments, we instead use a standard REINFORCE agent as the top-K addition is not needed.
- **SAC + Top-K:** This method was introduced in [11] as a simple yet strong baseline for slate recommendation.

[7]The implementation for those methods is included in our code at https://github.com/RomDeffayet/SARDINE_Experiments and made available for the sake of reproducibility.

Table 3. Value of the simulator hyperparameters for each of the nine environments used in our experiments. The description of the hyperparameters' meaning and role is detailed in Table 1. An N/A value signals that the phenomenon related to the hyperparameter is absent in this environment (e.g., the influence parameter ω is N/A for the `SlateTopK-Bored` environment which does not include the influence mechanism).

Environment name	Hyperparameter value												
	L	S	$n_{\mathcal{I}}$	$n_{\mathcal{T}}$	λ	μ	α	ϵ	n_{b}	t_{b}	τ_{b}	ω	O
`SingleItem-Static`	100	1	1000	10	100	0.65	1.0	0.85	N/A	N/A	N/A	1.0	full
`SingleItem-PartialObs`	100	1	1000	10	100	0.65	1.0	0.85	N/A	N/A	N/A	1.0	partial
`SingleItem-BoredInf`	100	1	1000	10	100	0.65	1.0	0.85	10	5	5	0.95	full
`SlateTopK-Bored`	100	10	1000	10	100	0.65	1.0	0.85	10	5	5	1.0	full
`SlateTopK-BoredInf`	100	10	1000	10	100	0.65	1.0	0.85	10	5	5	0.95	full
`SlateTopK-PartialObs`	100	10	1000	10	100	0.65	1.0	0.85	10	5	5	0.95	partial
`SlateTopK-Uncertain`	100	10	1000	10	{2, 5, 10}	0.65	1.0	0.85	10	5	5	0.95	partial
`SlateRerank-Static`	10	10	10	10	5	0.30	1.0	0.85	N/A	N/A	N/A	1.0	full
`SlateRerank-Bored`	10	10	10	10	5	0.30	1.0	0.85	10	5	5	1.0	full

It relies on a soft actor-critic (SAC) [14] policy that takes actions in the item embedding space. The recommended slate is then formed by identifying the items which maximize the dot-product with the action, i.e., the K-nearest neighbors, and by ordering them in a top-down fashion. For the `SingleItem` environments, we adopt a standard SAC agent and simply replace the top-K selection by a top-1 selection.

- **SAC + GeMS:** Proposed in [11], this approach relies on a variational autoencoder (VAE) to embed the high-dimensional slate space into a low-dimensional latent space, that is used as a tractable action space for a SAC agent. This process is done in two steps. First, a VAE is trained on logged data containing past user sessions with slates and clicks. For that purpose, we generate a dataset which collects interactions between the environment of interest and a logging policy corresponding to a uniformly balanced mixture of a Random agent and a Greedy Oracle agent.[8] Second, the frozen decoder of the VAE is plugged on the output of a SAC agent to reconstruct a slate from the agent's action in the latent space.
- **HAC:** Similarly to the GeMS framework, the hyper-actor critic (HAC) method [30] proposes to use an RL agent which takes actions in a latent space and introduces a module to translate latent actions into slates. Differently from SAC + GeMS, this approach relies on the DDPG [29] policy and it does not exploit a VAE to regularize the latent space. It also requires no pretraining as all parameters are learned in an off-policy fashion. Moreover, HAC uses a supervised click prediction objective in addition to the RL one, in order to stabilize the learning of the agent and directly exploit the user response signal on slate items.

In our experiments, we consider that methods have access to the ideal item embeddings, i.e., the item embeddings that are used in the simulator (whose generation was described in Section 3.1). This constitutes an advantage for the agents which explicitly use item embeddings in their method, namely, Greedy Oracle, SAC + Top-K, SAC + GeMS, and HAC. The other approaches (Random and REINFORCE + Top-K) therefore have a slight disadvantage over the former methods for that reason. To study the impact of the access to high-quality item embeddings, we also compared the results with ideal embeddings to those obtained using sub-optimal, matrix factorization embeddings (see the experiments on `SlateTopK-Bored` in Section 5.2).

4.3 Hyperparameter setting

The hyperparameters used for each of the environments introduced in Section 4.1 are detailed in Table 3. The hyperparameter values were chosen to reflect the environment-specific characteristics that we highlighted in

[8] In other words, each item in a slate generated by the logging policy has 50% chance to be the item the Greedy Oracle would recommend at this rank, and 50% chance to be a random item.

Table 2.

We now describe the hyperparameters used for the different methods.[9] For all RL recommendation agents, we set the discount factor γ to 0.0 for static environments (without boredom and influence) and 0.8 for dynamic ones (with boredom and/or influence). Agents are trained for 500,000 steps, where each step corresponds to the agent producing a recommendation – a slate or a single item depending on the environment. The policy learning rate and critic learning rate (for approaches with a critic) were fixed to 0.0003 and 0.01, respectively. Actors and Q-networks are MLPs with a hidden size of 256 at all layers. For REINFORCE agents, the buffer size was set to 100. For SAC-based approaches and HAC, we used a buffer size of 10^6, a batch size of 32, and a target smoothing coefficient τ equal to respectively 0.05 and 0.5. In SAC-based agents, we adopted auto-tuning for the entropy regularization coefficient α and we used a single Q-network. We also independently tuned the hyperparameters specific to the HAC approach: we set the learning rate of the behavior loss to 0.00003, the standard deviation for the reparameterization trick to 0.1, the weight for the hyper-actor loss to 0.1, and the dimension of the latent space to 32.

For environments with partial observability (`SingleItem-PartialObs`, `SlateTopK-PartialObs`, `SlateTopK-Uncertain`), we used two types of state encoders commonly used in RL-based recommender systems [17]: GRU and transformer. The input to the state encoder is a sequence containing for each step the concatenation of click embeddings and item embeddings averaged over the slate. The click and item embeddings are learned independently in the state encoder. The click embedding dimension was set to 2, and the item embedding dimension was set to 16 for the GRU state encoder and 32 for the transformer state encoder. The dimension of the state output by the state encoder was fixed to 32. We used 2 layers (both for the GRU and the transformer), as well as 4 attention heads, a dropout rate of 0.1, and a feedforward dimension of 64 (only for the transformer).

4.4 Evaluation protocol and metrics

For the evaluation, we performed 5 seeded runs for each method on each environment. For each run, we recorded the validation performance on 25 validation episodes every 50,000 training steps. An episode corresponds to a session of L steps where each step corresponds to the agent issuing a recommendation. For every episode, we sample a new random user embedding following the procedure described in Section 3.1. For that reason, validation users are distinct from training users, ensuring no leakage between training and validation. We also used different seeds during hyperparameter tuning (detailed in Section 4.3) and evaluation, in order to have different validation users in these two phases and thus avoid the situation where methods would be specifically optimized on the set of users sampled for tuning.

We considered two metrics in our evaluation. The first one is the return (i.e., the cumulated reward over an episode), averaged over the 25 validation episodes. This metric ranges from 0 to $L \times S$ (i.e., 100 for `SingleItem` environments and 1000 for `SlateTopK` environments), which corresponds to the case where the user clicked on all the items presented to them. A higher return indicates more clicks from the user across the episode and thus higher quality recommendation from the agent. On the environments that include a boredom mechanism (`SingleItem-BoredInf`, `SlateTopK-Bored`, `SlateTopK-BoredInf`, `SlateTopK-PartialObs`, `SlateTopK-Uncertain`), we also report the boredom metric. We define this metric as the number of steps in the episode where the user is bored on at least one topic – lower is better. In our churn-an-return boredom setting (see Section 3.4 for more details), this corresponds to the number of steps where the user embedding is zeroed out and the user cannot click. This metric is important as in order to be successful an agent should be able to balance accurate recommendations (to reach high immediate rewards) and diverse recommendations over time (to avoid triggering boredom and temporary churn).

[9]With our source code we provide the detailed hyperparameters used for each agent on each environment to facilitate reproducibility.

5 RESULTS

In this section, we describe the results of the experiments done on single item recommendation (Section 5.1), slate top-K recommendation (Section 5.2), and slate reranking (Section 5.3). Again, we remind the reader that the goal of these experiments is to demonstrate the possibilities and challenges of the environments derived from SARDINE, rather than to create a benchmark for state-of-the-art approaches on a limited set of environments. These experiments should be seen as a starting point for researchers and practitioners to further investigate the specific scenarios and approaches of their interest.

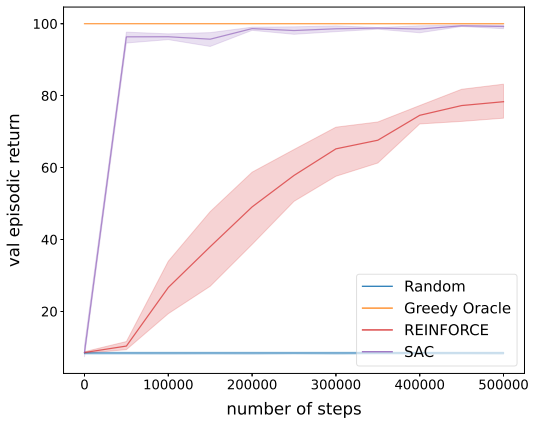
5.1 Experiments on single item recommendation

We performed experiments on three `SingleItem` environments whose characteristics are recalled below (see Section 4.1 for a more detailed description). In `SingleItem-Static`, we consider an easy, static recommendation scenario in a fully observable setting and with low uncertainty in order to validate that learned agents can reach optimal performance. `SingleItem-BoredInf` adds boredom and influence mechanisms to `SingleItem-Static`, increasing the difficulty of the environment. For `SingleItem-PartialObs`, we start as well from `SingleItem-Static` but change it to a POMDP setting. The results on the `SingleItem` environments are given in Fig. 2 and discussed below.

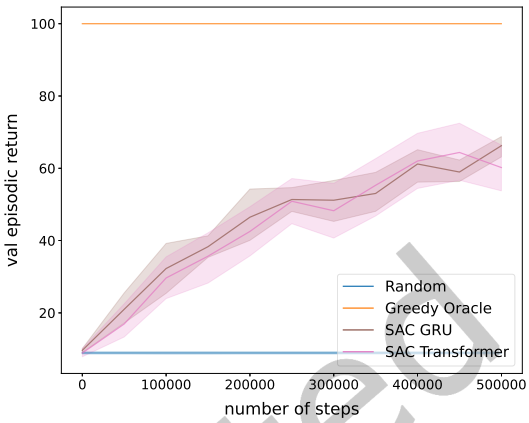
`SingleItem-Static`. The validation return over the different training steps on `SingleItem-Static` is plotted in Fig. 2a. In this experiment we compared SAC and REINFORCE agents against the Greedy Oracle and Random baselines. In this specific setting, the Greedy Oracle is by design optimal due to the absence of long-term mechanisms (boredom or influence). It is therefore not surprising that the Greedy Oracle achieves a return of 100, meaning that all recommended items in the session have been clicked. However, it is interesting to note that among the learned agents, SAC fares much better than REINFORCE. Indeed, the former is able to reach optimal performance (or very close to it) after only 200,000 steps, whereas the latter struggles to close the gap. This difference might be explained by the fact that SAC exploits the ideal item embeddings whereas REINFORCE simply selects actions through a softmax over items. Additionally, SAC is generally a better performing RL agent than REINFORCE in most cases, due to a better bias-variance trade-off. Nonetheless, it is reassuring to see that in this simple environment, a learned agent such as SAC is able to easily find the optimal policy.

`SingleItem-BoredInf`. We now turn to the more challenging `SingleItem-BoredInf` environment which includes boredom and influence mechanisms. The results according to the return and boredom metrics are illustrated in Fig. 2c and 2d, respectively. This environment corresponds to typical RL-based recommendation in an MDP setting and we expect RL agents to be able to beat a myopic approach such as the Greedy Oracle. Indeed, the Greedy Oracle is no longer optimal here due to the introduction of long-term mechanisms. This is confirmed by the results in the plots, which show that the Greedy Oracle only yields a return of around 50, and a boredom of around 50 as well. This means that for 50% of the steps the user is in a bored state, and for the remaining 50% the recommendation leads to a click. Turning to the RL agents, we first see that REINFORCE struggles to learn an effective policy and remains inferior to the Greedy Oracle in terms of return. However, SAC is able to provide high-quality recommendations, with a return close to 70 even after only 50,000 training steps. SAC is also able to recommend diversified items, as its low (and diminishing) boredom confirms. On the other hand, the boredom of REINFORCE increases steadily as return increases, showing that its policy is still in an accuracy improvement stage and is not favoring result diversification.

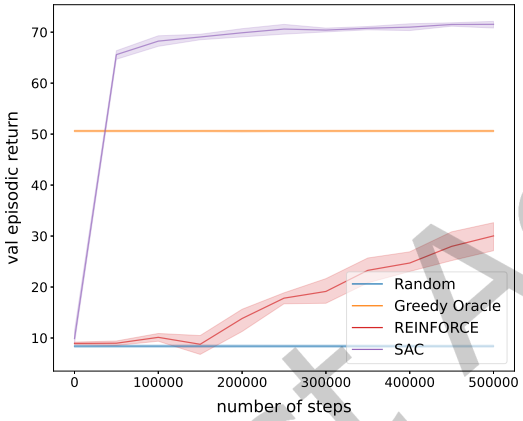
`SingleItem-PartialObs`. This environment is static like `SingleItem-Static` but we consider here a POMDP setting, i.e., partial state observability. This corresponds to the typical sequential recommendation scenario based on offline feedback, where recommendations have no causal effect on user behavior [10]. Due to the partial observability, RL agents require a state encoder to convert the observations returned by the environment into a state that can be exploited by the agent. We consider both a GRU and a transformer state encoder, which we



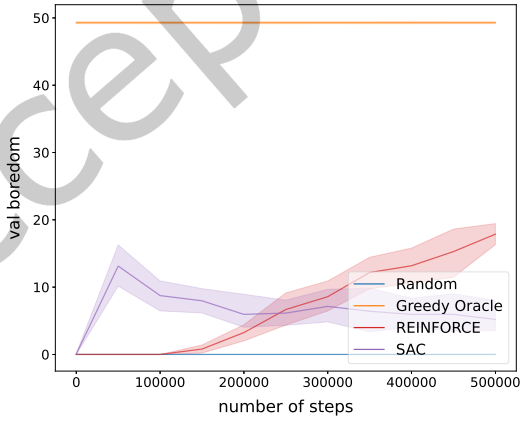
(a) Return (↑) on `SingleItem-Static`



(b) Return (↑) on `SingleItem-PartialObs`



(c) Return (↑) on `SingleItem-BoredInf`



(d) Boredom (↓) on `SingleItem-BoredInf`

Fig. 2. Results on the `SingleItem-Static` (2a), `SingleItem-PartialObs` (2b), and `SingleItem-BoredInf` (2c, 2d) environments. The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs. Boredom results are not shown for `SingleItem-Static` and `SingleItem-PartialObs` as these static environments do not include a boredom component and thus all methods have a default boredom of 0.

test in combination with a SAC agent as it obtained convincing results on `SingleItem-Static`. The results are shown in Fig. 2b. Here, the Greedy Oracle is still optimal as it has access to the true user state (and thus is not affected by the POMDP setting). The partial observability does have an impact on the SAC agents, leaving a gap between the 60+ return obtained by these and the 100 return obtained by the Greedy Oracle. This highlights that more research might be needed on state encoders to be able to accurately estimate the true user state in this setting. Comparing the variants of SAC equipped with a GRU and transformer state encoder, we do not notice statistically significant differences between those two in this case.

5.2 Experiments on slate top-K recommendation

We now move on to the experiments performed on slate top-K recommendation, i.e., where the recommendation presented to the user is a list instead of a single item as in `SingleItem` environments. We studied four `SlateTopK` environments which are summarized below (and further detailed in Section 4.1). `SlateTopK-Bored` and `SlateTopK-BoredInf` are both fully observable environments which require agents to do multi-step reasoning to perform well. The difference between the two is that the former only includes a boredom mechanism, whereas the latter additionally integrates an influence mechanism – causing user embeddings to drift based on clicked items and thus making it more difficult to track user interest. We also investigated a partially observable version of `SlateTopK-BoredInf` through `SlateTopK-PartialObs`, further increasing the difficulty of the task. Finally, we experimented with various levels of uncertainty in the clicking process through the `SlateTopK-Uncertain` environments. The results on the `SlateTopK` environments are shown in Fig. 3, 4, 5, and 6.

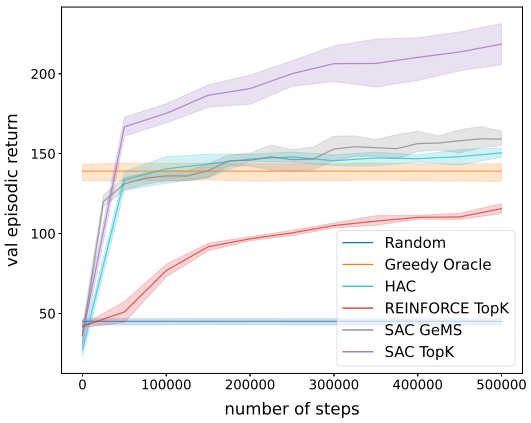
`SlateTopK-Bored`. The results on the `SlateTopK-Bored` environment are plotted in Fig. 3a (for the return) and Fig. 3b (for the boredom). We compared the Greedy Oracle baseline, which is sub-optimal here, to four learned agents: REINFORCE + Top-K, SAC + Top-K, SAC + GeMS and HAC. Similar to the results observed on `SingleItem` environments, we see here that REINFORCE + Top-K fails to reach the performance of the Greedy Oracle, while SAC + Top-K beats by a good margin the Greedy Oracle. HAC and SAC + GeMS are also able to beat the oracle baseline, but by a much smaller margin. In terms of boredom, SAC + Top-K is also the winner with a much lower value. We also report the distribution of item relevance scores for the different methods tested here in Appendix D. We hypothesize that the superiority of SAC + Top-K, in particular over SAC + GeMS and HAC, is due to the use of ideal item embeddings in this experiment. Indeed, SAC + Top-K directly uses the item embedding space as action space and thus rely heavily on the quality of item representations. To investigate this hypothesis, we repeated the same experiment as shown in Fig. 3a and Fig. 3b, but we replace the ideal item embeddings used by default in our experiments with item embeddings learned by matrix factorization (MF).[10] We report the results in Fig. 3c and Fig. 3d for the return and boredom metrics, respectively. We observe that changing from ideal to MF embeddings does have a drastic effect on the recommendation performance (measured by the return metric) of SAC + Top-K, which degraded to the level of the Greedy Oracle. The performance of HAC is also greatly impacted – its return dropping even below that of REINFORCE Top-K. SAC + GeMS is the approach that underwent the smallest performance drop in comparison to the ideal embedding case. This suggests that this latter approach might overall be more robust to sub-optimality in item embeddings.

In addition to the results described above on `SlateTopK-Bored`, we also investigated a challenging variant of this environment that is closer to a real-life scenario, where certain topics tend to co-occur and the distribution of topics for items and users is skewed. The setting and the experiments done in this environment are further detailed in Appendix B.

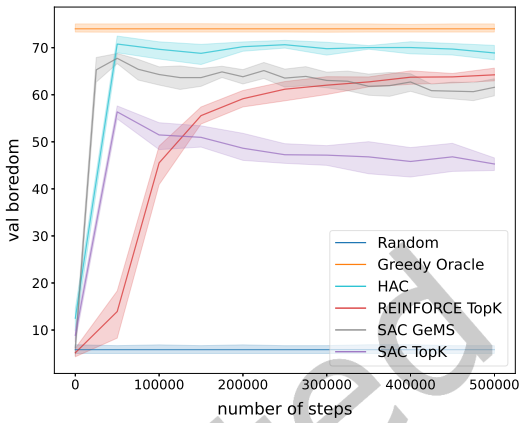
`SlateTopK-BoredInf`. On the `SlateTopK-BoredInf` environment, which adds an influence mechanism to `SlateTopK-Bored` with ideal embeddings, we observe similar trends as this latter environment. The return and boredom results are given in Fig. 4a and Fig. 4b, respectively. One notable difference with `SlateTopK-Bored`, however, is that in `SlateTopK-BoredInf` HAC fails to beat the Greedy Oracle and gets a return that is significantly worse than that of SAC + GeMS. This might be explained by the fact that HAC integrates a supervised click prediction loss which may hinder the model performance due to the greater dynamics in the user embedding caused by the influence drift.

`SlateTopK-PartialObs`. The results on the `SlateTopK-PartialObs` environment, which increases `SlateTopK-BoredInf`'s challenge with partial observability, are shown in Fig. 5a (for return) and Fig. 6a (for boredom). Given

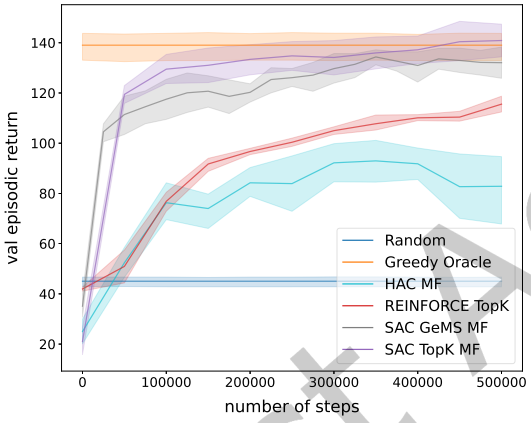
[10]To obtain these embeddings, we used the dataset generated for SAC + GeMS pretraining (described in Section 4.2) to train a matrix factorization model with an embedding dimension of 10.



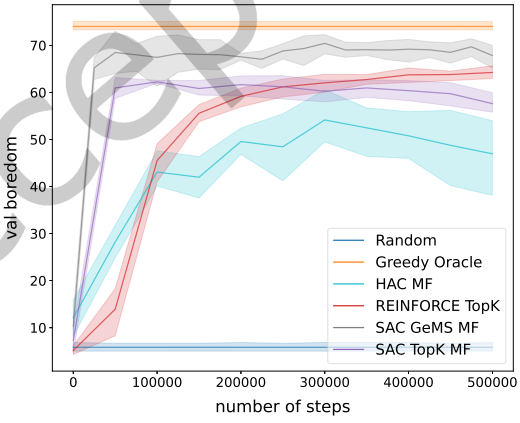
(a) Return (↑) on `SlateTopK-Bored` (Ideal)



(b) Boredom (↓) on `SlateTopK-Bored` (Ideal)



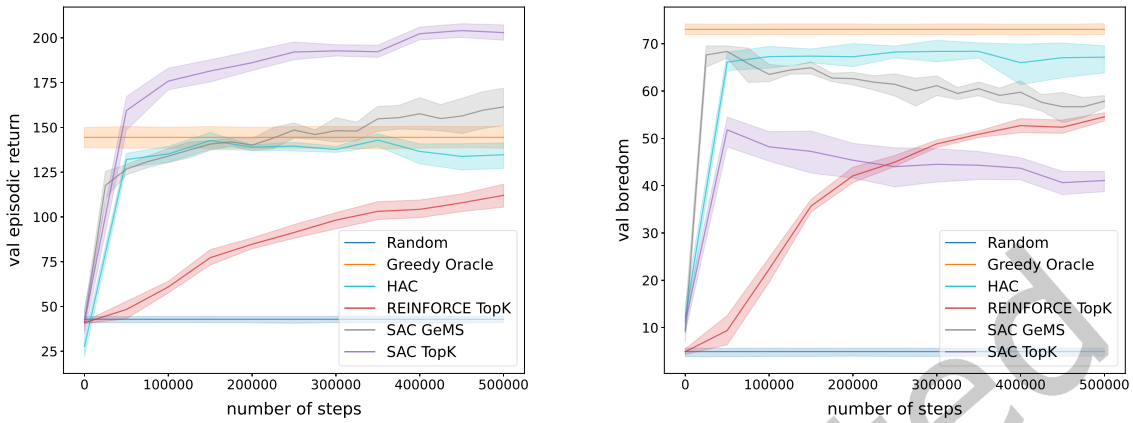
(c) Return (↑) on `SlateTopK-Bored` (MF)



(d) Boredom (↓) on `SlateTopK-Bored` (MF)

Fig. 3. Results on the `SlateTopK-Bored` environment with default, ideal item embeddings (3a, 3b) and with matrix factorization item embeddings (3c, 3d). The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs. Some approaches keep the same performance across the two settings as they either do not rely on item embeddings (Random, REINFORCE Top-K) or are an oracle baseline and only make sense with ideal item embeddings (Greedy Oracle).

the superior performance of SAC + Top-K on `SlateTopK-BoredInf`, we focus here on variants of this method based on a GRU or a transformer state encoder. In this setting, we observe that the performance of the transformer variant leads on most training steps to a significant improvement in terms of return over the GRU variant. This result goes in line with previous findings on state encoders for RL-based recommendation [17]. However, both SAC + Top-K variants fail to beat the Greedy Oracle baseline, highlighting the difficulty of this environment and showing that additional efforts on the agent and/or state encoder might be needed to achieve high-quality recommendation.



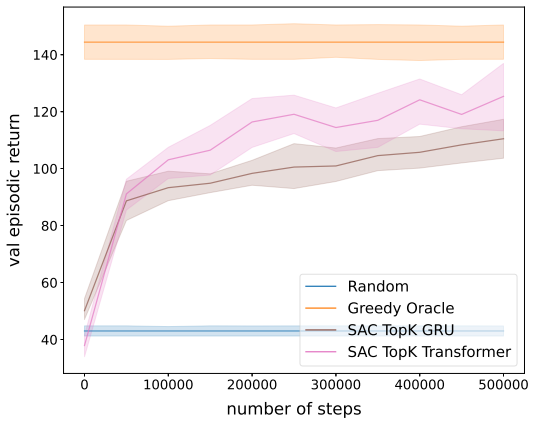
(a) Return (↑) (b) Boredom (↓)

Fig. 4. Results on the `SlateTopK-BoredInf` environment. The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

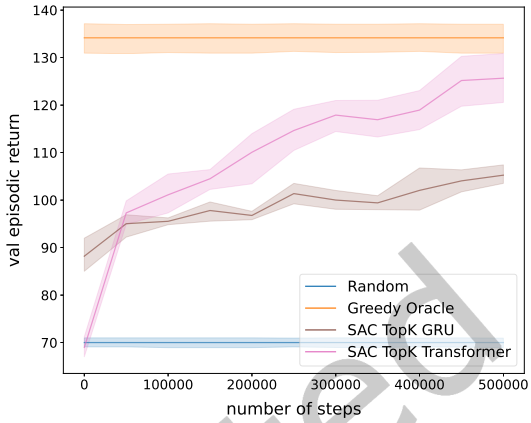
`SlateTopK-Uncertain`. Starting from `SlateTopK-PartialObs`, we varied the level of uncertainty in the clicks through the λ scale hyperparameter in the simulator's relevance function. In particular, we compared the setting of `SlateTopK-PartialObs` with low uncertainty ($\lambda = 100$) to different `SlateTopK-Uncertain` environments with medium uncertainty ($\lambda = 10$), high uncertainty ($\lambda = 5$) and very high uncertainty ($\lambda = 2$) – which we will refer to as `SlateTopK-Uncertain10`, `SlateTopK-Uncertain5`, and `SlateTopK-Uncertain2` for simplicity. The return and boredom results in these environments are illustrated in Fig. 5 and Fig. 6, respectively. Comparing the return on `SlateTopK-PartialObs` (Fig. 5a) to `SlateTopK-Uncertain10` (Fig. 5b), we observe that the gap between the SAC + Top-K transformer and GRU variants increases. Indeed, while the overall performance of SAC + Top-K GRU slightly decreases with the uncertainty increase, we see that SAC + Top-K transformer is able to maintain its performance at around 125 at 500,000 steps. This suggests that the transformer state encoder is more robust to a medium level of uncertainty. When we increase the uncertainty to a high level in `SlateTopK-Uncertain5` (Fig. 5c), we notice that the SAC Top-K variants beat the Greedy Oracle baseline, and that the gap between the Random baseline and the Greedy oracle shrinks. This is explained by the fact that with more stochasticity in the clicking process, less relevant items get more clicks – which reduces the advantage of the greedily optimal recommendations from the Greedy Oracle. Clicks on more varied items also means that user boredom is less likely to be triggered, which is confirmed by the comparison of the boredom scores of the SAC Top-K variants across Fig. 6a, Fig. 6b, and Fig. 6c. When the uncertainty level is further increased in `SlateTopK-Uncertain2`, we observe that the environment rewards random recommendations more than the accurate recommendations from the Greedy Oracle, as shown in Fig. 5d. This is, again, explained by the fact that less relevant items lead to a click probability similar to that of relevant items. In this setting, the SAC Top-K variants both perform similarly to the Random baseline and are thus learning to favor more diverse recommendations over accurate ones.

5.3 Experiments on slate reranking

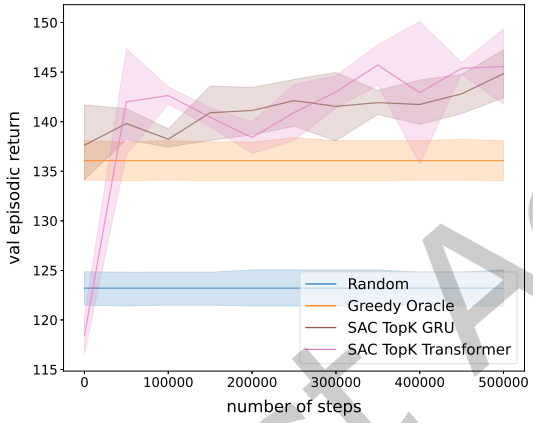
With this last set of experiments, we explore the reranking task using two `SlateRerank` environments: the static `SlateRerank-Static` and the interactive, multi-step `SlateRerank-Bored`. We conduct experiments on click modeling according to the following protocol: (i) we generate a dataset of interactions using a certain logging policy, (ii) we train a click model on the generated dataset, and (iii) we rerank the items by decreasing amount



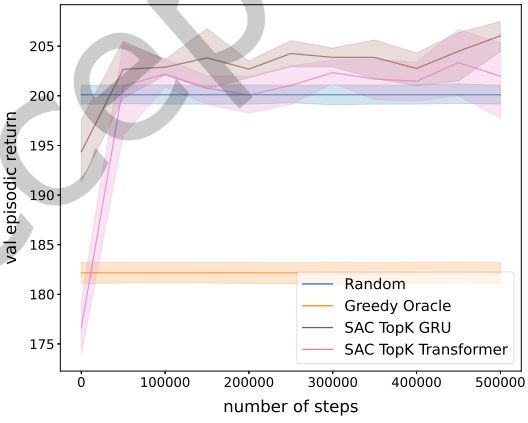
(a) `SlateTopK-PartialObs`, low uncertainty



(b) `SlateTopK-Uncertain`, medium uncertainty



(c) `SlateTopK-Uncertain`, high uncertainty

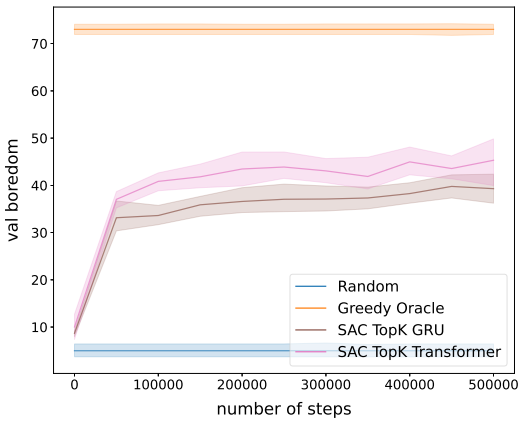


(d) `SlateTopK-Uncertain`, very high uncertainty

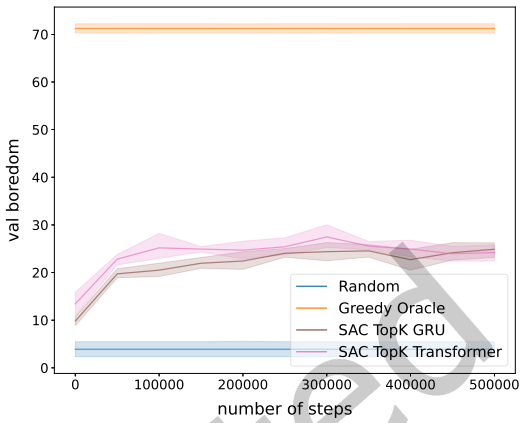
Fig. 5. Results in terms of return (↑) on the `SlateTopK-PartialObs` (5a) and `SlateTopK-Uncertain` (5b, 5c, 5d) environments. The click uncertainty degree varies from low (5a), medium (5b), high (5c) to very high (5d), corresponding to a scale hyperparameter λ in the relevance function equal to 100, 10, 5, and 2, respectively (see Section 3.3 for more details). The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

of relevance, according to the model. For both environments, we use the reverse-oracle policy, i.e., the policy that orders items by *increasing* order of relevance, as the logging policy. It therefore generates a dataset that contains substantial spurious correlations due to position bias. We report the observed return when applying the reranking methods in the live environment in Table 4.

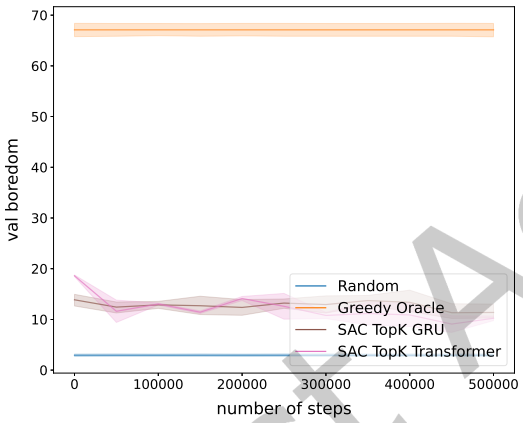
`SlateRerank-Static`. On the static environment, the Greedy Oracle policy is the optimal policy, while the Reverse Oracle yields minimal return. We can indeed first verify in Table 4 that an online-trained SAC + Top-K, as in Section 5.2, even with full observability of user state and ideal item embeddings, does not beat the greedy oracle.



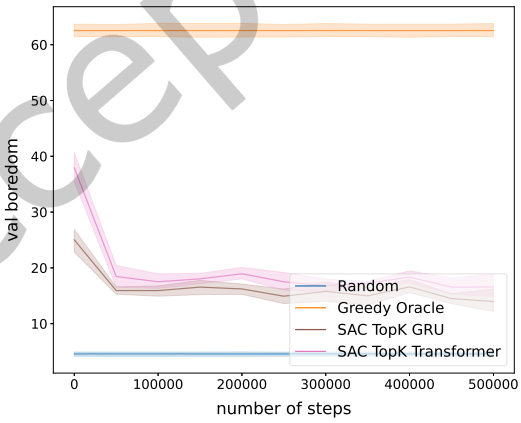
(a) `SlateTopK-PartialObs`, low uncertainty



(b) `SlateTopK-Uncertain`, medium uncertainty



(c) `SlateTopK-Uncertain`, high uncertainty



(d) `SlateTopK-Uncertain`, very high uncertainty

Fig. 6. Results in terms of boredom (↓) on the `SlateTopK-PartialObs` (6a) and `SlateTopK-Uncertain` (6b, 6c, 6d) environments. The click uncertainty degree varies from low (6a), medium (6b), high (6c) to very high (6d), corresponding to a scale hyperparameter λ in the relevance function equal to 100, 10, 5, and 2, respectively (see Sections 3.3 for more details). The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

Secondly, we can see that a position-based model (PBM) correctly identifies the biases in the logged data and almost reaches the performance of the oracle policy, while the naive document click-through rate model (dCTR) model fails to do so, and barely improves on the Reverse Oracle policy it was trained on. This result does not come as a surprise since the underlying user click model in the `SlateRerank` environments is also a position-based model. We can nonetheless verify that the learned propensities, i.e., observation probabilities at each rank, match the true propensities of the simulator. We therefore compute the mean-squared error (MSE) of the normalized propensities, i.e., where the probability of observation at the first position is set to 1, and we find that the learned PBM's propensities have an MSE of 0.570. That indicates that despite documents being correctly ordered, the

Table 4. Online return obtained by debiasing the logged data on the reranking environments. Averaged over five seeded runs.

Method	`SlateRerank-Static`	`SlateRerank-Bored`
Greedy Oracle	21.45	13.69
Reverse Oracle	8.82	8.47
dCTR	9.28	8.97
PBM	21.17	13.14
Online SAC + Top-K	19.01	14.82

learned model does not fully match the underlying model.

The experiments on `SlateRerank-Static` call for further experiments with different underlying user click models and candidate click models, e.g. as done in [9], so as to investigate the performance of click models under the more realistic settings of model mismatch. While we leave this for readers to experiment with, we turn to another natural extension that is, to the best of our knowledge, unexplored, and that SARDINE enables.

`SlateRerank-Bored`. In this interactive environment, the Greedy Oracle policy is not optimal anymore, because the agent must trade off the accuracy and diversity of the most-exposed topics. Indeed, we can see in Table 4 that an online-trained SAC + Top-K agent beats the Greedy Oracle. This environment therefore constitutes a testbed for (offline) RL agents with biased feedback, and notably the combination of RL and click modeling.

Another important difference that comes with this interactive environment is the fact that the logged data may appear relatively noisier to a click model, as the click/skip feedback can be explained by something else than relevance and position: the boredom. While the boredom information is contained in the ideal user state we use for click model training and the model should therefore in theory be able to correctly identify biases, we expect the training process to be harder. We observe what seems to be a slight degradation of relative performance, compared to the static environment. Indeed, while the PBM managed to fill 98% of the gap between the logging policy and the oracle policy on the `Static` environment, it only fills 89% of the gap on the `Bored` environment. But the extent of the degradation is most apparent when we compare the propensities learned by the model in this new dynamic environment. The MSE now increases to 0.915, compared to 0.570 in the static case. This suggests that using the learned propensities of the model in downstream tasks, e.g., counterfactual learning-to-rank, fairness or reinforcement learning, is likely to lead to imperfect and biased policies.

Effectively using the user behavior learned by click models in a dynamic and interactive environment with, e.g., reinforcement learning, including when the learned variables are imperfect, is to the best of our knowledge still an open question. Our proposed simulator offers the possibility to study this topic in an interpretable and controllable way.

6 RELATED WORK

In this section we highlight how our work differs from previously published simulators. Considering the research agenda we defined in Section 1.2 as well as our target specifications 1.1, we draw a comparison of existing simulators, along with our proposed SARDINE, in Table 5. Note that some of these simulators may have been proposed to target a different research outcome, but we analyze only what we think to be relevant to interactive recommendation research and our corresponding research agenda. Also, we acknowledge that some of the criteria used here are subjective and we try to substantiate our claims as much as possible.

We now describe related simulators that have been published in recent years, and how they may differ from our objective.

RecoGym [39] is an e-commerce and advertising simulator where the agent aims to display attractive ads so that the users come back on an e-commerce website they have previously visited. It comes bundled with multiple bandit agents and use cases, including the effect of selection bias on offline agents, and stochasticity in user

Table 5. Comparison of the proposed SARDINE to existing recommendation simulators. ✓ indicates that the research topic is addressed by the simulator and ∼ that it is partially addressed. Our assessment of whether the simulators fulfill the specifications is graded according to { −, ±, + }. Overall, we find that only RecSim [21] addresses the research agenda that SARDINE targets, but that it does not fulfill our specifications for such a simulator.

	Research agenda 1.2				**Specifications 1.1**		
Simulator	RT1	RT2	RT3	RT4	Interpretability	Effect isolation	Configurability
RecoGym		∼	✓		±	+	+
MARS-Gym					−	−	±
RL4RS	∼	∼		✓	−	−	−
RecSim	✓	✓	∼	∼	±	−	±
Virtual-TB	✓		✓		−	−	−
SOFA		✓			+	±	+
OBP		✓			+	+	+
SARDINE	✓	✓	✓	✓	+	+	+

response and evolution, as well as in observed returning time. Its ease of configurability and experimentation makes it a desirable choice, but it does not address multi-step reasoning, slate recommendation and presentation bias.

MARS-Gym [43] aims to simulate online marketplaces, and is based on real data from such platforms. It includes next-item prediction and off-policy metrics for evaluation of trained agents. The agent's objective, for a given user, is to select one of the items that were observed in the real data for that user. Therefore, MARS-Gym aims to evaluate the quality of static, semantic information learned by agents and does not meet our research agenda targeting dynamic and interactive systems.

RL4RS [51] is an e-commerce, slate recommendation simulator based on real purchase data, and where the reward function is a black-box sequential recommendation model. It is composed of two variants: one-shot (i.e., single-turn) and sequential slate recommendation. Offline RL agents can be trained on the real logged data and evaluated in the simulator, but one cannot directly control the logging policy, and presentation bias is not modeled. The authors verify that a transformer model can better capture the item sequence using non-greedy decoding strategies, which might indicate multi-step dependencies. However, the simulator is opaque and hardly tunable, and thus does not satisfy our specifications for a research-oriented simulator.

RecSim [21] is certainly the effort closest to ours. The authors provided a configurable simulator and three environment instantiations that cover, at least partially, all research topics that we wish to study. However, we found practical drawbacks in using it, motivating us to propose our take on interactive recommendation simulators. First, installation and usage is made very difficult as it relies on older, unmaintained packages, without specifying the version being used. Moreover, relying on third-party software like Tensorflow 1.15 or Google dopamine hurts the ease of configurability of both the environment and agents. In contrast, our simulator relies only on Numpy (and Gymnasium). Second, we found tweaking the environment properties and singling out specific research questions to be hard, as there are often multiple parameters controlling the same research dimension, without clear guidance on their effect, and they are not always tunable without substantial modifications: e.g., the uncertainty in user response comes from a binary coin flip, which does not allow to draw profiles of robustness to increasing uncertainty. There is also no simple way to use an oracle for user state or item embeddings as we do in our environment to single out certain modules of the agent. Finally, while the simulator aims to tackle slate recommendation, no proposed environment uses a number of candidates greater than 15 or a slate size greater than 3, while we wish to study slate recommendation at a larger scale, e.g., with a number of candidates of 1000 and a slate size of 10 as in our proposed `SlateTopK` environments. Overall, we adopt the general philosophy of

RecSim and propose our take on making a lightweight, flexible, and research-oriented simulator.
Virtual-TaoBao [46] is an online retail simulator trained from real data, where generative adversarial networks are trained via multi-agent imitation learning in order to approximate the user response to recommendations. It incorporates certain uncertainties, e.g., on the user churning mechanism, and rewards multi-step reasoning, but it does not address other research topics, i.e., biases in the data and slate recommendation. Additionally, since the simulator consists of model approximations of real user behavior, the notion of items is lost (state and actions are continuous latent variables) and the user response is a black-box that cannot be tweaked for further experimentations.
SOFA [18] uses an intermediate reweighting step in order to remove popularity and positivity biases in the resulting simulator. The authors verify that policies trained in the debiased simulators perform better when evaluated on datasets from the same platform but where biases have been alleviated (i.e., through random recommendations). The simulator is relatively easy to tweak as we can replace the intermediate inverse-propensity scoring step with a different technique, and change the underlying logged data. However, SOFA does not target the study of the other research topics in our agenda, i.e., multi-step reasoning, environment uncertainty and slate recommendation.
OBP [41] is a semi-synthetic, research-oriented simulator for off-policy training evaluation of bandit agents. Using real logs of an online retail platforms collecting with several policies, it can evaluate the quality of off-policy evaluation estimators and therefore help research in that direction. However, it does not address our other concerns, i.e., multi-step reasoning, environment uncertainty and slate recommendation.

7 CONCLUSION

Summary. In this paper, we have introduced SARDINE, a simulator for automated recommendation in a dynamic and interactive environments. Our efforts seek to address different shortcomings identified in existing recommendation simulators: (i) a lack of comprehensiveness in the covered research questions, that compels researchers and practioners to scatter their study across several simulators; (ii) a lack of interpretability and controllability, when specific aspects of the simulator depend on the setting of multiple parameters; (iii) the inability to study in isolation the phenomena and effects of interest in the simulator; (iv) the solvability of the simulator through trivial off-the-shelf baselines; and (v) the difficulty for researchers and practitioners to make additions and changes to the simulator to study certain directions in more depth, or to investigate new research questions.

In an effort to cover a wide range of problems studied in recommendation, we devised our simulator to enable the investigation of four over-arching research topics, including the multi-step reasoning capacity of models (RT1), the ability to learn models from biased data (RT2), the robustness to uncertainty (RT3), and the challenges associated with recommending slates (RT4). Concretely, these research topics translate into six dimensions – that the practioner may or may not decide to include in their instantiation of the simulator – spanning the recommendation type (single-item vs. slate recommendation), the inclusion of a boredom and/or influence mechanism, the level of uncertainty in the clicking process, the state observability (full vs. partial), and whether the task is reranking.

We then conducted extensive experiments on a set of nine environments derived from SARDINE. These environments have been selected to constitute diverse combinations of the aforementioned dimensions and thus provide a good coverage of our four research topics. In our experiments, we compared various methods which include both simple baselines and state-of-the-art approaches. To foster reproducibility and enable researchers to draw from this work to develop their own environments, we release the code for the simulator,[11] as well as the

[11] https://github.com/naver/sardine

detailed specifications for each environment and the implementation for all the compared methods.[12]

Findings. Through our experiments on nine SARDINE environments, we derived some valuable insights on the behavior of existing approaches in certain settings, demonstrating the usefulness of our simulator. First, we found that the SAC + Top-K approach, which combines the widely used SAC agent to a simple top-K ranker, showed impressive performance across the different environments and demonstrated a high stability. To the best of our knowledge, this approach is rarely considered as a baseline in RL-based slate recommendation works (except in [11]) despite its effectiveness and relative simplicity in comparison to state-of-the-art models. Therefore, we advocate for its usage as a baseline in future work on slate recommendation in dynamic environments.

To slightly nuance this first finding, we wish to add as a caveat that SAC + Top-K may be particularly dependent on the high quality of the item embeddings used. The performance of this approach was particularly high when using the ideal item embeddings (i.e., the ones that are used inside the simulator), but it decreased by a good margin when we replaced the ideal item embeddings with sub-optimal, matrix factorization embeddings. In comparison, the SAC + GeMS [11] approach seemed to be more robust overall to the item embedding quality. The recent hyper-actor critic (HAC) approach [30] was the most impacted by the quality of item embeddings in the studied settings. Moreover, we found that this approach was more affected than other methods by a highly dynamic environment with a drift in the user interests. We attribute this to the supervised click prediction loss used in HAC, which favors immediate reward over multi-step reasoning in the model.

Secondly, we studied how a transformer state encoder compare to a GRU state encoder in partially observable environments, and identified that the former tends to outperform the latter. This was notable in particular on environments where the click uncertainty was medium or high. This finding on the superiority of the transformer over the GRU as a state encoder goes in line with previous studies [17] and thus it does not come as a surprise. However, the impact of the level of click uncertainty on the state encoder is a subject that has not been considered a lot in the recommendation literature, and might be a topic worth investigating more deeply.

Finally, we conducted experiments on the impact of presentation bias in the user feedback in a recommendation scenario. We notably found that when the environment is dynamic, click models trained offline may be less accurate than on static environments, which can have a detrimental effect on downstream tasks, such as counterfactual learning-to-rank or offline reinforcement learning. Our experiments also open up the possibility of studying the end-to-end training of RL agents from biased data, including a click modeling step.

Overall, the experiments we conducted act as guidance on how to use the simulator, examples of use cases that can be studied through SARDINE, and more importantly as a call for more research on dynamic and interactive approaches for recommender systems. The insights we gathered throughout our experiments also reinforce the usefulness of simulated evaluation in general, and SARDINE in particular.

Future work. While we designed our simulator to be flexible and configurable, so that researchers can tweak the experimental setup to their needs, we did not implement variants of the simulator that target the study of many of the research questions associated with our agenda (Section 1.2). For instance, the performance of agents when the environment is non-stationary (e.g., due to changes in the world) is still largely unknown [38], and could be investigated in SARDINE. Similarly, reaching the best possible policy in a limited number of deployments, a task known as deployment-efficiency [32], as well as continual learning [52], which aims to deploy agents that keep on learning, could be explored in the recommendation scenario thanks to SARDINE. We hope our simulator can foster the experimentation of such novel ideas in recommender systems research.

ACKNOWLEDGMENTS

This research was (partially) funded by the Hybrid Intelligence Center, a 10-year program funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research,

[12] https://github.com/RomDeffayet/SARDINE_Experiments

Table 6. Value of the simulator hyperparameters for the `WebtoonSlateTopK-Bored` environment. The description of the hyperparameters' meaning and role is detailed in Table 1.

Environment name	Hyperparameter value												
	L	S	$n_{\mathcal{I}}$	$n_{\mathcal{T}}$	λ	μ	α	ϵ	n_{b}	t_{b}	τ_{b}	ω	$\mathcal{O}$
`WebtoonSlateTopK-Bored`	100	10	669	16	100	0.65	1.0	0.85	10	5	5	1.0	full

https://hybrid-intelligence-centre.nl, project LESSEN with project number NWA.1389.20.183 of the research program NWA ORC 2020/21, which is (partly) financed by the Dutch Research Council (NWO), and the FINDHR (Fairness and Intersectional Non-Discrimination in Human Recommendation) project that received funding from the European Union's Horizon Europe research and innovation program under grant agreement No 101070212.

All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

A EFFICIENCY

On a single Intel Xeon Gold 6338 CPU, we found that our simulator can operate at approximately 4,500 steps (i.e., user interactions) per second with the `SlateTopK-Bored` environment and up to 5,000 steps per second on the `SingleItem-Static` environment. Moreover, training a SAC+TopK agent on `SlateTopK-Bored` or `SlateTopK-BoredInf` for 500,000 training steps, as in Section 5.2, takes around 40 minutes on a single NVIDIA A100 GPU.

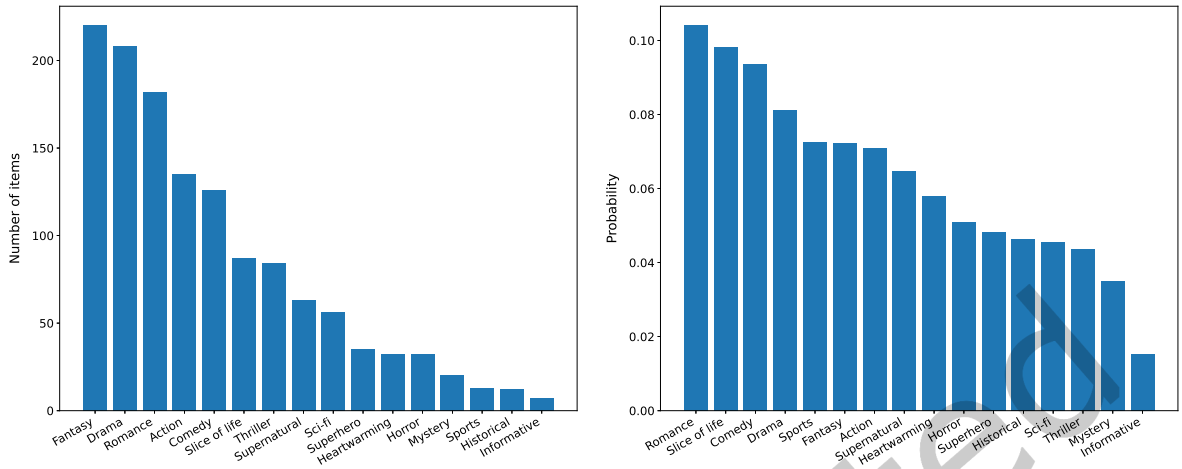
B WEBTOON EXPERIMENT

B.1 Environment description

The environments introduced in Section 4 and experimented on in Section 5 are based on purely synthetic items with uniformly drawn topic-item assignments. While these environments enabled us to derive interesting insights, one might question whether such environments reflect a real-life scenario where (i) topic-item assignments are not uniformly drawn (i.e., some topics tend to co-occur within items), (ii) item-topic distribution is skewed (i.e., some topics are more prominent than others among items), and (iii) user-topic distribution is skewed (i.e., some topics are more popular than others among users). To showcase the possibilities of SARDINE to address such a scenario, we define a semi-synthetic environment named `WebtoonSlateTopK-Bored` that presents the same characteristics as `SlateTopK-Bored` with the difference that its items are based on the real-world catalog of the Webtoon[13] online comics platform. The hyperparameters for this environment are summarized in Table 6. Differently from `SlateTopK-Bored`, `WebtoonSlateTopK-Bored` includes 669 items and 16 topics.

User and item embeddings. In this experiment, we consider that we only have access to an item catalog, which does not directly include embeddings. Therefore, item and user embeddings need to be generated under the constraints imposed by the catalog (e.g., item-topic assignments), leading to a semi-synthetic setting. To obtain item and user embeddings based on the Webtoon catalog, we slightly changed the generation process described in Section 3.1. For item embeddings, steps (2) and (3) are removed as item-topic assignments are directly obtained from the catalog – these correspond to the genres of an item, such as *Drama*, *Romance*, *Superhero*, *Sci-fi*, etc. Exploiting these assignments ensures a meaningful co-occurrence of topics within items: for example, the pairs (*Drama*, *Romance*) as well as (*Superhero*, *Sci-fi*) are more likely to co-occur than the pair (*Romance*, *Superhero*). The distribution of items across topics, i.e., the number of items attributed to each topic, is illustrated in Fig. 7a. This figure highlights a skewed distribution, with a large share of items pertaining to *Fantasy*, *Drama*, or *Romance*

[13] https://www.webtoons.com/en (item catalog accessed in January 2022).



(a) Item distribution across topics

(b) Topic prior for user embeddings

Fig. 7. Properties of Webtoon items and topics (i.e., Webtoon categories). Fig. 7a depicts the number of items attributed to each topic, based on the actual item-category assignments in the Webtoon catalog. Fig. 7a illustrates the topic prior used for generating user embeddings in SARDINE, which is drawn from the number of likes obtained for the items of each category.

topics, and a much smaller share of items with *Sports*, *Historical* or *Informative* topics.

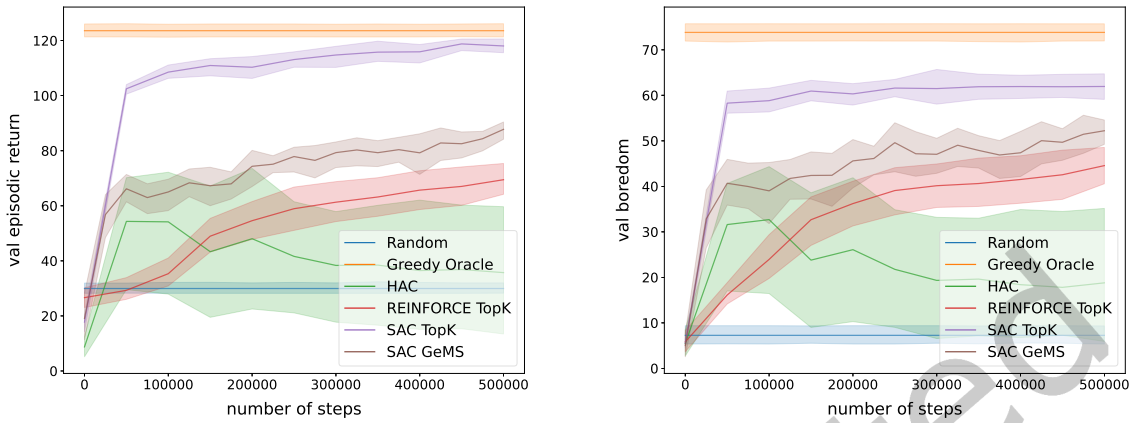
To generate user embeddings, we changed step (3) from the embedding generation process in Section 3.1 to reflect the fact that topics may not be uniformly popular among users. More specifically, instead of being sampled from Unif($\mathcal{T}$), the topics of interest for a user u (denoted as $\mathcal{T}_u = \{T_{u,1}, \ldots, T_{u,n_{\mathcal{T}_u}}\} \subset \mathcal{T}$) are sampled from a categorical, non-uniform prior $p_{\mathcal{T}}$ without replacement. The probability $p_{\mathcal{T}}(j)$ for a topic j is defined as the ratio of the average number of likes for items with category j divided by the average number of likes for items with any category. Formally, $p_{\mathcal{T}}(j)$ is defined as follows:

$$p_{\mathcal{T}}(j) = \frac{\frac{1}{|\mathcal{I}_j|} \sum_{i \in \mathcal{I}_j} \#\text{likes}(i)}{\sum_{j' \in \mathcal{T}} \frac{1}{|\mathcal{I}_{j'}|} \sum_{i \in \mathcal{I}_{j'}} \#\text{likes}(i)}, \tag{3}$$

where $\#\text{likes}(i)$ indicates the number of likes given to item i, and $\mathcal{I}_j$ is the set of items which pertain to topic j (i.e., items i such that $T_{i,j} > 0$). The distribution $p_{\mathcal{T}}$ is plotted in Fig. 7b, which highlights as well the skewed nature of user-topic preferences in the `WebtoonSlateTopK-Bored` environment.

B.2 Results

Following the protocol and hyperparameters used for the experiments on `SlateTopK-Bored` (Ideal) (the results of which are described in Section 5.2, and illustrated in Fig. 3a and Fig. 3b), we compared the same methods on the `WebtoonSlateTopK-Bored` environment. The results are shown in Fig. 8, with the return in Fig. 8a and the boredom in Fig. 8b averaged over validation episodes. A notable difference with the results on the `SlateTopK-Bored` environment is the fact that no RL-based approach is able to beat the Greedy Oracle. This might be explained by the greater difficulty of `WebtoonSlateTopK-Bored` due to its skewed item-topic and user-topic distributions. Nonetheless, similarly to the results on `SlateTopK-Bored`, we observe that among the RL-based approaches, SAC + Top-K performed the best and is followed by SAC + GeMS. However, differently from previous results, HAC underperformed and showed some instability which is illustrated by its high variance. This suggests that HAC might be less robust and more sensitive to changes in the environment in comparison



(a) Return (↑) (b) Boredom (↓)

Fig. 8. Results on the `WebtoonSlateTopK-Bored` environment. The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

to other methods. Overall, this experiment demonstrates that deriving environments with realistic, long-tail distributions provides interesting and challenging use-cases in SARDINE.

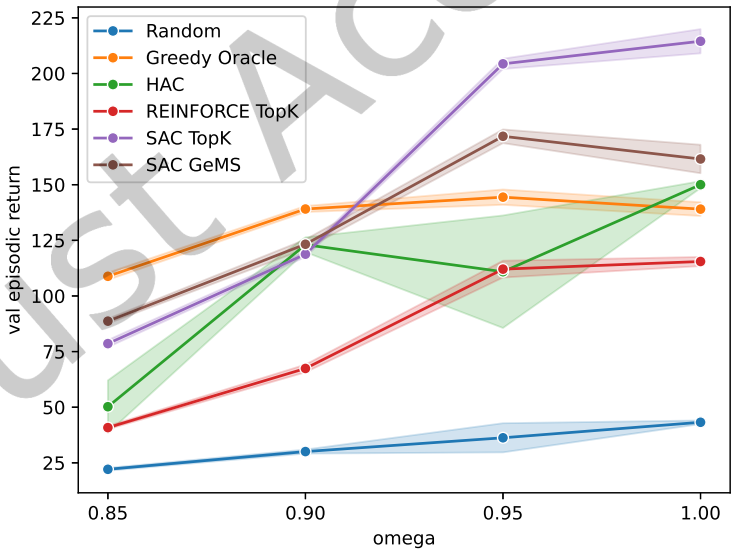


Fig. 9. Validation return after 500,000 training steps on the `SlateTopK-BoredInf` environment at different levels of influence by the clicked items (the lower ω, the higher the influence of clicked items on user preference). The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

C CLICKED ITEM INFLUENCE EXPERIMENT

In this section we focus on the effect of clicked items on the user preference, which can be controlled by the ω parameter introduced in Section 3.4. Concretely, we use the `SlateTopK-BoredInf` environment and build a performance profile of the benchmarked algorithms under increasing influence of the clicked items: $\omega \in \{1.0, 0.95, 0.9, 0.85\}$. Note that $\omega = 1$ correponds to the `SlateTopK-Bored` environment, where clicked items do not change the user preference. With $\omega < 1$, clicked items attract the user by shifting the user preference embedding towards the embedding of the clicked item. As a result, the system must control for the long-term effect of its recommendations on user preference, possibly yielding complex dynamics.

As in Section 5.2, we train all models for $500,000$ steps, with five different random seeds. In Figure 9, we report the validation episodic return obtained after $500,000$ training steps, at four different values of ω. We find that the complex dynamics created by increasing the influence of clicked items makes it harder for all methods to reach a high return. In particular, for $\omega \leqslant 0.9$, none of the tested methods manages to beat a greedy oracle agent. Moreover, the relative performance of the different methods is sensitive to the strength of item influence, with SAC+GeMS being overall more robust than its counterparts and even beating SAC+TopK under lower ω values.

D ITEM SCORES

In Figure 10 we report the distribution of scores for items recommended by the methods benchmarked on the `SlateTopK-Bored` environment. In this environment, there exists a trade-off between recommending highly relevant items and mitigating user boredom. We can indeed see that while the greedy oracle algorithm recommends only highly relevant items, it does not yield as much return as multi-step approaches like HAC, SAC+TopK and SAC+GeMS. Furthermore, we can spot differences across methods, as it appears that HAC often recommends highly relevant items, but also often defaults to poorly relevant items, in contrast to REINFORCE+TopK, which mostly avoids irrelevant items but also usually fails to accurately estimate user preferences and propose highly relevant items.

Overall, the availability and interpretability of standardized item scores within our simulator[14] allows us to complement the information contained in the final return obtained by each method, and therefore to better qualify the strengths and weaknesses of each method.

[14]Item scores are returned by the `step` method of the simulator under `info["scores"]`.

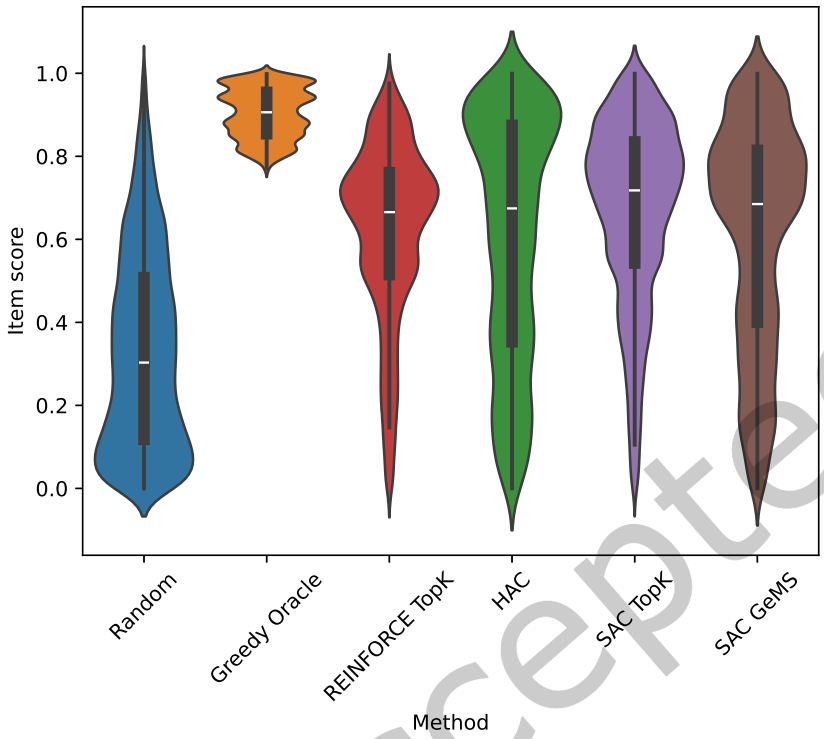


Fig. 10. Distribution of the relevance score of items recommended by different methods after training for 500,000 steps on `SlateTopK-Bored`. The higher the score, the higher the click probability. See Section 3.3 for how the relevance score is computed within the simulator.

REFERENCES

[1] Shipra Agrawal and Navin Goyal. 2013. Thompson Sampling for Contextual Bandits with Linear Payoffs. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 127–135. https://proceedings.mlr.press/v28/agrawal13.html

[2] Ashton Anderson, Lucas Maystre, Ian Anderson, Rishabh Mehrotra, and Mounia Lalmas. 2020. Algorithmic Effects on the Diversity of Consumption on Spotify. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 2155–2165. https://doi.org/10.1145/3366423.3380281

[3] Iason Chaimalas, Duncan Martin Walker, Edoardo Gruppi, Benjamin Richard Clark, and Laura Toni. 2023. Bootstrapped Personalized Popularity for Cold Start Recommender Systems. In *Proceedings of the 17th ACM Conference on Recommender Systems* (Singapore, Singapore) *(RecSys '23)*. Association for Computing Machinery, New York, NY, USA, 715–722. https://doi.org/10.1145/3604915.3608820

[4] Rana Chamsi Abu Quba, Salima Hassas, Hammam Chamsi, and Usama Fayyad. 2014. From a "Cold" to a "Warm" Start in Recommender systems. In *IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2014)*. IEEE Digital Library, Parma, Italy, 1–7. https://doi.org/10.1109/WETICE.2014.66

[5] 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 *WSDM*. 456–464. https://doi.org/10.1145/3289600.3290999

[6] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search*. Morgan & Claypool. https://doi.org/10.2200/S00654ED1V01Y201507ICR043

[7] Federico Cinus, Marco Minici, Corrado Monti, and Francesco Bonchi. 2022. The Effect of People Recommenders on Echo Chambers and Polarization. *Proceedings of the International AAAI Conference on Web and Social Media* 16, 1 (May 2022), 90–101. https://doi.org/10.1609/icwsm.v16i1.19275

[8] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An Experimental Comparison of Click Position-Bias Models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining* (Palo Alto, California, USA) *(WSDM '08)*. Association for Computing Machinery, New York, NY, USA, 87–94. https://doi.org/10.1145/1341531.1341545

[9] Romain Deffayet, Philipp Hager, Jean-Michel Renders, and Maarten de Rijke. 2023. An Offline Metric for the Debiasedness of Click Models. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 558–568. https://doi.org/10.1145/3539618.3591639

[10] Romain Deffayet, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke. 2022. Offline Evaluation for Reinforcement Learning-Based Recommendation: A Critical Issue and Some Alternatives. *ACM SIGIR Forum* 56, 2 (2022), 3:1–3:14. https://doi.org/10.1145/3582900.3582905

[11] Romain Deffayet, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke. 2023. Generative Slate Recommendation with Reinforcement Learning. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining* (Singapore, Singapore) *(WSDM '23)*. Association for Computing Machinery, New York, NY, USA, 580–588. https://doi.org/10.1145/3539597.3570412

[12] Chongming Gao, Kexin Huang, Jiawei Chen, Yuan Zhang, Biao Li, Peng Jiang, Shiqi Wang, Zhong Zhang, and Xiangnan He. 2023. Alleviating Matthew Effect of Offline Reinforcement Learning in Interactive Recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 238–248. https://doi.org/10.1145/3539618.3591636

[13] Shashank Gupta, Philipp Hager, Jin Huang, Ali Vardasbi, and Harrie Oosterhuis. 2023. Recent Advances in the Foundations and Applications of Unbiased Learning to Rank. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 3440–3443. https://doi.org/10.1145/3539618.3594247

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *ICML*. 1856–1865. https://proceedings.mlr.press/v80/haarnoja18b.html

[15] Chen He, Denis Parra, and Katrien Verbert. 2016. Interactive Recommender Systems: A Survey of the State of the Art and Future Research Challenges and Opportunities. *Expert Systems with Applications* 56 (2016), 9–27. https://doi.org/10.1016/j.eswa.2016.02.013

[16] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *ICLR*. http://arxiv.org/abs/1511.06939

[17] Jin Huang, Harrie Oosterhuis, Bunyamin Cetinkaya, Thijs Rood, and Maarten de Rijke. 2022. State Encoders in Reinforcement Learning for Recommendation: A Reproducibility Study. In *SIGIR*. 2738–2748. https://doi.org/10.1145/3477495.3531716

[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 *Proceedings of the 14th ACM Conference on Recommender Systems* (Virtual Event, Brazil) *(RecSys '20)*. Association for Computing Machinery, New York, NY, USA, 190–199. https://doi.org/10.1145/3383313.3412252

[19] Wasim Huleihel, Soumyabrata Pal, and Ofer Shayevitz. 2021. Learning User Preferences in Non-Stationary Environments. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 130)*, Arindam

Banerjee and Kenji Fukumizu (Eds.). PMLR, 1432–1440. https://proceedings.mlr.press/v130/huleihel21a.html
[20] 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-19*. International Joint Conferences on Artificial Intelligence Organization, 2592–2599. https://doi.org/10.24963/ijcai.2019/360
[21] Eugene Ie, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. arXiv:1909.04847 [cs.LG]
[22] Dietmar Jannach, Paul Resnick, Alexander Tuzhilin, and Markus Zanker. 2016. Recommender Systems — beyond Matrix Completion. *Commun. ACM* 59, 11 (oct 2016), 94–102. https://doi.org/10.1145/2891406
[23] Olivier Jeunen. 2023. A Common Misassumption in Online Experiments with Machine Learning Models. In *PERSPECTIVES workshop at RecSys'23*. https://arxiv.org/abs/2304.10900
[24] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *WSDM*. 781–789. https://doi.org/10.1145/3018661.3018699
[25] Haruka Kiyohara, Ren Kishimoto, Kosuke Kawakami, Ken Kobayashi, Kazuhide Nataka, and Yuta Saito. 2023. SCOPE-RL: A Python Library for Offline Reinforcement Learning, Off-Policy Evaluation, and Policy Selection. *arXiv preprint arXiv:23xx.xxxxx* (2023). https://github.com/hakuhodo-technologies/scope-rl
[26] Norman Knyazev and Harrie Oosterhuis. 2023. A Lightweight Method for Modeling Confidence in Recommendations with Learned Beta Distributions. In *RecSys'23*. Association for Computing Machinery, New York, NY, USA. https://arxiv.org/abs/2308.03186
[27] Hojoon Lee, Dongyoon Hwang, Kyushik Min, and Jaegul Choo. 2022. Towards Validating Long-Term User Feedbacks in Interactive Recommendation Systems. In *SIGIR*. 2607–2611. https://doi.org/10.48550/arXiv.2308.11137
[28] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. Association for Computing Machinery, New York, NY, USA, 661–670. https://doi.org/10.1145/1772690.1772758
[29] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *ICLR*.
[30] Shuchang Liu, Qingpeng Cai, Bowen Sun, Yuhao Wang, Ji Jiang, Dong Zheng, Peng Jiang, Kun Gai, Xiangyu Zhao, and Yongfeng Zhang. 2023. Exploration and Regularization of the Latent Action Space in Recommendation. In *WWW*. 833–844. https://doi.org/10.1145/3543507.3583244
[31] Angshul Majumdar and Anant Jain. 2017. Cold-start, warm-start and everything in between: An autoencoder based approach to recommendation. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 3656–3663. https://doi.org/10.1109/IJCNN.2017.7966316
[32] Tatsuya Matsushima, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu. 2020. Deployment-Efficient Reinforcement Learning via Model-Based Offline Optimization. arXiv:2006.03647 [cs.LG]
[33] Sean M. McNee, John Riedl, and Joseph A. Konstan. 2006. Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems* (Montréal, Québec, Canada) *(CHI EA '06)*. Association for Computing Machinery, New York, NY, USA, 1097–1101. https://doi.org/10.1145/1125451.1125659
[34] Rishabh Mehrotra. 2021. Algorithmic Balancing of Familiarity, Similarity, & Discovery in Music Recommendations. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) *(CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 3996–4005. https://doi.org/10.1145/3459637.3481893
[35] Prem Melville and Vikas Sindhwani. 2010. *Recommender Systems*. Springer US, Boston, MA, 829–838. https://doi.org/10.1007/978-0-387-30164-8_705
[36] Harrie Oosterhuis. 2021. Computationally Efficient Optimization of Plackett-Luce Ranking Models for Relevance and Fairness. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 1023–1032. https://doi.org/10.1145/3404835.3462830
[37] Harrie Oosterhuis and Maarten de Rijke. 2021. Robust Generalization and Safe Query-Specialization in Counterfactual Learning to Rank. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 158–170. https://doi.org/10.1145/3442381.3450018
[38] Sindhu Padakandla, Prabuchandran K. J., and Shalabh Bhatnagar. 2020. Reinforcement Learning Algorithm for Non-stationary Environments. *Applied Intelligence* 50, 11 (jun 2020), 3590–3606. https://doi.org/10.1007/s10489-020-01758-5
[39] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. 2018. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. arXiv:1808.00720 [cs.IR]
[40] Neil Rubens, Mehdi Elahi, Masashi Sugiyama, and Dain Kaplan. 2015. *Active Learning in Recommender Systems*. Springer US, Boston, MA, 809–846. https://doi.org/10.1007/978-1-4899-7637-6_24
[41] Yuta Saito, Shunsuke Aihara, Megumi Matsutani, and Yusuke Narita. 2021. Open Bandit Dataset and Pipeline: Towards Realistic and Reproducible Off-Policy Evaluation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*,

J. Vanschoren and S. Yeung (Eds.), Vol. 1. Curran. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/33e75ff09dd601bbe69f351039152189-Paper-round2.pdf

[42] Otmane Sakhi, David Rohde, and Nicolas Chopin. 2023. Fast Slate Policy Optimization: Going Beyond Plackett-Luce. arXiv:2308.01566 [cs.LG]

[43] Marlesson R. O. Santana, Luckeciano C. Melo, Fernando H. F. Camargo, Bruno Brandão, Anderson Soares, Renan M. Oliveira, and Sandor Caetano. 2020. MARS-Gym: A Gym Framework to Model, Train, and Evaluate Recommender Systems for Marketplaces. In *2020 International Conference on Data Mining Workshops (ICDMW)*. 189–197. https://doi.org/10.1109/ICDMW51313.2020.00035

[44] Masahiro Sato. 2021. Online Evaluation Methods for the Causal Effect of Recommendations. In *Proceedings of the 15th ACM Conference on Recommender Systems* (Amsterdam, Netherlands) *(RecSys '21)*. Association for Computing Machinery, New York, NY, USA, 96–101. https://doi.org/10.1145/3460231.3474235

[45] Sebastian Sequoiah-Grayson and Luciano Floridi. 2022. Semantic Conceptions of Information. In *The Stanford Encyclopedia of Philosophy* (Spring 2022 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.

[46] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng. 2019. Virtual-Taobao: Virtualizing Real-World Online Retail Environment for Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 4902–4909. https://doi.org/10.1609/aaai.v33i01.33014902

[47] Nícollas Silva, Heitor Werneck, Thiago Silva, Adriano C.M. Pereira, and Leonardo Rocha. 2022. Multi-Armed Bandits in Recommendation Systems: A Survey of the State-of-the-art and Future Directions. *Expert Systems with Applications* 197 (2022), 116669. https://doi.org/10.1016/j.eswa.2022.116669

[48] Aixin Sun. 2023. Take a Fresh Look at Recommender Systems from an Evaluation Standpoint. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 2629–2638. https://doi.org/10.1145/3539618.3591931

[49] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miroslav Dudík, John Langford, Damien Jose, and Imed Zitouni. 2017. Off-Policy Evaluation for Slate Recommendation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3635–3645.

[50] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun K.G., Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. 2023. Gymnasium. https://doi.org/10.5281/zenodo.8127026

[51] Kai Wang, Zhene Zou, Minghao Zhao, Qilin Deng, Yue Shang, Yile Liang, Runze Wu, Xudong Shen, Tangjie Lyu, and Changjie Fan. 2023. RL4RS: A Real-World Dataset for Reinforcement Learning Based Recommender System. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 2935–2944. https://doi.org/10.1145/3539618.3591899

[52] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2023. A Comprehensive Survey of Continual Learning: Theory, Method and Application. arXiv:2302.00487 [cs.LG]

[53] Yu Wang, Xin Xin, Zaiqiao Meng, Joemon M. Jose, Fuli Feng, and Xiangnan He. 2022. Learning Robust Recommenders through Cross-Model Agreement. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 2015–2025. https://doi.org/10.1145/3485447.3512202

[54] 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* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 1347–1357. https://doi.org/10.1145/3477495.3531714

[55] Eva Zangerle and Christine Bauer. 2022. Evaluating Recommender Systems: Survey and Framework. *ACM Comput. Surv.* 55, 8, Article 170 (dec 2022), 38 pages. https://doi.org/10.1145/3556536