Taming the Dynamics of Recommender Systems

Academisch Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties
ingestelde commissie,
in het openbaar te verdedigen in de Aula der Universiteit
op woensdag 2 oktober 2024, te 14:00 uur

door

Romain Emmanuel Deffayet

geboren te Épinal

PROMOTIECOMMISSIE

Promotor:

prof. dr. M. de Rijke	Universiteit van Amsterdam

Copromotores:

dr. J.M. Renders	Naver Labs Europe
dr. A.C Yates	Universiteit van Amsterdam

Overige leden:

dr. H.C. van Hoof	Universiteit van Amsterdam
prof. dr. D. Jannach	University of Klagenfurt
prof. dr. E. Kanoulas	Universiteit van Amsterdam
dr. H.R. Oosterhuis	Radboud Universiteit
prof. dr. S.J.L. Smets	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The work described in this thesis has been primarily carried out at the Information Retrieval Lab of the University of Amsterdam and at Naver Labs Europe.

Printed by Proefschriftspecialist.

ISBN: 978-94-93391-36-9

ACKNOWLEDGEMENTS

This thesis has been made not only possible, but also a pleasant and enriching time, by several people:

First and foremost, my daily supervisors: Maarten and Jean-Michel. Maarten, when I initially presented a vague research plan to you, you trusted me and offered the support required to pursue my PhD under your supervision. Thank you so much for that. Throughout the work, you taught me how to conduct research and tackle the research world while staying true to myself. It certainly required a good touch, knowing when to contain my ideas and when to push them. Jean-Michel, even before the first draft of a thesis topic, you trusted me to take on this journey and pushed me to make the most of it. By carefully inspecting everything I wrote, said, or cited, and by always questioning the arguments I formulated, you certainly improved the quality of this thesis, and taught me how to a be a better scientist. Also, thank you for enlarging my knowledge about many different topics, from machine learning to wine-making.

This thesis, to be a complete research work, required the help of those who were there from the beginning, but also those who stood for it at the end. Dietmar, Evangelos, Harrie, Herke, Sonja: thank you for being members of my committee.

I must obviously also thank my collaborators along the way. Philipp: you taught me how to get something properly done, how to extract the most out of our experiments, and how to convert an interesting idea into an impactful research work. Most importantly, thank you for being a great friend from day 0 to day -1. As I write these lines, you have pulled me out of a tricky situation once again. It seems only natural that you are my paranymph now through this last step. Sami: Despite the many ups and downs with our work, you kept being there, iterating, going forward and you never gave up. Thanks to your pugnacity with reviewers and your samisearch™ hyperparameter tuning algorithm, we got it done. Thank you for guiding me through many steps

of my professional and personal life, and for always being the devil on my shoulder. Thibaut: Thank you for always being firmly present when deadlines approached, and for pushing me to deliver convincing experiments. Your handling of the project helped me a lot structuring and managing my research. Dongyoon, Onno, Paul, Vassilissa: thank you for for your time, patience and critical thinking that definitely improved this thesis.

Now, I have to talk about the amazing environment in which I was lucky to work. First the IRLab. Gabriel, thanks for being so accessible and enthusiastic and for teaching me that one cannot suffer from too much confidence. Mariya, your kindness made me instantly feel at home in the lab. I learned a lot form you. Pooya, thank you for sharing so many interesting stories and activities. Thilina, I appreciate a lot your constant willingness to organize dinners or board games night. You might learn how to pronounce my name some day. Since I can't name everyone here, I would like to thank the entire IRLab, who really made this period a fun time. Thank you to the Dune players and the squash players, Team Café Neo and Team main building, the first slides and the last slides of the research meeting, those who unknowingly lended me their desk while they were away, those who left their desk to get me a coffee one of the countless times I forgot my access card, and most importantly all those that I spent great moments with when I travelled to Amsterdam.

Before I talk about my other research lab, NLE, it seems fitting to speak about Simon, who has been there with me on both sides since (almost) the beginning, from playing frenzied ping-pong games to introducing me to gravel riding, and now to being my paranymph. Thank you so much for that.

At Naver Labs, I met many fascinating people who became close colleagues and friends. In particular, the search and recommendation team was a great environment to work in. Stéphane, you have always been there to support me, and always encouraged me to try different things and keep learning. You had an important role in making me consider the PhD as a learning experience rather than a succession of deadlines and presentations. Carlos, thank you for helping in my projects and for sharing great moments all around the world. Hervé, the wise man of the team, thank you for your phlegm and humor. Till, thank you for all the interesting and challenging discussions, both scientific ones and about life in general. Thibault, thank you for demonstrating that even as a researcher, being relaxed and accessible is a strength. I would also like to thank everyone I met at NLE that made this thesis possible and enjoyable,

each in their own way: Augustin, Bingbing, Cécile, David, Ginger, Gustavo, Jean-Yves, Marcely, Tomi, Veronika, ...

Finally, I am thankful to my family, especially my parents, for supporting me long before this journey even started, and throughout the entire process. Eva, my development as a person and as a researcher is deeply linked to living with you. I would have been much less open, serene and happy during this time, if not thanks to you. Thank you.

CONTENTS

1

INTRODUCTION

When you start doing research on recommender systems, you get surprisingly quickly accustomed to certain idiomatic phrases: *preference* and *information need*, *relevance* and *bias*, *items* and *users*. *User*, in particular, is a peculiar word: no one primarily describes themselves as a user, and yet all our definitions seem to build on the immovable notion of a user: we recommend things that are relevant to *the user*, we estimate *the user* preference and fulfill *the user*'s information need. But who is *the user*? After asking people around me whether they are *the user*, I still haven't found out. Fortunately, it seems that researchers have created *the user* from data: it is a ranking function over possible items – another generic term – which is determined from the judgement of certain people on said items in the past (Aggarwal, 2016). When recommendations are personalized, they can often be refined based on a history of past interactions with the system, in which case *the user* is a function of interaction logs that came before the recommendation event, and its value is determined by the logs that followed (e.g., clicks, likes, watch time) (Wang et al., 2019).

Once *the user* has been built, we can conveniently learn our own models from data, which are also ranking functions, and compare them against *the user* (Robertson, 1977). Doing so essentially amounts to a static matching task: *the user* has preferences over items and we must find which ones it values the most, i.e., we must match its ranking function by learning from data. While this is a very convenient approach, *the user* does not exist. Or rather, it is made of multiple snapshots of what a person, at a given time in the past and given their environment at the time, did when facing the pieces of content returned by a (possibly unknown) recommender system. Matching the behavior of this idealized user is fine as long as these persons, their environment and the rec-

ommended results stay the same in the future (at least in distribution). When that is true, models that perform well against *the user* will continue to perform well once deployed on the actual online platform.

In this thesis, I investigate what happens when this is not the case, i.e., when there is a distribution shift between the observed data and the future interactions on the platform. In particular, I focus on the distribution shifts that result from our system's own decisions. They can typically occur in two cases: (i) when a new system is deployed and is different from the previous version that was in place during data collection, which happens at virtually every deployment as the goal is usually to improve the recommender system in production (Oosterhuis, 2021b); and (ii) whenever the recommended items have the power to shape future user behavior, i.e., when recommendations are *performative* (Perdomo et al., 2020; Wang et al., 2024). In the first case, the previous recommender system may have constrained the choice of the person whose interactions are logged in the data (for instance by placing results on the page in an unequal manner or by not returning at all certain results), and therefore the value of items may have been wrongly reflected in the data. Regarding the second case, we must acknowledge that the recommendations can change a person's behavior (for example by boring them due to redundant recommendations) (Gao et al., 2023) and even change their worldview (e.g., by exposing them to biased world representations) (Cinus et al., 2022).

But contrary to exogenous distribution shifts (e.g., due to people being influenced by their social circle or due to changes in the world that affect the value of certain items), the effects described above directly result from decisions that we control, and therefore that we can adapt. Controlling for these unexpected dynamic effects of our recommendations on the future user experience is crucial as a large part of the information available on the internet is algorithmically filtered before reaching its intended target user (European Commission, 2024). Moreover, accounting for such effects is not merely a way to avoid potential pitfalls of static recommender systems, but can also be seen as an opportunity: to better help people navigate the large amounts of available information by getting them interested in topics they didn't know initially or by providing a consistent sequence of recommendations as they keep using the online service.

Throughout the thesis, I employ various techniques that present favorable properties towards harnessing the complex dynamics of recommender systems: reinforcement learning (RL), including its distributional variant, unbi-

ased learning to rank through click modeling, generative user modeling, ... Essentially, I wish to identify what these techniques can bring to recommender systems and more importantly, to the people using them, what are the challenges in applying these general methodologies to the very specific recommendation task, and finally what kind of tools they require to be made reliable and trustworthy. While it is illusory – and undoubtedly inappropriate – to expect recommender systems to perfectly understand and control the inner mechanisms that explain how people behave on the internet, I argue that developing tools that are able to observe, forecast and manage the performative effects of the systems that power most online services is both key to their improvement and a societal responsibility.

1.1 RESEARCH QUESTIONS

I try to address certain specific questions that stem from the overarching research topic described above. I list them here.

Before training any model that could be capable of handling dynamic and interactive recommendation environments, we must make sure our evaluation setup is up to the task. I therefore introduce the first research question:

Research Question 1. *How can we evaluate recommender systems in a way that accounts for their dynamic and interactive nature?*

This topic is mostly covered in Part I, i.e., Chapter 2 and Chapter 3. Specifically, in Chapter 2, we review how reinforcement learning-based models, which are often seen as being promising for controlling the dynamics of recommender systems (Afsar et al., 2022), are usually evaluated in the context of recommendation. We find that the common static approach of next-item prediction is inadequate in this new, dynamic setting, and list its limitations. We then propose alternative ways forward. One of these is the use of simulators as an evaluation tool, in a research context. In Chapter 3, we therefore propose a simulator suited to research the dynamic aspects of recommendation.

Distribution shifts also manifest themselves in the form of biased data used for training new models. While selection bias (e.g., some items have not been recommended and we therefore don't know their value) is prevalent in virtually all application fields of offline bandit and reinforcement learning (Levine

et al., 2020), information retrieval systems present a more unusual kind of bias: multiple items are often returned at once, and their presentation on the result page may affect the likelihood of the user engaging with these items. Therefore, the observed data is not a reliable indicator of item value. This is clearly an issue for later deployment where models learned on such biased data may perform poorly or unfairly promote certain items above others. This leads to the following question:

Research Question 2. *Can we predict in a fully offline manner the performance of models learned on biased data?*

We review the existing studies on offline evaluation of learning-to-rank algorithms under biased data and highlight their limitations in Chapter 2. Chapter 4 and Chapter 5 are dedicated to answering this question. In Chapter 4 we perform a large empirical study on click models, i.e., models that aim to learn user biases from logged data. Then in Chapter 5, we propose an offline metric that yields a better correlation with the performance of the learned click models after deployment.

It quickly becomes apparent that making assumptions about user behavior, for instance by leveraging user studies, often helps to enable an accurate estimation and mitigation of bias, or simply to correctly capture the structure of the data and the value of recommendations. However, making such assumptions comes at a cost: they may not fit the actual user behavior well. We therefore investigate this issue:

Research Question 3. *When do we need assumptions on user behavior, and how can we test for the validity of these assumptions?*

In Chapter 4 and Chapter 5, we compare click models making different assumptions about the user behavior. We find that, generally, simplifying the problem by imposing strong constraints on the learned parameters helps mitigate bias in the data, and that simpler user assumptions lead to more robust performance on the downstream task, even when the actual user behavior is more complex. Moreover, we find that our metric proposed in Chapter 5 is an effective way of choosing the right set of assumptions. In Chapter 6, we take a different perspective, and ask the question: can we train reinforcement learning-based recommender systems without formulating assumptions on data structure and user behavior. We propose an assumption-free reinforcement learning agent that is typically able to capture the signal in the data.

A major challenge in training such an assumption-free RL agent is the size of the action space, which can grow very large in recommender systems scenarios: from a few hundred or thousand possible items for small-scale recommender systems to an intractably large number of possible slates (i.e., lists of items) for large-scale recommender systems that present multiple items at a time. This is summarized by the following research question:

Research Question 4. *How to train reinforcement learning agents that recommend slates of items to users effectively and efficiently?*

Chapter 6 is entirely dedicated to this RQ, and we propose an approach based on pretraining a generative model of slates and user responses, and then using its latent space as action space for a continuous control RL agent.

Finally, another prevalent challenge in recommender systems, and even more so when we consider their dynamics, is very high uncertainty about user preference, item value, feedback reliability, ... The poor observability of user preference, coupled with click noise and dynamics that can be very different depending on the specific user we consider, make the whole process very uncertain. Recommender systems should therefore be able to assess and adapt to this uncertainty.

Research Question 5. *How can we train reinforcement learning algorithms to handle high degrees of uncertainty, which is common in interactive recommender systems?*

In Chapter 7, we investigate distributional reinforcement learning, i.e., reinforcement learning agents that learn the full distribution of future returns instead of their expected value. Distributional RL agents are known to handle uncertainty in the environment better than their point estimate counterparts, but we note that maintaining a theoretically valid estimation of the full distribution usually comes at the cost of decreased performance. We therefore propose a method that is valid in theory and matches the performance of the most effective agents.

1.2 CONTRIBUTIONS

In this section, I list the main contributions of this thesis. I split them into theoretical contributions, i.e., proofs, insights and guidelines, and practical contributions, i.e., implementations, benchmarks, and resources.

Theoretical contributions

- A formalization of the next-item prediction evaluation protocol in sequential recommendation and its limitations for evaluating interactive recommender systems. (Chapter 2)
- A set of requirements for a research-oriented simulator of dynamic and interactive recommender systems. (Chapter 3)
- The discovery of a critical limitation of annotator-based evaluation under policy distribution shift: biased models can obtain very high performance on annotation-based ranking metrics. (Chapter 4)
- An offline metric for the debiasedness of click models. (Chapter 5)
- A comparison of assumptions commonly used in reinforcement learning for slate recommendation. (Chapter 6)
- A proof of the Bellman-closedness of mapped expectiles, that indicates that it is possible to learn theoretically valid distributional RL agents using efficient L_2 loss functions. (Chapter 7)

Practical contributions

- A simulator for interactive recommender systems research, and a benchmark of common approaches in various scenarios. (Chapter 3)
- A benchmark of the robustness of various click models to policy distribution shift, and their performance on downstream tasks. (Chapter 4)
- An assessment of the predictive power of our proposed metric for the debiasedness of click models and its usefulness for selecting models to be deployed. (Chapter 5)
- An assumption-free agent for RL-based slate recommendation. (Chapter 6)
- A dual implicit expectile-quantile networks agent for distributional reinforcement learning. (Chapter 7)

1.3 THESIS OVERVIEW

I now provide an overview of how the thesis is organized. The thesis is thematically split into three parts, each containing two chapters. While each part – and each chapter – can be read independently, I recommend starting with Part I, especially Chapter 2, in order to get a better understanding of how this thesis builds on considerations that are different from many existing studies on recommender systems. Chapter 5 should ideally be read right after Chapter 4, as it is a direct follow-up of the work presented in Chapter 4. Chapter 3 includes in its experiments the method presented later in Chapter 6, but a detailed understanding of how this method works is not necessary when reading Chapter 3.

1.4 ORIGINS

The research chapters in this thesis are based on the following publications:

(Chapter 2) Romain Deffayet, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke. 2023. Offline Evaluation for Reinforcement Learning-based Recommendation: A Critical Issue and Some Alternatives. In *SIGIR Forum 56, 2, Article 3 (December 2022)*.
RD proposed the idea. RD and TT surveyed relevant articles. JMR and MdR had an important advisory role. All authors participated in writing. RD did most of the writing.

(Chapter 3) Romain Deffayet, Thibaut Thonet, Dongyoon Hwang, Vassilissa Lehoux, Jean-Michel Renders, and Maarten de Rijke. 2024. SARDINE: A Simulator for Automated Recommendation in Dynamic and Interactive Environments. In *ACM Transactions on Information Systems (TORS), "Just accepted"*.
RD proposed the idea and drew the requirements. RD, TT, DH, VL paritcipated in the implementation. RD and TT performed experiments. All authors participated in writing. RD and TT did most of the writing.

(Chapter 4) Romain Deffayet, Jean-Michel Renders, and Maarten de Rijke. 2023. Evaluating the Robustness of Click Models to Policy Distributional

Shift. In *ACM Transactions on Information Systems (TOIS) 41, 4, Article 84 (October 2023)*.
RD proposed the idea, implemented the models, and performed the experiments. JMR and MdR had an important advisory role. All authors participated in writing. RD did most of the writing.

(Chapter 5) **Romain Deffayet**, Philipp Hager, Jean-Michel Renders, and Maarten de Rijke. 2023. An Offline Metric for the Debiasedness of Click Models. In *SIGIR'23: the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*.
RD proposed the idea and theory. PH derived the metric and implemented the evaluation setup. RD and PH implemented models and ran the experiments. JMR and MdR had an important advisory role. All authors participated in writing. RD and PH did most of the writing.

(Chapter 6) **Romain Deffayet**, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke. 2023. Generative Slate Recommendation with Reinforcement Learning. In *WSDM'23: the 16th ACM International Conference on Web Search and Data Mining*.
RD proposed the idea and implemented the method. RD and TT created the experimental setup and ran the experiments the models. JMR and MdR had an important advisory role. All authors participated in writing. RD and TT did most of the writing.

(Chapter 7) Sami Jullien, **Romain Deffayet**, Jean-Michel Renders, Paul Groth, and Maarten de Rijke, 2024. Distributional Reinforcement Learning with Dual Expectile-Quantile Regression. *Under review*.
SL and RD did the initial investigation and proposed the idea. RD built the theory and the toy experiment. SJ derived the practical agent and ran the large-scale experiments. JMR and MdR had an important advisory role. All authors participated in writing. RD and SJ did most of the writing.

The thesis also benefitted from the following publication: Philipp Hager, **Romain Deffayet**, Jean-Michel Renders, Onno Zoeter, and Maarten de Rijke, 2024. Unbiased Learning to Rank Meets Reality: Lessons from Baidu's Large-Scale Search Dataset. In *SIGIR'24: the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Part I

Evaluation of Dynamic and Interactive Recommender Systems

2

OFFLINE EVALUATION FOR REINFORCEMENT LEARNING-BASED RECOMMENDATION: A CRITICAL ISSUE AND SOME ALTERNATIVES

With this first research chapter, we try to establish a sound basis for research on dynamic and interactive recommender systems. We review the growing trend of modeling recommender systems in an interactive fashion, mostly by training reinforcement learning agents. In particular, we focus on how reinforcement learning-based recommender are usually evaluated. We find that most studies use a form of next-item prediction, where the next interacted item in the sequence of interactions acts as a label, whose score should be maximized by the agent.

We note that this evaluation protocol is unsuited to RL agents: on the one hand it cannot reflect the expected benefits that RL can supposedly bring, and on the other hand, some critical pitfalls of RL agents trained offline can fly under the radar of this type of evaluation. Consequently, we give our suggestions going for forward, and list a few existing or emerging alternatives.

This chapter is based on the following publication: **Romain Deffayet**, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke. 2023. Offline Evalua-

tion for Reinforcement Learning-based Recommendation: A Critical Issue and Some Alternatives. In *SIGIR Forum 56, 2, Article 3 (December 2022)*.

2.1 INTRODUCTION

Recommender systems play a major role in defining internet users' experience due to their ubiquitous presence on, e.g., content providing and e-commerce platforms. Correct and careful evaluation of recommender systems is therefore critical as it directly impacts business metrics as well as user satisfaction – and sometimes even society as a whole.

While recommendation accuracy (i.e., recommending relevant items) is often taken to be the main indicator of performance, the literature on recommender systems highlights the importance of additional criteria. Beyond-accuracy goals include, e.g., diversity, novelty or serendipity, fairness, and user experience in general (McNee et al., 2006a). Such criteria sometimes cannot be enforced in one-shot recommendation (i.e., in a single interaction between the user and the recommender system) but they may require that we consider the longer-term experience. These concerns have motivated researchers and practitioners alike to acknowledge the sequential nature of many recommendation engines, and to seek to optimize over whole sequences instead of one-shot predictions (Quadrana et al., 2018).

Reinforcement learning (RL) formulates this problem as a Markov decision process (MDP), in which we wish to select appropriate actions (i.e., item recommendations) in order to maximize the sum of rewards (e.g., clicks, purchases, etc.) along the full sequence of user interactions with the recommender system. RL is a natural fit for this problem because the underlying MDP is able to model the long-term influence of recommendations on the user. Note that in recommendation scenarios, online exploration is often impossible, so the policy must be trained from a fixed dataset of interactions, i.e., by offline RL. While sequence optimization with offline RL is not expected to entirely fulfill all the desired beyond-accuracy criteria highlighted in the literature, it holds the promise of making some of the desired properties naturally emerge as a result of whole-sequence optimization. Indeed, one can expect that, given an appropriate reward function, policies that are effective over the entire span of the user's experience require some of these desired properties: diversity,

novelty, etc. Because these auxiliary metrics are embedded into a sequence's cumulative reward, whole-sequence optimization with RL can be seen as a way to bridge the gap between offline and online performance.

In this chapter, we argue that the progress supposedly achieved in sequential recommendation, thanks to RL, lacks *ecological validity* (Andrade, 2018): the trained agents are likely not to generalize to real-world scenarios, because of certain shortcomings in the current evaluation practices. Namely, RL-based recommender systems are often evaluated in an offline fashion, following a traditional one-shot accuracy-oriented protocol that cannot capture the potential benefits introduced by the use of RL algorithms. We refer to this evaluation protocol as *next-item prediction* (NIP). More critically, we highlight that the specifics of this protocol are likely to hide the deficiencies of recommender systems trained by offline RL. Briefly, we argue that *with the most commonly employed evaluation practices, we cannot verify that the RL algorithm correctly optimizes the very metric it is designed to optimize*, i.e., expected cumulative reward. We worry that instead of bridging the gap between offline and online performance, it only widens it. We then provide suggestions towards a sound evaluation methodology for RL-based recommendation in order to help practitioners and researchers avoid common pitfalls and to inspire future research on this important topic.

After contrasting our criticism with that formulated by previous studies in Section 2.2, in Section 2.3 we provide a definition of the *next-item prediction* (NIP) evaluation protocol along with an overview of its use in sequential recommendation with RL. Section 2.4 dives into the three major issues of the NIP protocol, and their implications for the evaluation of RL-based recommender systems. Finally, we formulate our suggestions towards a sound evaluation methodology in RL-based recommendation in Section 2.5.

2.2 RELATED STUDIES

Deficiencies in recommender systems evaluation have been a long-standing problem in the recommendation literature. In this section we review previous studies that discuss this topic.

Firstly, as we recalled in the introduction, McNee et al. (2006a) and Jannach et al. (2016) have highlighted the need for recommender systems that go beyond

accuracy of the proposed item, i.e., which do not only consider recommendation as a matrix completion problem. This is motivated by an observed gap between offline and online performance, sometimes rendering any conclusions drawn from offline evaluation obsolete (Garcin et al., 2014; Gomez-Uribe and Hunt, 2016; Jeunen, 2019).

Secondly, pitfalls of recommender system evaluation – including the next-item prediction protocol for offline evaluation that we focus on in this chapter – have been extensively discussed in the past: Chen et al. (2017), Jeunen (2019), Ji et al. (2020), Cremonesi and Jannach (2021), Sun (2023), and Zhao et al. (2022) highlighted multiple issues resulting from data leakage and other dataset construction fallacies, which can lead to counter-intuitive statements. The presence of selection bias in the data used for evaluating recommender systems from implicit feedback has also been identified as a major source of inaccuracies (Gomez-Uribe and Hunt, 2016; Jannach et al., 2016; Chen et al., 2017; Jeunen, 2019). In addition, and more specifically to the next-item prediction protocol, Krichene and Rendle (2020) and Zhao et al. (2022) have shown that sampling negative items at inference time in order to ease the computation of ranking metrics leads to drawing incorrect conclusions on the recommendation performance.

Finally, many studies reaffirm the importance of appropriate baseline selection in order to ensure that progress has been made, and have shown that certain claims do not hold against properly tuned baselines (Ludewig et al., 2019; Ferrari Dacrema et al., 2019; Rendle et al., 2019; Sun et al., 2020; Zhao et al., 2022).

The argument we formulate in this chapter is specific to RL-based recommendation and while it has, to the best of our knowledge, never been expressed, it is not incompatible with the issues listed in this section. It is rather to be considered as an additional caveat when evaluating RL-based recommender systems.

2.3 NEXT-ITEM PREDICTION IN RL-BASED RECOMMENDATION

We propose an (informal) definition of *next-item prediction* that encompasses the offline evaluation protocols of many sequential recommendation studies, and that we consider to be problematic when used to evaluate RL-based recommender systems:

Definition 1. *Next-item prediction* (NIP) is an offline evaluation protocol for sequential item recommendation from real user feedback. The task is to ensure that the next interacted item is among the top items ranked by the model, given the sequence of past interactions. Model performance is measured according to ranking metrics (e.g., hit rate, recall, NDCG, etc).

We propose this definition because it is representative of the evaluation setup adopted in many sequential recommendation studies, e.g., GRU4REC (Hidasi et al., 2016), and also encompasses several variants. In particular, the choice of "next interacted item" can vary depending on the dataset and task at hand: the next clicked item in content recommendation (e.g., Last.fm (Last.fm, n.d.)), the next purchased product in product recommendation (e.g., RecSys Challenge 2015 (Ben-Shimon et al., 2015) or RetailRocket (RetailRocket, 2016)), the next highly rated movie in movie recommendation (e.g., MovieLens (GroupLens, n.d.)), the next basket in grocery shopping (Instacart, 2017), etc.

How prevalent is it in RL-based recommendation? RL-based recommendation (RL4REC) has become increasingly popular in recent years: we counted 55 papers about RL4REC in the proceedings of major information retrieval and recommender systems (or related) conferences between January 2017 and October 2022. To obtain this result, we queried "reinforcement learning recommendation" and "reinforcement learning recommender" on DBLP[1] and included papers published at AAAI, CIKM, ICDM, IJCAI, KDD, RecSys, SIGIR, WSDM or WWW. Figure 2.1 shows the increasing trend in published RL4REC papers. Out of the 55 papers retrieved from DBLP, we identified 39 papers that address sequential item recommendation using RL-based approaches. Other tasks irrelevant to our argument included conversational recommendation or explainable recommendations, so we ignore papers related to these topics in

1 https://dblp.org/

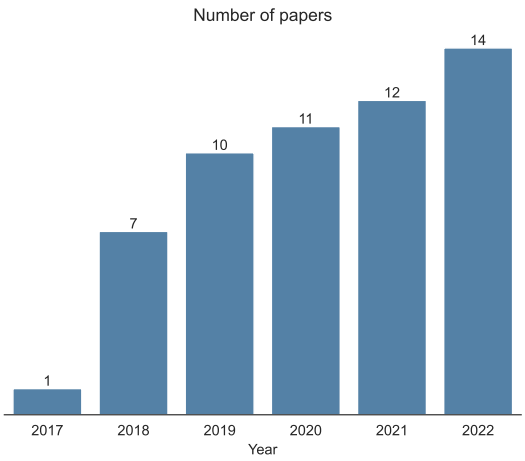


Figure 2.1: Evolution of the number of RL-based recommendation papers published in major RecSys and IR conferences between 2017 and 2022.

this study. Among the 39 relevant articles, we found 24 papers performing a form of offline evaluation, including 22 papers that followed the NIP protocol from Definition 1. The 15 other papers exclusively rely on online evaluation, either in production using an industrial recommendation platform or based on a simulator. *The NIP protocol is therefore by far the most commonly adopted type of offline evaluation.*

2.4 THREE SHORTCOMINGS OF NIP

Before engaging with the explanation of the issues with next-item prediction, we would like to recall the benefits promised by the use of RL algorithms:

- RL aims to optimize long-term outcomes resulting from a sequence of decisions. This requires accounting for the effect of the recommender on the user. RL-based methods are able to optimize whole-sequences by assigning the credit for observed rewards to individual actions, thereby preventing costly search throughout the combinatorial space of action sequences.
- RL algorithms learn in a self-supervised manner, by maximizing scalar rewards. Doing so allows them to recover open-ended solutions and generate novel policies. However, training the agent in an offline fashion

also comes with the risk of deriving policies with inaccurate estimation of their expected return.

In the following, we list three major shortcomings of the NIP protocol for evaluating offline RL agents, and explain how they harm the ecological validity of the claims derived from this evaluation protocol.

2.4.1 A myopic evaluation

Evaluating an offline RL-based recommender system using Definition 1 only accounts for short-term rewards and ignores the causal effect of the recommendations on the user. Indeed, an important motivation to design RL algorithms is to maximize the return (i.e., sum of rewards) along full trajectories, as opposed to bandit algorithms that aim to maximize the average reward at each timestep. When the actions (i.e., recommendations) cause the environment (i.e., user) to change its state, RL algorithms still have convergence guarantees, while the environment appears as non-stationary to bandit algorithms that fail to find the optimal policy both in theory and in practice. But the *next-item prediction* evaluation protocol only requires short-term thinking as it rewards one-shot prediction of the next interacted item – this is due to the offline, static nature of the evaluation that overlooks the causal impact of the recommendation policy of interest over subsequent interactions. This argument has been formulated by Lee et al. (2022a), who also empirically verified that greedy, myopic agents achieve similar or better performance on the NIP protocol than long-term-aware RL agents on standard recommendation datasets. Quadrana et al. (2018) also warned about the limits of the NIP evaluation protocol in sequential recommendation when not only immediate satisfaction but also diversity or user guidance in content discovery is desired.

However, in contrast to Lee et al. (2022a), we additionally argue that the inclusion of delayed rewards such as dwell-time in content recommendation or lifetime value in product recommendation would not be sufficient to solve this issue. Indeed, the long-term outcomes encoded in the delayed reward (e.g., was the product satisfactory over its whole lifetime?) can be orthogonal to the long-term outcomes encoded in the sum of rewards along the trajectory (e.g., was the trajectory diverse enough to avoid boring out the user?). While the former clearly seem to be important in order to obtain useful and enjoyable recommender systems, the latter are the ones that are modeled by the Markov

decision process underlying the RL agent. Consequently, if we include delayed rewards but ignore the long-term outcomes induced by the sequential decision-making process, we still cannot observe the benefits brought by RL training from the NIP protocol. Note that these two types of long-term outcomes are not incompatible and we recommend using a reward function that is as close as possible to the user's needs and satisfaction, including delayed outcomes.

2.4.2 A suboptimal target

As explained in Section 2.3, in datasets commonly employed for next-item prediction, we observe the rewards (e.g., clicks, purchases) only on the items that the user interacted with. This incurs a selection bias in the evaluation protocol, caused by the application of a particular treatment to the user. This treatment can take the form of a logging policy or a mixture of logging policies when data is gathered from organic interactions on recommendation platforms, or the implicit effect of exogenous factors when the observed data is the result of active user feedback, e.g., voluntary movie reviews or product search. We refer to the latter kind of bias as an implicit logging policy for simplicity. Note that another source of sub-optimality of the interacted items is that user choice may also be shortsighted or reluctant to novelty, even though acting so may lead to a less enjoyable experience overall.

By considering the fact that selecting the interacted item is a binary target, instead of a scalar reward to be maximized, the NIP evaluation incentivizes researchers and practitioners to build policies that are close to the (implicit) logging policy, at the expense of choosing optimal actions. It is a close-ended task of policy matching while RL allows for open-ended outcomes, i.e., generating novel policies achieving high return. There exist simpler methods to replicate the policy which generated the data, e.g., imitation learning (Hussein et al., 2017), and the reward maximization objective of RL is likely to deteriorate the results on this evaluation by selecting items that are different from the interacted item but incurring higher returns. Consequently, NIP will discard performant policies and encourage policies similar to the logging policy, even when the sequences in the dataset were highly suboptimal. Considering stronger signals such as purchases or high ratings mitigates this issue, but the selection bias that users were exposed to during data collection implies that some highly rewarding items are likely discarded.

2.4.3 Risky deployment

The two previous points that we have formulated indicate that the *next-item prediction* evaluation cannot reflect the potential benefits brought by offline RL-based recommender systems. The third problematic aspect that we discuss shows that next-item prediction may also hide critical deficiencies of offline RL agents.

Even though in the evaluation protocol of Definition 1 we account for the position of the next interacted item in the model predictions, through the use of ranking metrics, the recommender system will only select its most preferred item (or top-k most preferred items in slate recommendation) when used in production, while none of the other items will be shown to the user. It therefore seems crucial to ensure that the top item is satisfactory, regardless of the full ranking. This is unfortunately not possible with a fixed dataset where only one or a few items have been shown to the considered user. A tacit assumption of NIP is that higher ranking metrics correlate with a top item causing high return. However, a gap between offline and online results has been identified in previous studies (Garcin et al., 2014; Gomez-Uribe and Hunt, 2016). More importantly, it has been shown that even under the strong assumption that the Q-value associated to every action (i.e., item recommendation) can be correctly estimated in expectation (i.e, no bias), there can be an overestimation of the predicted offline reward with respect to the actual online reward, because the selected item is more likely to be one of those with an overestimated Q-value (Jeunen and Goethals, 2021). This phenomenon is called the *optimizer's curse*, and while its practical impact in certain cases can be limited, we argue that it can critically affect RL algorithms. Indeed, a particular set of conditions has been identified to cause a catastrophic impact of the optimizer's curse and is often called the *deadly triad* (Hasselt et al., 2018; Sutton and Barto, 2018a). It can be observed with most RL algorithms and occurs when (i) the value estimate at one state is used to update the value estimate at the previous state, (ii) function approximation is used to build the estimate of the value function, and (iii) the RL agent is trained in an off-policy fashion.

Under such conditions, small overestimations of the value function on out-of-distribution actions can be amplified and propagated to neighboring states and actions, potentially leading to divergence of the value function. In that case, while the model predicts high Q-values for its policy, the observed return after

deployment can be arbitrarily bad. The highly damaging effect of the deadly triad has been observed in multiple scenarios and motivated the emergence of extensive research on offline reinforcement learning (Hasselt et al., 2018; Fu et al., 2019; Fu et al., 2020; Levine et al., 2020; Brandfonbrener et al., 2021; Kostrikov et al., 2022). Unfortunately, this harmful phenomenon cannot be detected in the standard *next-item prediction* evaluation of Definition 1: while the interacted item may rightfully be ranked high by the model, it is likely that at least one out-of-distribution item is drastically overestimated and preferred by the model. Since this item will be the one selected by the model, we may observe an unpredicted catastrophic failure at deployment time. Even worse, this probability of failure tends to increase with the size of the action-space (Gu et al., 2022), which can be enormous in certain recommendation scenarios.

2.4.4 Upshot

The three shortcomings we presented in this section render offline evaluation using the NIP protocol of RL-based recommender systems unreliable. They effectively widen the gap between offline and online metrics, where RL algorithms were actually supposed to bridge this gap. In the next section, we suggest potential solutions to address this issue.

2.5 SOME ALTERNATIVES TO NIP

The limitations of NIP make offline evaluation of RL-based recommender systems difficult. We detail below some partial solutions to this problem and discuss their limitations and remaining open questions.

2.5.1 Online evaluation in recommendation platforms

The most obvious counter-measure to the issues raised above is to evaluate recommender systems online when possible, directly on the metrics we care about. This is usually done by deploying the policies on an actual recommendation platform. However, it is obvious that not all researchers and practitioners have access to an operational industrial platform, and online evaluation itself may

include other forms of biases, e.g., through the inclusion of business rules in recommendations. Online evaluation clearly circumvents the three issues we highlighted in the previous section, but since the focus of this chapter is on offline evaluation, we will not further detail it.

2.5.2 Counterfactual off-policy evaluation

There is a large body of work on off-policy evaluation (OPE) in information retrieval, often based on techniques such as inverse propensity scoring (Swaminathan and Joachims, 2015; Joachims et al., 2017), where a propensity weight is applied to rescale the observed rewards and returns. Although OPE has mostly been tackled for the one-shot bandit problem, some studies address OPE of RL policies both in the RL community (Fu et al., 2021) and in the IR community (Chen et al., 2019b), and more recently a library for off-policy evaluation of RL algorithms in IR has been proposed in (Kiyohara and Kawakami, 2022).

Counterfactual methods for off-policy evaluation are attractive in that they can provide unbiasedness guarantees under mild assumptions. However, we want to stress three (known) deficiencies of these methods: (i) IPS suffers from a notoriously high variance which becomes exponentially higher when applied on sequences, because of the product of inverse propensity weights (Precup et al., 2000); (ii) in non-tabular settings (i.e., when one can generalize the predictions from a state-action pair to another, for example with continuous spaces), generalization capabilities must implicitly or explicitly be assumed when the logging policy is not known, in order to compute the propensity (Hanna et al., 2019); and (iii) when we train RL algorithms in an offline manner, the error of the off-policy training and of the off-policy evaluation are likely correlated, which means that counterfactual OPE may still be biased and wrongly choose certain methods above others. An extreme example of the latter occurs if we train and evaluate a policy-gradient recommender with the same propensity weights, which makes the agent appear as optimal regardless of its true performance. While using an ensemble of estimators might mitigate this issue, it remains unclear how to fully alleviate this issue. Counterfactual OPE circumvents all three shortcomings highlighted in the previous section in theory, but as we have seen it comes with its own shortcomings which may make it unreliable in certain practical settings.

2.5.3 Simulator-based evaluation

Simulators have proved useful to assess progress in other domains, such as robotics, games or industrial applications (Fu et al., 2020; Gulcehre et al., 2020; Qin et al., 2021). While the interaction with a recommender system is arguably one of the hardest problems to simulate because of the complexity and apparent stochasticity of human behavior, the true value of simulators lies in their ability to observe how recommenders react under a chosen set of assumptions on user behavior. Additionally, by allowing the researcher to access otherwise unobservable metrics, they can enlighten us on the inner workings of the systems we build.

Many studies proposed to build semi-synthetic simulators, where the synthetic part is as limited as possible in order to adhere to real-world scenarios. This can for instance be done by using real item embeddings (Shi et al., 2019) or by extending the implicit feedback to unseen data, with debiasing in the missing-not-at-random case (Huang et al., 2020). Moreover, it is possible to assess the generalizability of a method by benchmarking it against a wide range of simulated configurations, so as to mitigate the influence of simulator design on the results. Regardless of the chosen setup, one should ensure that the simulator exhibits the characteristics we wish to model, most notably long-term influence of the recommender system on the user.

Simulators are not sensitive to the three issues of the NIP protocol, but their ecological validity may clearly be limited. On top of building simulators from real data, some approaches aim to bridge the gap between simulation and reality, for example with domain randomization (Tobin et al., 2017; OpenAI et al., 2020).

2.5.4 Intermediate evaluation

By intermediate evaluation, we refer to the offline evaluation of models, simulators or propensities that are used as building blocks in the final recommendation model (Huang et al., 2020; Deffayet et al., 2023b). In certain cases, it may be easier to evaluate these intermediate models than the final model, for example when they can be evaluated thanks to the availability of human annotations, e.g., of item relevance. By breaking down the evaluation protocol into several components, we can isolate and reduce the sources of bias. For

instance, in top-k recommendation for cumulative click maximization, if the click model is correctly estimated, i.e., the relevance and propensity scores are correct, then only state dynamics (i.e., how a user changes in response to a recommendation) are left as a source of uncertainty.

Doing so mitigates the risks associated with deploying RL agents, but does not suppress them. Moreover, we want to stress that offline RL agents will likely use the intermediate models outside of their training distribution in order to perform policy evaluation, and therefore may exploit inaccuracies in these high uncertainty regions if no proper countermeasure is applied (Deffayet et al., 2023b).

2.5.5 Uncertainty-aware evaluation

While it may not be feasible to accurately evaluate the final performance of an RL policy in a purely offline fashion, we argue that quantifying its performance at different levels of uncertainty can help assess the risks of deployment. Indeed, the value overestimation issue highlighted in the previous section results from the high uncertainty on out-of-distribution state-action pairs. We can constrain the RL algorithm to recover safe policies, that stay within the distribution of the logging policy, or allow exploration in order to find potentially high-return policies, at the cost of increasing uncertainty (Brandfonbrener et al., 2021). By quantifying the match between the support of the logging policy and that of the target policy, we can assess the risk induced by the deployment of the target policy. In particular, if we restrict the set of available actions to those considered "in-support", we can get an accurate estimate of the performance of the policy on those actions. Indeed, uncertainty is low inside the support of the logging policy, and it is anyway possible to evaluate the quality of the Q-value prediction on a held-out test set of the offline dataset as in, e.g., (Ji et al., 2021). A safe policy achieving high in-support expected return would constitute a reliable improvement, while an unsafe policy not even achieving good in-support expected return can probably be discarded. This type of evaluation needs a proper definition of in-support and out-of-support, e.g., as in (Fujimoto et al., 2019; Gu et al., 2022), which is not trivial in the non-tabular setting and requires assuming a certain degree of tolerance to uncertainty, but Kumar et al. (2021) show that it is possible to adjust this tolerance based on the training curves of certain offline RL algorithms.

This type of evaluation focuses on characterizing and mitigating the risks induced by the third issue we raise in Section 2.4.3, while potentially allowing us to detect the benefits brought by RL training. The main open question lies in the ability to properly define distance measures between the support of the logging and target policy.

2.6 CONCLUSION

In this study, we highlighted that the most commonly employed protocol for the offline evaluation of RL-based recommender systems is in fact unsuitable, because it cannot reflect the benefits that RL supposedly brings compared to more traditional approaches and because it may hide critical deficiencies of offline RL agents that can lead to catastrophic deployment. These shortcomings can be summarized as follows: (i) a myopic protocol aimed only at measuring shortterm accuracy, (ii) a close-ended, suboptimal recommendation target, and (iii) sensitivity to the optimizer's curse.

As of now, there exists no truly satisfactory solution to the problem of evaluating RL policies in an entirely offline fashion. Yet, several proxies for online performance can be used to bridge the gap between offline metrics and online performance. Finding appropriate offline evaluation protocols is still an active research area in the offline RL literature, and we urge the sequential recommendation community to join the effort and develop protocols suitable for the recommendation scenario. Additionally, acknowledging the presence of uncertainty in the deployment of RL-based recommender systems paves the way towards solutions that are robust or resilient to such uncertainty. For instance, Oosterhuis and de Rijke (2021a) propose a criterion for fallback to a safer policy when out-of-distribution (although in a different context, i.e., counterfactual learning to rank), and Ghosh et al. (2022) and Reichlin et al. (2022) propose adaptive offline RL policies that are able to recover from stepping in uncertain states during deployment by branching back to supported states. We hope that future research in recommender systems will put stronger emphasis on these aspects and reduce the gap between offline and online performance.

2.7 REFLECTIONS ON THE CHAPTER

2.7.1 Research outcomes

In this chapter, we worked towards answering my first research question:

Research Question 1. *How can we evaluate recommender systems in a way that accounts for their dynamic and interactive nature?*

We highlighted the limits of the traditional next-item prediction evaluation, and proposed some alternatives that are more suited to dynamic recommender systems. The exact implementation of these guidelines, and their link with online real-world experiments is still largely an open question. In the next chapter, we explore these remaining questions for one of the alternatives: simulators.

2.7.2 Additional thoughts

A correct and appropriate evaluation setup is critical for research. Unfortunately, it became apparent after we had listed the limitations of existing alternatives that there is no silver bullet. Instead, it seems that the combination of multiple types of evaluation, along with being generally careful with claims and expected results, is necessary going forward.

Online evaluation is certainly necessary at some point, and I see the offline evaluation methods described in this chapter more as a way for practitioners to avoid wasting time and resources on obviously wrong models. For researchers, I think this chapter is a stark reminder that we should be very careful with claiming improvements based on a single offline metric and dataset, and that making progress in a research topic as complex and hard to evaluate as recommendation requires a well-rounded assessment for multiple perspectives and reproducibility efforts, rather than blindly maximizing a single metric. See for instance (Ferrari Dacrema et al., 2019) for a large empirical review of progress in recommender systems.

I was happy to see this work being adopted by the community for evaluating new algorithms (Gao et al., 2023). Interesting research questions have also come up from this reflection. For example, Silva et al. (2024) studied why RL agents perform well on NIP benchmarks when they should not bring anything in theory.

3

SARDINE: A SIMULATOR FOR AUTOMATED RECOMMENDATION IN DYNAMIC AND INTERACTIVE ENVIRONMENTS

One of the possible alternatives for next-item prediction that we highlighted in the previous chapter is the use of simulators. I naturally wanted to experiment and find such a simulator, but among the existing ones, none really fitted the way I wished to use them. Either they did not incorporate any of the mechanisms that make recommender systems dynamic and interactive, or they were hardly interpretable and configurable or felt like black-boxes, where you essentially have to "trust" the simulator to be faithful to reality.

But it seems very unlikely to me that simulators will ever be realistic approximations of a real recommender systems, as that would require accurately modeling the inner workings of the human brain – and you could argue that if that ever happens, the problem will then be solved and the simulator will not be useful anymore. We therefore came up with the idea of proposing our own take at making a simulator, but we also explain in detail, using examples, how we think it can be used to investigate specific research questions.

This chapter is based on the following publication: **Romain Deffayet**, Thibaut Thonet, Dongyoon Hwang, Vassilissa Lehoux, Jean-Michel Renders, and Maarten de Rijke. 2024. SARDINE: A Simulator for Automated Recommendation in

Dynamic and Interactive Environments. In *ACM Transactions on Information Systems (TORS), "Just accepted"*.

3.1 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 (Melville and Sindhwani, 2010). In other words, they need to learn the semantic information (Sequoiah-Grayson and Floridi, 2022) 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 well as for content providers and other potential stakeholders. 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 (McNee et al., 2006b; Jannach et al., 2016) and that often view recommendation as a dynamic and interactive task (He et al., 2016). First, recommender systems are often trained from user interaction data, either in an online (Silva et al., 2022) or offline (Gupta et al., 2023) fashion. As a result, recommender systems must learn to deal with noisy user feedback (Wang et al., 2022), limited knowledge about new users in the cold start scenario (Knyazev and Oosterhuis, 2023), as well as potential biases in user behavior that may impact the training data (Gupta et al., 2023). Second, the items consumed by a user may have an effect on the user state (Anderson et al., 2020; Deffayet et al., 2023c; Gao et al., 2023). 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 (Anderson et al., 2020). Third, exogenous factors may change

the value of items and the preferences of users, yielding an ever-changing dynamic environment (Huleihel et al., 2021).

3.1.1 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 (Agrawal and Goyal, 2013; Li et al., 2010), reinforcement learning (RL) (Chen et al., 2019b; Deffayet et al., 2023c; Gao et al., 2023), active learning (Rubens et al., 2015), counterfactual learning-to-rank (Joachims et al., 2017; Gupta et al., 2023), and click modeling (Craswell et al., 2008; Chuklin et al., 2015; Deffayet et al., 2023a). These approaches are trained from user data, and it has been shown that they should not be evaluated solely on accuracy-centric benchmarks (Deffayet et al., 2022; Jannach et al., 2016; Sun, 2023) as these miss the potential benefits brought by beyond-accuracy methods.

While online evaluation (Zangerle and Bauer, 2022; Sato, 2021) remains a gold standard – when done right (Jeunen, 2023) – 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 (Deffayet et al., 2022; Swaminathan et al., 2017; Kiyohara et al., 2023) 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 environmental 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 3.1.1 – 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.

3.1.2 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 (Deffayet et al., 2023c; Gao et al., 2023; Chen et al., 2019b; Xin et al., 2022). 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 (Chen et al., 2019b; Gao et al., 2023; Xin et al., 2022), counterfactual learning-to-rank (Gupta et al., 2023; Joachims et al., 2017), or click modeling (Deffayet et al., 2023a; Chuklin et al., 2015). 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 (Knyazev and Oosterhuis, 2023; Oosterhuis and de Rijke, 2021a). 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 (Ie et al., 2019b; Chen et al., 2019b; Swaminathan et al., 2017), including methods that improve on the efficiency of slate recommender systems (Oosterhuis, 2021a; Sakhi et al., 2023). It is possible to train slate recommender systems on all our `SlateTopK` and `SlateRerank` environments.

3.1.3 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 3.1.2, 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 (Towers et al., 2023) 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 existing approaches and baselines, and (iii) uncover 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 described in the previous section; 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

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.

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 chapter is organized as follows. We formally define the recommendation problem of interest in Section 3.2. We then describe the technical details of the SARDINE simulator in Section 3.3. Section 3.4 covers the details about our experimental setup, which includes the description of the SARDINE environments tested in our experiments as well as the compared approaches. The experimental results are presented and discussed in Section 3.5. Finally, we compare our proposed SARDINE to existing recommendation simulators in Section 3.6, and conclude the chapter in Section 3.7.

3.2 PROBLEM DEFINITION

The problem studied in this chapter 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.

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.

- 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 chapter, 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.

3.3 SIMULATOR

In this section, we detail the components of our 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 (Hidasi et al., 2016).

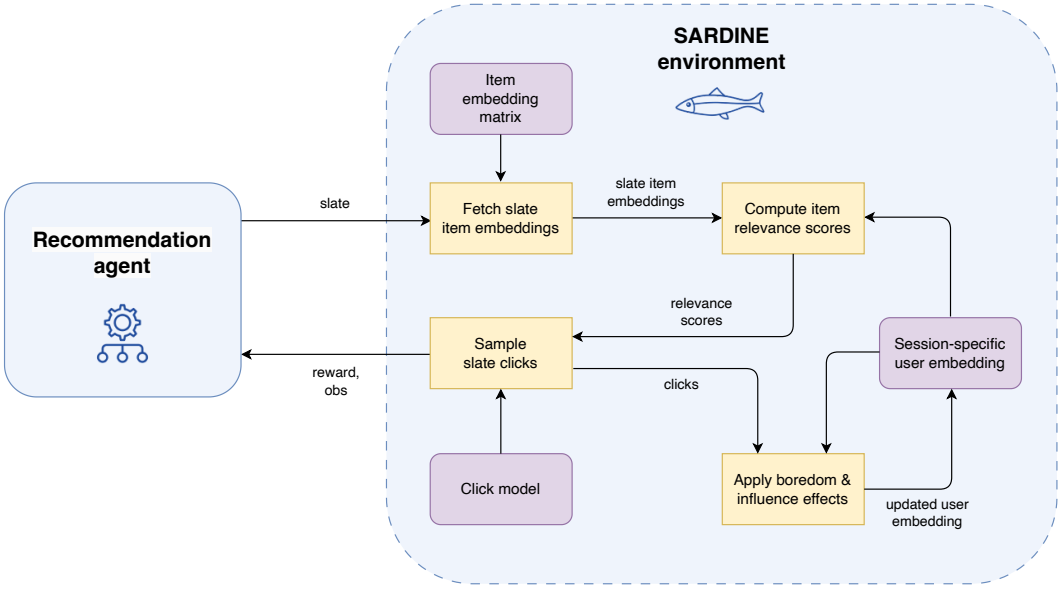


Figure 3.1: Diagram summarizing the different components of the proposed SARDINE simulator, and its interaction with the recommendation agent.

First, our simulator is initialized by forming synthetic embeddings for the set of recommendable items (Section 3.3.1). Then, each user session is generated by following these successive steps:

1. Sample a user embedding for the current session's user (Section 3.3.1);
2. Provide an initial recommendation (Section 3.3.2) or prompt the agent to recommend a slate to the user;
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 3.3.3);
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 3.3.4);
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 3.3.5. 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,

Table 3.1: 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.
$\mathcal{O}$	Whether the state is fully or partially observable.

and targeting various research outcomes. We introduce these environments in Section 3.4.1.

Fig. 3.1 illustrates the different components of our simulator and its interactions with the recommendation agent. In Table 3.1 we additionally provide a description for the hyperparameters of the simulator, which are further defined in the remainder of this section.

3.3.1 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:

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.

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 $\mathbf{e}_{i,j} \sim \text{Unif}([0, 1])$;

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).

3.3.2 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 (Chaimalas et al., 2023), prior user profile (Chamsi Abu Quba et al., 2014) as well as user metadata (Majumdar and Jain, 2017). We leave the investigation of alternative initial recommendations for future work.

3.3.3 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:

$$\text{rel}(i, u) = \mathbf{e}_i^T \mathbf{e}_u. \tag{3.1}$$

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:

$$A_{u,i} = \alpha \cdot \sigma(\text{rel}(i, u)) \text{ where } \sigma(x) = \frac{1}{1 + \exp(-\lambda(x - \mu))}. \tag{3.2}$$

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. 3.2 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 $\text{Bern}(A_{u,i} \times E_r)$.

3.3.4 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 (Lee et al., 2022a). 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 rate (Gao et al., 2023; Anderson et al., 2020; Mehrotra, 2021). The first mechanism we define is referred to as *boredom* and intuitively reflects the fact that a user may become

5 The notion of boredom introduced in this simulator is further detailed in Section 3.3.4.

less interested in consuming content (i.e., clicking on items) when the items recommended in successive slates are too similar, similarly to (Gao et al., 2023; Deffayet et al., 2023c). 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., Cinus et al., 2022). 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 $\tau_b \leq n_b$ 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.

3.3.5 *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 $\mathcal{O}$. 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 3.4

and 3.5), 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).

3.4 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 3.1.2. 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 3.1.2, and (iii) to demonstrate the simulator's utility for recommendation research by uncovering some novel insights about these methods.

First, Section 3.4.1 introduces nine recommendation environments instantiated from our simulator, that we use in our experiments. Then, Section 3.4.2 describes the recommendation agents we seek to compare on the environments. Finally, Section 3.4.3 summarizes the simulator hyperparameters adopted by

the different environments, as well as the agent hyperparameters used in our experiments.

3.4.1 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 3.3. They are characterized along six dimensions, which are directly linked to the research topics defined in Section 3.1.2:

(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 3.3.3 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 3.3.5 — **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., $n_{\mathcal{I}} = S$) and the recommendation agent has to find the best permutation of those items[6] — **RT2**.

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.

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 (Deffayet et al., 2022).
- `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 is derived from `SlateTopK-BoredInf` and 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`: To create this environment, we build on top of `SlateTopK-PartialObs` and gradually 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 hyperparameter 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 3.1.2: the inclusion of multi-step mechanisms (in `SingleItem-BoredInf`, `SlateTopK-Bored`, `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 3.4.3.

As a side note, in Section 3.3.4, 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.

3.4.2 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 3.4.1. 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 3.3.3, 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 (Chen et al., 2019b) 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.

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.

- **SAC + Top-K:** This method was introduced in (Deffayet et al., 2023c) as a simple yet strong baseline for slate recommendation. It relies on a soft actor-critic (SAC) (Haarnoja et al., 2018) 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 (Deffayet et al., 2023c), 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 (Liu et al., 2023) 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 (Lillicrap et al., 2016) 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 3.3.1). This constitutes an advantage for the agents which explicitly use item embeddings in their method, namely, Greedy

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 3.2: 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 3.1. 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	$\mathcal{O}$
`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

Table 3.3: 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

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 3.5.2).

3.4.3 Hyperparameter setting

The hyperparameters used for each of the environments introduced in Section 3.4.1 are detailed in Table 3.2. The hyperparameter values were chosen to reflect the environment-specific characteristics that we highlighted in Table 3.3.

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 a partial observable state (`SingleItem-PartialObs`, `SlateTopK-PartialObs`, `SlateTopK-Uncertain`), we used two types of state encoders commonly used in RL-based recommender systems (Huang et al., 2022a): 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

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

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).

3.4.4 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 3.3.1. 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 3.4.3) 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 therefore higher quality recommendations from the agent. On the five environments that include a notion of boredom (`SingleItem-BoredInf`, `SlateTopK-Bored`, `SlateTopK-BoredInf`, `SlateTopKPartialObs`, `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 3.3.4 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).

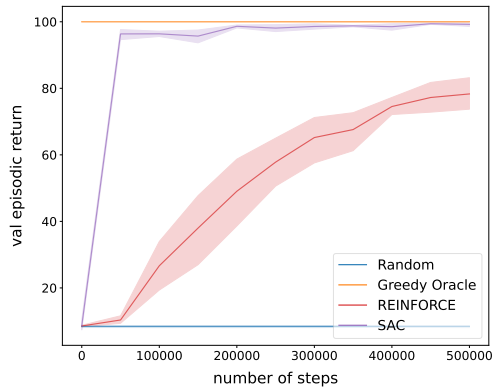
3.5 RESULTS

In this section, we describe the results of the experiments done on single item recommendation (Section 3.5.1), slate top-K recommendation (Section 3.5.2), and slate reranking (Section 3.5.3). 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.

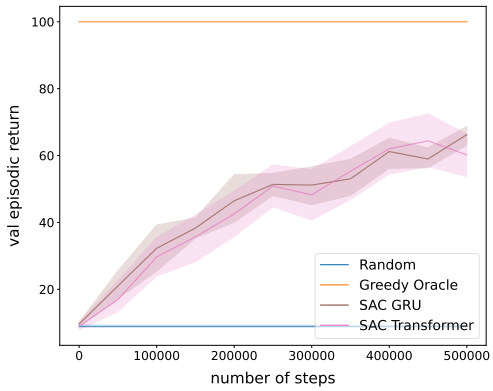
3.5.1 Experiments on single item recommendation

We performed experiments on three `SingleItem` environments whose characteristics are recalled below (see Section 3.4.1 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. 3.2 and discussed below.

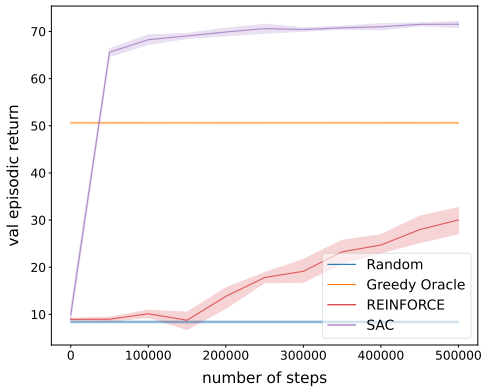
`SingleItem-Static`. The validation return over the different training steps on `SingleItem-Static` is plotted in Fig. 3.2a. In this experiment we compared SAC and REINFORCE agents against the Greedy Oracle and Random baselines. In this specific setting, the Greedy Oracle is by design optimal due to 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



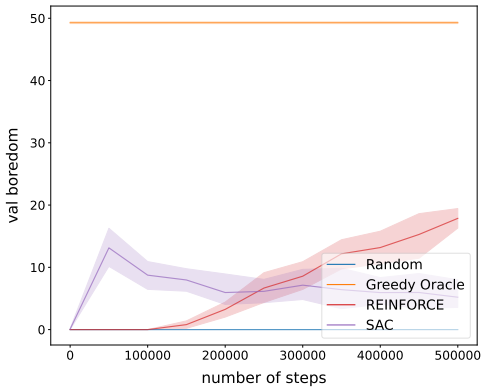
(a) Return (↑) on `SingleItem-Static`



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



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



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

Figure 3.2: Results on the `SingleItem-Static` (3.2a), `SingleItem-PartialObs` (3.2b), and `SingleItem-BoredInf` (3.2c, 3.2d) 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.

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 challenging `SingleItem-BoredInf` environment which includes boredom and influence mechanisms. The results according to the return and boredom metrics are illustrated in Fig. 3.2c and 3.2d, 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 (Deffayet et al., 2022). 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 test in combination with a SAC agent as it obtained convincing results on `SingleItem-Static`. The results are shown in Fig. 3.2b. 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.

3.5.2 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 3.4.1). `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. 3.3, 3.4, 3.5, and 3.6.

`SlateTopK-Bored`. The results on the `SlateTopK-Bored` environment are plotted in Fig. 3.3a (for the return) and Fig. 3.3b (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 3.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. 3.3a and Fig. 3.3b, but we replace the ideal item embeddings used by default in our experiments with item embeddings learned by matrix

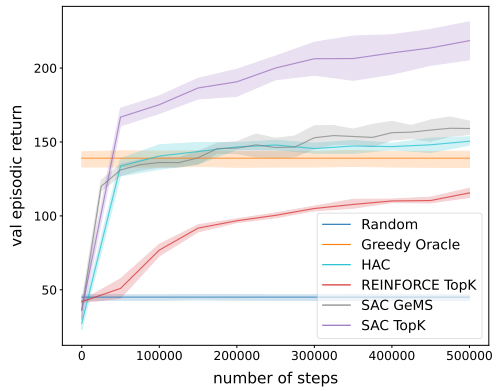
factorization (MF).[10] We report the results in Fig. 3.3c and Fig. 3.3d for the return and boredom metrics, respectively. We observe that changing from ideal to MF embeddings does have a drastic effect on the recommendation performance (measured by the return metric) of SAC + Top-K, which degraded to 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 3.B.

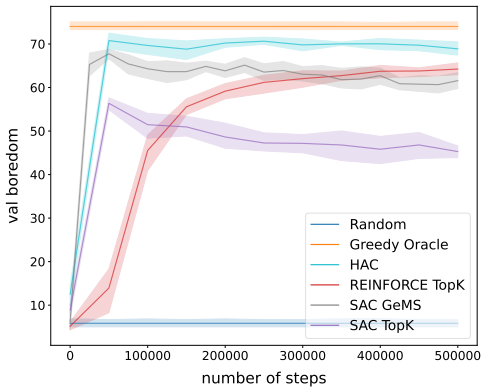
`SlateTopK-BoredInf`. On the `SlateTopK-BoredInf` environment, which adds an influence mechanism to `SlateTopK-Bored` with ideal embeddings, we observe similar trends as this latter environment. The return and boredom results are given in Fig. 3.4a and Fig. 3.4b, respectively. One notable difference 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. 3.5a (for return) and Fig. 3.6a (for boredom). Given 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 is in line with previous findings on state encoders for RL-based recommendation (Huang et al., 2022a). However, both SAC + Top-K variants fail to beat the Greedy Oracle baseline, highlighting the difficulty of

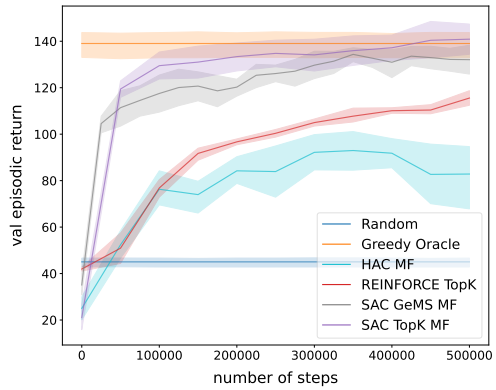
10 To obtain these embeddings, we used the dataset generated for SAC + GeMS pretraining (described in Section 3.4.2) to train a matrix factorization model with an embedding dimension of 10.



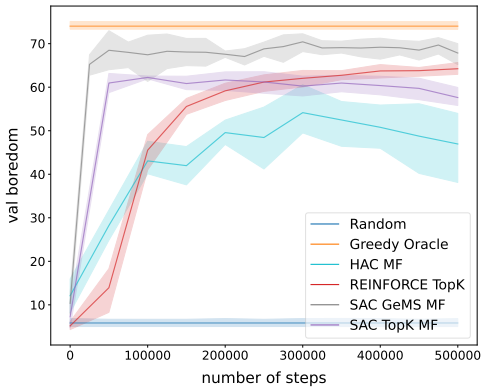
(a) Return (↑) on `SlateTopK-Bored` (Ideal)



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



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

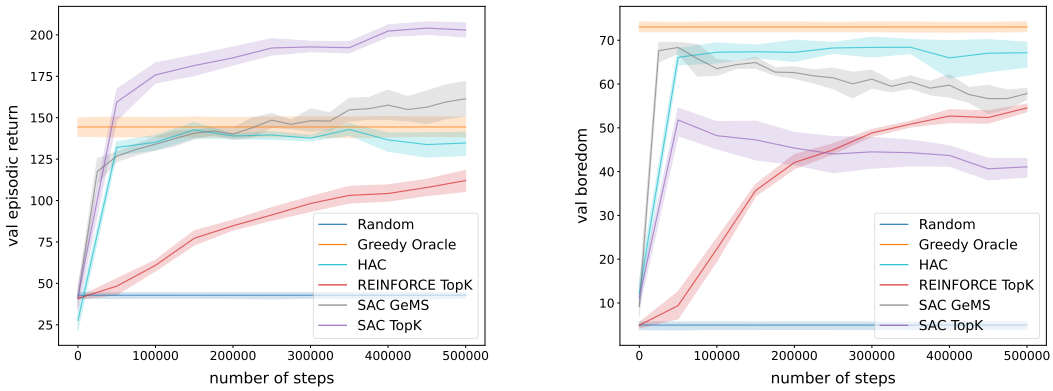


(d) Boredom (↓) on `SlateTopK-Bored` (MF)

Figure 3.3: Results on the `SlateTopK-Bored` environment with default, ideal item embeddings (3.3a, 3.3b) and with matrix factorization item embeddings (3.3c, 3.3d). 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).

this environment and showing that additional efforts on the agent and/or state encoder might be needed to achieve high-quality recommendation.

`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 original setting of `SlateTopK-PartialObs` with low uncertainty ($\lambda = 100$) to different

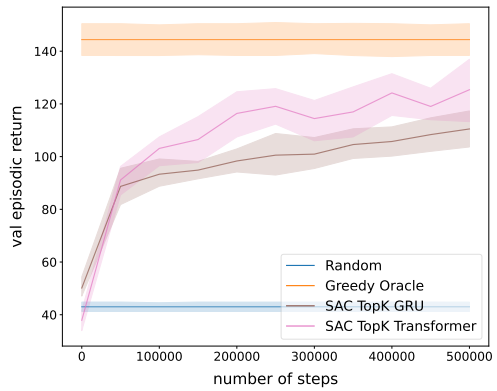


(a) Return (↑) **(b)** Boredom (↓)

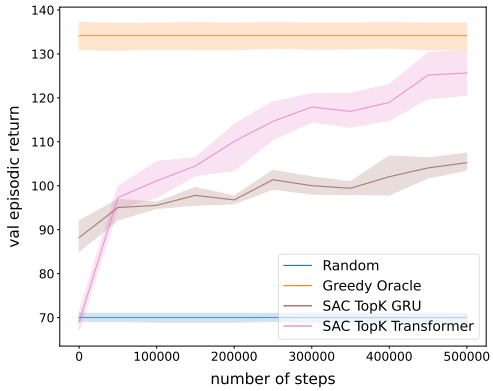
Figure 3.4: Results on the `SlateTopK-BoredInf` environment. The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

`SlateTopK-Uncertain` 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. 3.5 and Fig. 3.6, respectively. Comparing the return on `SlateTopK-PartialObs` (Fig. 3.5a) to `SlateTopK-Uncertain10` (Fig. 3.5b), 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. 3.5c), 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. 3.6a, Fig. 3.6b, and Fig. 3.6c. When the

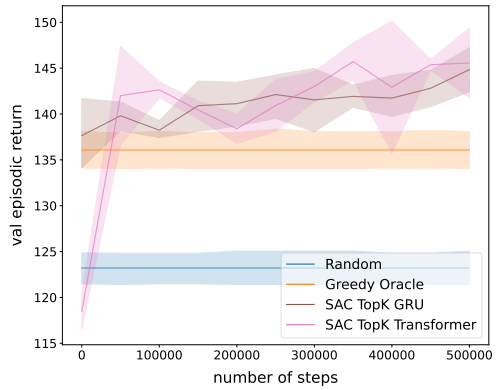
uncertainty level is further increased in `SlateTopK-Uncertain2`, we observe that the environment rewards random recommendations more than the accurate recommendations from the Greedy Oracle, as shown in Fig. 3.5d. This is, again, explained by the fact that less relevant items lead to a click probability 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.



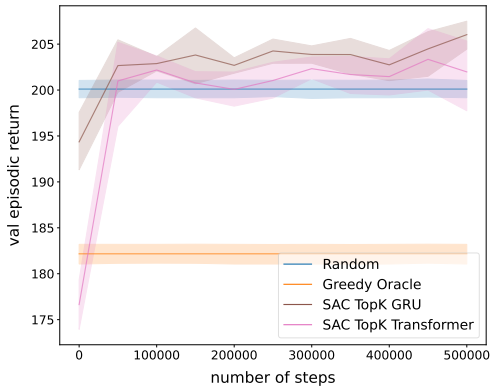
(a) `SlateTopK-PartialObs`, low uncert.



(b) `SlateTopK-Uncertain`, medium uncert.

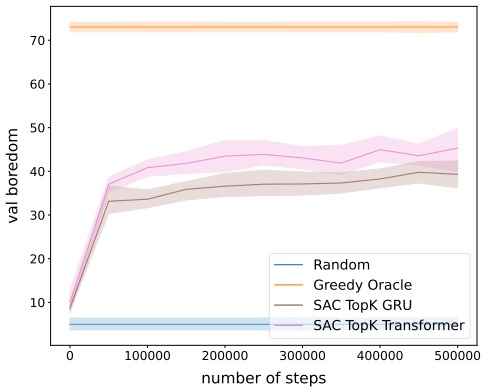


(c) `SlateTopK-Uncertain`, high uncert.

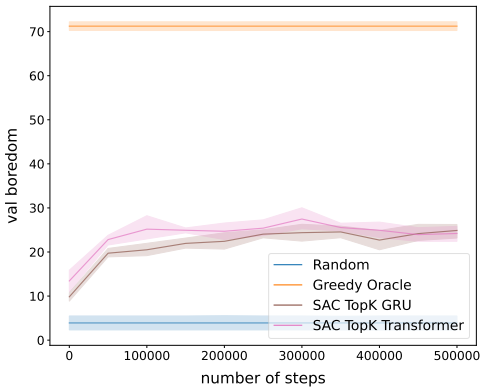


(d) `SlateTopK-Uncertain`, very high uncert.

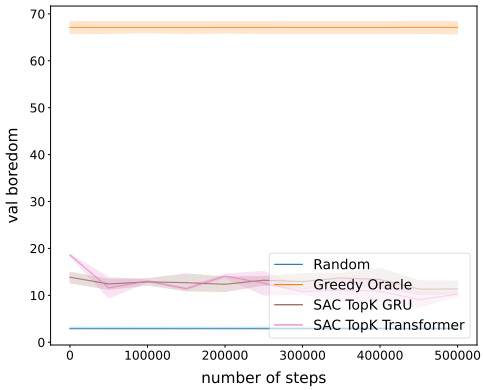
Figure 3.5: Results in terms of return (↑) on the `SlateTopK-PartialObs` (3.5a) and `SlateTopK-Uncertain` (3.5b, 3.5c, 3.5d) environments. The click uncertainty degree varies from low (3.5a), medium (3.5b), high (3.5c) to very high (3.5d), corresponding to a scale hyperparameter λ in the relevance function equal to 100, 10, 5, and 2, respectively (see Section 3.3.3 for more details). The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.



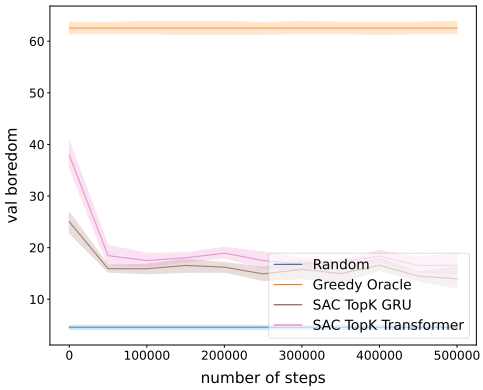
(a) `SlateTopK-PartialObs`, low uncert.



(b) `SlateTopK-Uncertain`, medium uncert.



(c) `SlateTopK-Uncertain`, high uncert.



(d) `SlateTopK-Uncertain`, very high uncert.

Figure 3.6: Results in terms of boredom (↓) on the `SlateTopK-PartialObs` (3.6a) and `SlateTopK-Uncertain` (3.6b, 3.6c, 3.6d) environments. The click uncertainty degree varies from low (3.6a), medium (3.6b), high (3.6c) to very high (3.6d), corresponding to a scale hyperparameter λ in the relevance function equal to 100, 10, 5, and 2, respectively (see Section 3.3.3 for more details). The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

3.5.3 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 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 3.4.

Table 3.4: 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

`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 3.4 that an online-trained SAC + Top-K, as in Section 3.5.2, even with full observability of user state and ideal item embeddings, does not beat the greedy oracle.

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 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 (Deffayet et al., 2023a), 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 3.4 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.

Table 3.5: Comparison of the proposed SARDINE to existing recommendation simulators. ✓ indicates that the research topic is addressed by the simulator and $\sim$ that it is partially addressed. Our assessment of whether the simulators fulfill the specifications is graded according to $\{ -, \pm, + \}$. Overall, we find that only RecSim (Ie et al., 2019a) addresses the research agenda that SARDINE targets, but that it does not fulfill our specifications for such a simulator.

	Research agenda 3.1.2				**Specifications 3.1.1**		
Simulator	RT1	RT2	RT3	RT4	Interpretability	Effect isolation	Configurability
RecoGym		$\sim$	✓		$\pm$	$+$	$+$
MARS-Gym					$-$	$-$	$\pm$
RL4RS	$\sim$	$\sim$		✓	$-$	$-$	$-$
RecSim	✓	✓	$\sim$	$\sim$	$\pm$	$-$	$\pm$
Virtual-TB	✓		✓		$-$	$-$	$-$
SOFA		✓			$+$	$\pm$	$+$
OBP		✓			$+$	$+$	$+$
SARDINE	✓	✓	✓	✓	$+$	$+$	$+$

3.6 RELATED WORK

In this section we highlight how our work differs from previously published simulators. Considering the research agenda we defined in Section 3.1.2 as well as our target specifications 3.1.1, we draw a comparison of existing simulators, along with our proposed SARDINE, in Table 3.5. 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 (Rohde et al., 2018) 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 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 (Santana et al., 2020) 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 (Wang et al., 2023a) 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 depen-

dencies. However, the simulator is opaque and hardly tunable, and thus does not satisfy our specifications for a research-oriented simulator.

RecSim (Ie et al., 2019a) 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 (Shi et al., 2019) 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 (Huang et al., 2020) 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 (Saito et al., 2021) 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.

3.7 CONCLUSION

Summary. In this chapter, 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 – span-

ning 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 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 (Deffayet et al., 2023c)) 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 (Deffayet et al., 2023c) approach seemed to be more robust overall to the item embedding quality. The recent hyper-actor critic (HAC) approach (Liu et al., 2023) was the most impacted by the quality of item embeddings in the studied settings. Moreover, we found that this approach was

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

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

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 (Huang et al., 2022a) 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 3.1.2). For instance, the performance of agents when the environment is non-stationary (e.g., due to changes in the world) is still largely unknown (Padakandla et al., 2020), and could be investigated in SARDINE. Similarly, reaching the best possible policy in a limited number of deployments, a task known as deployment-efficiency (Matsushima et al., 2020), as well as continual learning (Wang et al., 2023b), 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.

3.8 REFLECTIONS ON THE CHAPTER

3.8.1 Research outcomes

We proposed one of the many possible tools that can help answer my first research question:

Research Question 1. *How can we evaluate recommender systems in a way that accounts for their dynamic and interactive nature?*

While they bring interesting insights and fast development cycles, simulators are clearly lacking real-world validity. Our use of users and items from the Webtoon platform gives a way to make simulators more plausible representation of reality, but even doing so cannot bring the gap between simulated and real user behavior. But what if we relax the requirements and, rather than trying to build accurate representations of real humans, merely aim to predict the online performance of models ? This is the purpose of my second research question, that the next part is dedicated to exploring:

Research Question 2. *Can we predict in a fully offline manner the performance of models learned on biased data?*

3.8.2 Additional thoughts

It is clear from the experiments of this chapter that the recommendation task is very hard: even with perfect knowledge of the user preference and item content, and even in this relatively simple scenario, interactive slate recommendation is far from solved. This was not obvious to me before working on SARDINE. That means that the current limitations of recommender systems are not merely due to a lack of understanding of users and items, but also to limited reasoning capabilities, at least for out-of-the-box RL agents. Going forward, I believe RL-based recommender systems will need to be improved

on this aspect, and that in the more realistic scenario where the user and item representations must be learnt, semantic representations that are designed to help the agent reason will be a necessity.

CHAPTER APPENDIX

3.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 3.5.2, takes around 40 minutes on a single NVIDIA A100 GPU.

3.B WEBTOON EXPERIMENT

3.B.1 Environment description

The environments introduced in Section 3.4 and experimented on in Section 3.5 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 3.B.1. Differ-

13 `https://www.webtoons.com/en` (Item catalog accessed in January 2022).

Table 3.B.1: Value of the simulator hyperparameters for the `WebtoonSlateTopK-Bored` environment. The description of the hyperparameters' meaning and role is detailed in Table 3.1.

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

ently from `SlateTopK-Bored`, `WebtoonSlateTopK-Bored` includes 669 items and 16 topics.

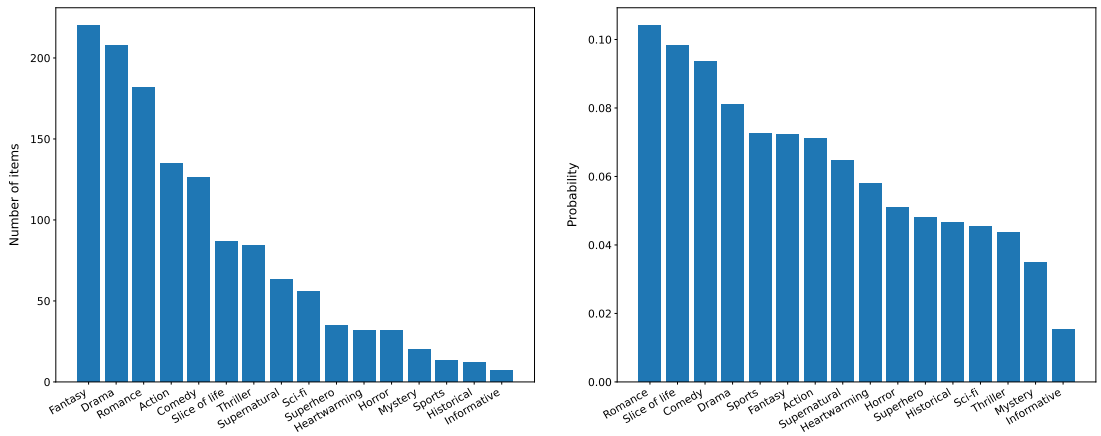
User and item embeddings. In this experiment, we consider that we only have access to an item catalog, which does not directly include embeddings. Therefore, item and user embeddings need to be generated under the constraints imposed by the catalog (e.g., item-topic assignments), leading to a semi-synthetic setting. To obtain item and user embeddings based on the Webtoon catalog, we slightly changed the generation process described in Section 3.3.1. 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. 3.B.1a. This figure highlights a skewed distribution, with a large share of items pertaining to *Fantasy*, *Drama*, or *Romance* 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 3.3.1 to reflect the fact that topics may not be uniformly popular among users. More specifically, instead of being sampled from $\text{Unif}(\mathcal{T})$, the topics of interest for a user u (denoted as $\mathcal{T}_u = \{T_{u,1}, \dots, T_{u,n_{\mathcal{T}_u}}\} \subset \mathcal{T}$) 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:

$$p_{\mathcal{T}}(j) = \frac{\frac{1}{|\mathcal{I}_j|}\sum_{i\in\mathcal{I}_j}\#\text{likes}(i)}{\sum_{j'\in\mathcal{T}}\frac{1}{|\mathcal{I}_{j'}|}\sum_{i\in\mathcal{I}_{j'}}\#\text{likes}(i)}, \tag{3.3}$$

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. 3.B.1b, which highlights as well the skewed nature of user-topic preferences in the `WebtoonSlateTopK-Bored` environment.



(a) Item distribution across topics

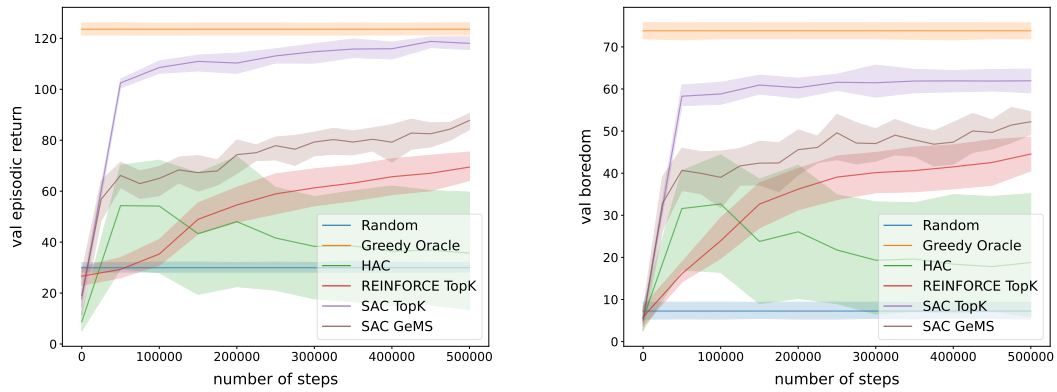
(b) Topic prior for user embeddings

Figure 3.B.1: Properties of Webtoon items and topics (i.e., Webtoon categories). Fig. 3.B.1a depicts the number of items attributed to each topic, based on the actual item-category assignments in the Webtoon catalog. Fig. 3.B.1a 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.

3.B.2 Results

Following the experimental setup and hyperparameters used for the experiments on `SlateTopK-Bored` (Ideal) (the results of which are described in Section 3.5.2, and illustrated in Fig. 3.3a and Fig. 3.3b), we compared the same methods on the `WebtoonSlateTopK-Bored` environment. The results are shown in Fig. 3.B.2, with the return in Fig. 3.B.2a and the boredom in Fig. 3.B.2b 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 to other methods. Overall, this experiment demonstrates that deriving environments with realistic, long-tail distributions provides interesting and challenging use-cases in SARDINE.



(a) Return (↑)

(b) Boredom (↓)

Figure 3.B.2: Results on the `WebtoonSlateTopK-Bored` environment. The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

3.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 3.3.4. 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

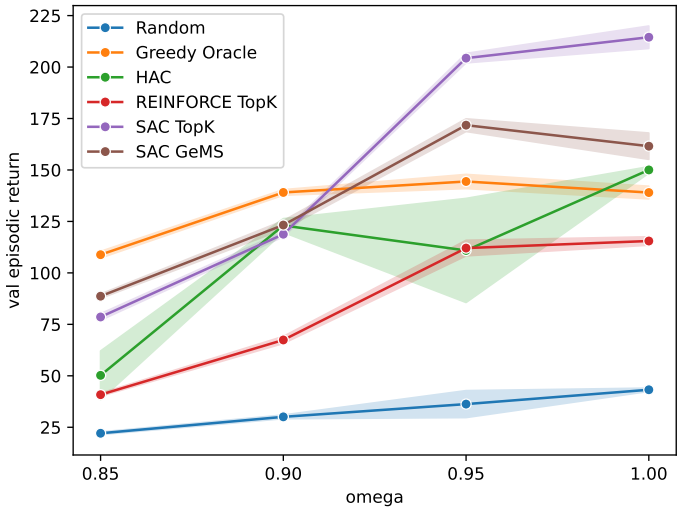


Figure 3.B.3: Validation return after 500,000 training steps on the `SlateTopK-BoredInf` environment at different levels of influence by the clicked items (the lower ω, the higher the influence of clicked items on user preference). The colored envelope surrounding lines indicates the 95% confidence interval around the mean computed from 5 seeded runs.

`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 3.5.2, we train all models for 500,000 steps, with five different random seeds. In Figure 3.B.3, 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.

3.D ITEM SCORES

In Figure 3.D.1 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"]`.

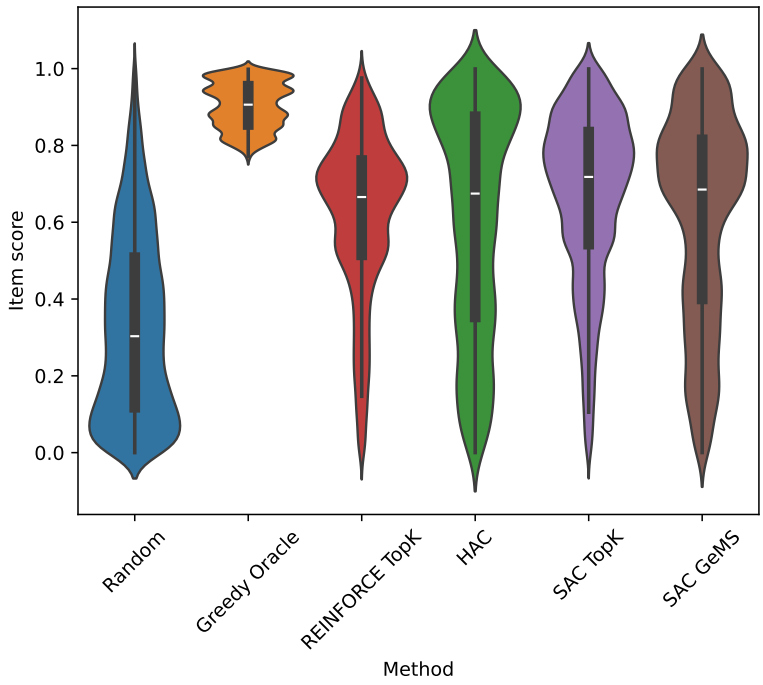


Figure 3.D.1: Distribution of the relevance score of items recommended by different methods after training for 500,000 steps on `SlateTopK-Bored`. The higher the score, the higher the click probability. See Section 3.3.3 for how the relevance score is computed within the simulator.

Part II

Estimation and Correction of Biases in Learning-to-Rank

4

EVALUATING THE ROBUSTNESS OF CLICK MODELS TO POLICY DISTRIBUTIONAL SHIFT

A major obstacle to learning recommender systems from data is biased data. It is well known that reinforcement learning agents are especially sensitive to selection bias: agents that use dynamic programming have a tendency to overestimate the value of actions that were not taken in the data, and therefore perform poorly at deployment time. The Offline RL community tries to overcome this challenge (Levine et al., 2020). In information retrieval, the selection bias problem is also well known, but it is often coupled with other types of data biases that misrepresent the true value of items: position bias (Joachims et al., 2007), trust bias (Agarwal et al., 2019), outlier bias (Sarvi et al., 2023), ... Even a perfectly calibrated offline RL agent would wrongly estimate the future expected returns under these biases, and the resulting agent would likely perform very poorly.

These biases have been studied for a long time in the information retrieval community. Most approaches can be grouped in two categories: those that postulate a certain type of bias and strive to find effective and efficient way to correct for the biases (they often identify with the term *unbiased learning-to-rank*), and those that propose and compare new bias models and the assumptions on user behavior that these models implement (they are often grouped under the umbrella term *click modeling*). In order to answer research question 3, i.e., to understand when we need assumptions and how to choose the correct ones, I chose to study biases from the click modeling perspective.

Therefore, in this chapter and the next one, we investigate the impact of the choice of click model on different downstream tasks, and how to select the right one. We perform our analysis in the context of web search, rather than recommendation, as biases are usually of similar nature, but more real-world datasets are available in this scenario and it allows us to isolate the effect of click modeling while ignoring the challenges that come with personalisation.

This chapter is based on the following publication: **Romain Deffayet**, Jean-Michel Renders, and Maarten de Rijke. 2023. Evaluating the Robustness of Click Models to Policy Distributional Shift. In *ACM Transactions on Information Systems (TOIS) 41, 4, Article 84 (October 2023)*.

4.1 INTRODUCTION

Search engines rank items according to their relevance to users, given the query they enter as well as the user and search context. To do so, many learning-to-rank (L2R) approaches leverage click logs, due to their abundance and the realistic settings they result from (Joachims et al., 2007; Chen et al., 2019a). However, clicks and skips are not direct signals of relevance. They emerge from interactions of users with the search system, meaning that the data is biased by the policy in place in the search system during data collection, often called the *logging policy* (Jagerman et al., 2019). Different sources of intrinsic bias induced by the logging policy have been identified, such as position bias (Joachims et al., 2005) (the position of documents in the search result page (SERP) influences their click likelihood) or trust bias (Vardasbi et al., 2020a) (users are likely to trust the system to return appropriate results and to click on top-returned documents regardless of their actual relevance).

Therefore, in order to improve L2R models, it is crucial to gain a better understanding of user behavior on search engines by using *click models* (Chuklin et al., 2015). They model user behavior by learning to generate realistic click sequences, while maintaining an interpretable structure that makes it possible to identify and disentangle the underlying causal factors of user behavior (relevance, examination, . . ., often in the form of latent variables of the model). This constrained structure allows one to extract a more accurate, unbiased relevance signal from the logs, and to derive models that do not replicate the biases seen in the logged data. The typical debiasing workflow with click models consists

of (i) collecting biased click logs from the search engine, (ii) training a click model on this dataset, and (iii) using the resulting (hopefully) unbiased model to improve the policy of the search engine.

4.1.1 Evaluating click models

Click models are often evaluated on two tasks (Grotov et al., 2015): (i) *click prediction*, typically measured by the perplexity (PPL) (Dupret and Piwowarski, 2008) obtained when predicting clicks on a separate test set, and (ii) *relevance estimation*, typically measured by the normalized discounted cumulative gain (nDCG) (Järvelin and Kekäläinen, 2002) of rankings produced by the ordering of relevance scores recovered by the click model compared to those based on relevance grades given by human annotators. The click prediction metrics are usually computed on a test set collected using the same ranking policy as the training set, thus presenting the same intrinsic biases. Consequently, this type of evaluation is able to quantify the goodness of fit of a click model to the distribution of the logged data, but cannot guarantee that this data has been effectively debiased by the click model. On the contrary, at first glance one could hypothesise that strong performance in the relevance estimation task ensures effective debiasing. After all, a high nDCG score should be an indication that the causal identification of underlying factors (relevance, examination, etc.) has been successful, which would mean that the resulting click model is unbiased with respect to the logging policy.

The central question that motivates this chapter is: *can the current evaluation practice for click models ensure that a click model is robust to changes in the ranking policy?* To make this question more precise, we introduce the notion of policy distributional shift.

4.1.2 Policy distributional shift and click models

Policy distributional shift (PDS) occurs when we ask the click model to predict clicks on rankings produced by a policy different from the logging policy, a task known as *off-policy evaluation* (OPE) (Swaminathan et al., 2017). PDS is a type of covariate shift, as it modifies the input distribution of the click model at test time. Under PDS, spurious correlations replicated from the data can have a detrimental impact on the performance. Therefore, robustness to PDS

requires successful causal identification of the underlying factors, i.e., effective debiasing of the logged data.

Policy distributional shift matters. To illustrate its real-world impact, we list five common downstream tasks for click models and group them into three categories:

Group ①: Latent variable extraction

- Label debiasing: extracting unbiased relevance scores from the model either to directly derive a policy or to serve as labels or features in a supervised L2R approach (Chapelle and Zhang, 2009; Vardasbi et al., 2020b);
- Counterfactual L2R: extracting propensity scores to reweight individual samples in a L2R loss (Oosterhuis and de Rijke, 2020a; Vardasbi et al., 2020b).

Group ②: Off-policy evaluation

- Click-through rate (CTR) prediction: deriving CTR estimates of specific rankings for optimizing ad placement and pricing (McMahan et al., 2013; Chen and Yan, 2012).

Group ③: Inverse model optimisation

- Fair ranking: extracting both relevance and exposure scores to measure and control the utility and fairness of fair re-ranking algorithms (Singh and Joachims, 2018);
- Offline Bandits and RL: using the model as an interaction predictor for training bandits or RL policies (Huang et al., 2020; Zhang et al., 2022).

Tasks in Group ① do not require OPE, so they are not subject to underperformance due to PDS. However, we show in Section 4.4 that existing offline evaluation protocols cannot guarantee that the relevance or propensity scores have been effectively debiased, which defeats the purpose of click models for such applications.

CTR prediction (②) consists in performing explicit OPE; as a consequence, it is sensitive to policy distributional shift. We show in Section 4.6.1 that existing offline evaluation protocols are not able to detect such sensitivity, and we investigate how click models compare to each other in this setting in Section 4.6.2.

The problem of PDS is especially critical for tasks in Group ③, where we seek to recover a high-performance policy through an optimisation process. Such tasks require (implicitly or explicitly) numerous instances of OPE to select the best-performing policy. This leads to a phenomenon known as the *optimiser's curse* where inaccuracies of the model are exploited by the optimisation algorithm, potentially leading to failure at inference time (Smith and Winkler, 2006; Jeunen and Goethals, 2021). We compare click models w.r.t. the performance of the policies recovered in such tasks in Section 4.6.3.

4.1.3 Research goal and findings

Our research goal is to assess whether the current evaluation practice for click models is able to detect a lack of robustness to policy distributional shift, i.e., whether the current offline metrics are good indicators of the performance of click models on downstream tasks involving PDS (②, ③). To address this question, we proceed as follows. We assess the robustness of four different *types* of click models (Craswell et al., 2008; Dupret and Piwowarski, 2008; Chapelle and Zhang, 2009; Borisov et al., 2016), ranging from older models based on probabilistic graphs to newer neural models, which all encode a different structural assumption in their architecture. To make the comparison fair, we instantiate them under a unified implementation based on modern tools such as neural networks and stochastic gradient descent. In other words, we wish to compare the robustness of the key assumptions encoded in each model, rather than specific implementation details. Moreover, to enrich the experiments, we add two additional click models corresponding to two additional structural assumptions.

The experimental setup we adopt to address our research goal involves an evaluation protocol simulating OPE and online deployment that allows us to assess the robustness of click models under policy distributional shift in a way that is as close as possible to practical use cases of click models. Specifically, we simulate the deployment of click models for two tasks: CTR prediction ② and Offline Bandits ③, respectively, in Sections 4.6.2 and 4.6.3. For both experiments, we introduce a wide range of semi-synthetic environments in which relevance labels are derived from a real-world dataset but the simulated user behavior and the ranking policy are controlled by us. *Our experiments show that click models exhibit largely different and sometimes unexpected behaviors*

when tested outside of their training distribution, and that downstream policies are affected by this lack of robustness.

The performance inside the simulator strongly depends on its design, and is thus not meant to be a reliable indicator of online performance, but it can provide insights into the inner working of click models and enable us to discriminate poorly robust ones. *It allows us to gain higher confidence that the ranking policies that we wish to derive, depending on the chosen downstream task, will behave as expected once deployed into the real system*: it is a way to mitigate the risks of deploying L2R models online. Moreover, evaluating click models in a wide range of simulated environments decreases the sensitivity to simulator design and allows us to identify trends in the results.

4.1.4 Contributions

Our main contributions are the following:

- We identify the problem of policy distributional shift (PDS) in the context of click models and their evaluation;
- We show in a simulated environment that existing evaluation protocols do not guarantee good robustness of click models to PDS; and
- We propose an evaluation protocol for assessing the robustness to PDS of click models and compare various click models in a range of semi-synthetic environments.

Related work is discussed in Section 4.2. In Section 4.3, we introduce the concepts and tools necessary to our analysis, notably the existing offline evaluation protocol and the models we study throughout the chapter. Section 4.4 provides a preliminary experiment on real-world datasets which indicates that the existing evaluation protocol suffers from important shortcomings. We therefore describe our augmented evaluation protocol and our experimental setup in Section 4.5. Finally, we instantiate this protocol in Section 4.6: we first provide counter-examples where the current evaluation protocol is unable to guarantee robustness to PDS (Section 4.6.1), and then perform a full comparative evaluation of click models in a wide range of environments for downstream tasks from Group ② in Section 4.6.2 and Group ③ in Section 4.6.3. An online appendix containing our code with reproduction instructions can be found at github.com/naver/dist-shift-click-models.

4.2 RELATED WORK

4.2.1 Off-Policy training and evaluation

Offline training and evaluation of search and recommendation systems has been extensively addressed in the literature (Ai et al., 2021; Li et al., 2018; Steck, 2010), because performing online testing with real users is very costly. Among the issues to be addressed when evaluating search and recommendation systems offline, mitigating the bias induced by the policy used for data collection in learning-to-rank (L2R) has been tackled in numerous previous studies (Oosterhuis and de Rijke, 2020a; Swaminathan et al., 2017; McInerney et al., 2020; Li et al., 2018), usually under the umbrella terms *Off-Policy Evaluation* or *Counterfactual L2R*.

In these approaches, the experimental setup is usually as follows: we aim to learn from click logs that have been generated by a logging policy. Since this logging policy is usually not uniformly random, certain documents are more likely to be clicked than they would be under a different policy. Therefore, a correction is applied to de-bias the observed clicks (Joachims et al., 2017). In domain-agnostic off-policy evaluation (Swaminathan et al., 2017; McInerney et al., 2020), the correction corresponds to the probability of the document being placed at the rank at which we found it, by assuming a certain structure for the logging policy. These approaches are domain-agnostic in the sense that they do not leverage the specific properties of user behavior on search and recommendation sytems. Conversely, in inverse propensity scoring (IPS) (Joachims et al., 2017; Oosterhuis and de Rijke, 2020a), one assumes a certain user click model and reweighs each observed document by its probability of being examined by the user under the logging policy. Vardasbi et al. (2020a) and Oosterhuis and de Rijke (2021b) also model trust bias in order to improve the relevance estimates found by the counterfactual estimator. While certain studies compute the off-policy corrections with online swapping interventions (Joachims et al., 2017) or with eye-tracking experiments (Joachims et al., 2007), in this study we focus on the fully observational training of user click models, directly on the logs, as used in (Oosterhuis and de Rijke, 2020a; Vardasbi et al., 2020a; Zhang et al., 2022).

Regardless of the method used for propensity estimation, one must ensure the validity of the off-policy correction. To do so, most of the existing click model and

counterfactual L2R literature evaluates the de-biasing capabilities of learned models by using relevance labels provided by human annotators (either on real-world or semi-synthetic datasets) (Vardasbi et al., 2020a; Oosterhuis and de Rijke, 2020a; Joachims et al., 2017; Craswell et al., 2008; Borisov et al., 2016). However, as we explained in the introduction, this setting does not cover all possible use cases of click models and most importantly does not evaluate what would happen under policy distributional shift, i.e., in many practical scenarios (see task groups ② and ③ in the introduction). Swaminathan et al. (2017) and McInerney et al. (2020) evaluate the performance of their respective estimator on a simulated CTR prediction task with a few different policies, which ensures a certain degree of robustness to distributional shift for this particular task. However, ensuring that type of robustness is especially critical in applications such as fairness-constrained utility maximisation (Singh and Joachims, 2018) or RL-based L2R (Kveton et al., 2015; Zhang et al., 2022), in which the optimisation process will exploit any inaccuracies of the click model if it serves its objective. Besides, their counterfactual estimators are domain-agnostic and do not leverage the specific behavior of users on search and recommendation services. *To the best of our knowledge, no other work has evaluated how counterfactual L2R estimators perform when they are applied on policies different from the logging policy, i.e., under policy distributional shift.* Conversely, the offline reinforcement learning (RL) literature has extensively tackled policy distributional shift (Levine et al., 2020; Precup et al., 2000; Janner et al., 2021), including model robustness in model-based RL (Yu et al., 2020; Argenson and Dulac-Arnold, 2021), which can be seen as a similar topic to ours, but these methods are both structure and domain-agnostic because they do not leverage the specific properties of 1. policies returning rankings and 2. user behavior on search and recommendation systems.

Dai et al. (2021) consider a form of intrinsic distributional shift which occurs when the model is asked to generate a consistent click sequence while having been trained using conditional click probabilities on ground truth clicks, but they did not consider the policy distributional shift induced by the change of policy for evaluation. In the work that is perhaps the closest in essence to ours, Huang et al. (2020) introduce a protocol for evaluating policies produced by models trained inside a de-biased simulator, but they address the topic of single-item recommendation, which does not require click models because the sources of bias are different from biases occurring in L2R.

4.2.2 Click models

As mentioned in the previous section, we focus on the evaluation of counterfactual estimators fitted from observed logged data, i.e. click models. In early work on click models (Craswell et al., 2008; Dupret and Piwowarski, 2008; Chapelle and Zhang, 2009; Liu et al., 2016), authors encode assumptions about user behavior into a probabilistic graphical model (PGM) framework in order to separate the influence of the result page's presentation from the influence of the document's intrinsic relevance.

For example, the examination hypothesis, introduced with the position-based model (PBM) (Craswell et al., 2008), postulates that a document must be examined and perceived as relevant in order to be clicked. Therefore, in this model, one must identify relevance and examination parameters through a method of parameter estimation such as expectation-maximisation or stochastic gradient descent. The cascade model (Craswell et al., 2008) additionally states that users browse the page in a top-down fashion, and that the click probability at a given rank depends on document relevance at lower ranks. Although some of these assumptions can easily be challenged in many modern search engines, they allow a certain level of generalisation in many practical settings (Chuklin et al., 2015).

More recently, neural click models have emerged (Borisov et al., 2016; Dai et al., 2021; Chen et al., 2020; Borisov et al., 2018; Zheng et al., 2019) because they enable a better representation of the relevant variables (query, document, vertical type, etc) and recent optimisation methods are efficient for training large-scale click models from abundant data. But some of these models (Chen et al., 2020; Dai et al., 2021) do not offer a straightforward solution for extracting intrinsic relevance scores, because they encode a notion of contextual relevance, i.e., dependent on rank, previous clicks and SERPs and vertical type.

The comparison of these different click models with respect to their robustness to policy distributional shift is the main focus of this chapter. To allow for a fair comparison between them and filter out differences originating from representational and algorithmic design choices, we adopt the same way of representing entities such as query, context and document and the same type of optimisation algorithms, while keeping the structural assumptions of the original methods. In particular, the "older" methods (Craswell et al., 2008; Dupret and Piwowarski, 2008; Chapelle and Zhang, 2009) are instantiated with the

representational and optimisation tools used by newer methods (Borisov et al., 2016; Dai et al., 2021; Chen et al., 2020; Borisov et al., 2018), i.e., deep neural networks with embedding-based input encodings and gradient-based optimisation algorithms.

4.3 EXPERIMENTAL SETUP

This section describes the experimental setup we adopt for the remainder of the chapter. Section 4.3.1 introduces the necessary notations and recalls how click models are traditionally evaluated, and in Section 4.3.2 we define the stack of click models we will use and compare throughout the experiments. We detail our evaluation protocol for assessing click models' robustness to policy distributional shift in Section 4.5, after a preliminary experiment highlighting the shortcomings of the existing evaluation protocol (Section 4.4).

4.3.1 Problem statement and existing protocol

We consider users engaging with a web search system displaying SERPs, i.e., fixed-size ranked lists of documents $\mathcal{S} = (d_1, \dots, d_R)$, as a response to a query q that the user entered. The user can then click on zero, one or more documents, which is represented by the sequence of binary outcomes $(c_1, \dots, c_R)$. Therefore, logs in the dataset $\mathcal{D}$ contain SERP-wide interactions $\mathcal{I}_\mathcal{S} = (\tau_1, \dots, \tau_R)$ with $\tau_r = (q, d_r, r, c_r)$, where $1 \leqslant r \leqslant R$. For simplicity of notation, we will write $\mathbf{1}_{(q,d,r)}$ for the function $\tau \mapsto 1$ if $\tau[0] = q, \tau[1] = d, \tau[2] = r$ and 0 otherwise for any $\tau \in \mathcal{D}$.

In order to better convey our argument, we do not consider session-based click models and we strip all datasets of additional context such as vertical type, device, etc.

We consider click models trained by maximum likelihood estimation on the task of predicting clicks based on query and presented SERP. The models' performance is evaluated on two tasks:

- *Click prediction on a separate test set* (randomly or chronologically split), which is measured by the conditional perplexity (PPL) at each rank and the average conditional perplexity:

$$\begin{aligned} \text{PPL@}r &= 2^{\frac{1}{|\mathcal{D}|}\sum_{\mathcal{I}\in\mathcal{D}} c_r^{\mathcal{I}} \log_2 \tilde{p}_r^{\mathcal{I}} + (1-c_r^{\mathcal{I}}) \log_2\left(1-\tilde{p}_r^{\mathcal{I}}\right)} \\ \text{PPL} &= \frac{1}{R}\sum_{r=1}^{R} \text{PPL@}r, \end{aligned} \tag{4.1}$$

  where $\tilde{p}_r^{\mathcal{I}}$ is the conditional click probability according to the model.

- *Estimation of the relevance* $\text{rel}(q,d)$ *against labels provided by human annotators*, which is measured by the normalized discounted cumulative gain (nDCG) at several truncation levels.

Throughout the chapter, we perform a statistical significance analysis by training each model using 10 random seeds, where the seed controls model initialisation as well as the order of input documents to rank in the relevance estimation task. Confidence bounds use Student's t-distribution and statistical tests are Welch's t-tests. Both use a confidence level of 0.95.

4.3.2 Click model definitions

In this section, we define several click models that will be used in the remainder of the chapter. To make the discussion easier in Section 4.4, we group these models into three families: (i) naive baselines based on number of clicks and impressions, (ii) as-is click models that we select from the existing literature, and (iii) modified click models, which are adapted from the literature to fit our requirements, that we introduce to enrich the experiments. As we mentioned in the introduction, we are interested in assessing the robustness of different structural assumptions to policy distributional shift, rather than the effectiveness of specific implementation choices. To this end, we test several *types* of click models which differ by their key structural assumptions, and we implement all models in the same way, i.e., with modern neural methods, to ensure a fair and up-to-date comparison, as we describe in Section 4.3.2.

Naive baselines

For each document-query pair (q,d), we define the number of clicks at rank r as $N^c_{q,d,r} = \sum_{\tau\in\mathcal{D}} c^\tau \mathbf{1}_{(q^\tau=q, d^\tau=d, r^\tau=r)}$ and aggregated over all ranks as $N^c_{q,d} =$

$\sum_r N^c_{q,d,r}$, where $\tau = (q^\tau, d^\tau, r^\tau, c^\tau)$, as defined in the previous section. Similarly, we define the number of impressions at rank r as $N^i_{q,d,r} = \sum_{\tau \in \mathcal{D}} \mathbf{1}_{(q^\tau = q, d^\tau = d, r^\tau = r)}$ and aggregated over all ranks as $N^i_{q,d} = \sum_r N^i_{q,d,r}$. Using these quantities, we define the following models:

DCTR: document-based click through rate model (Craswell et al., 2008): It is one of the simplest click models, assuming that every document in the list is examined equally, independently of its rank. Consequently, its estimate of CTR is identical to its estimate of relevance probability:

$$P(C_d = 1|q) = \text{rel}(q, d) = \frac{N^c_{q,d}}{N^i_{q,d}}. \tag{4.2}$$

where C_d is a random variable indicating whether the user has clicked document d.

DRCTR: rank-weighted dCTR: It is an improved version of the dCTR model, relaxing its rank-independence assumption. It basically consists of a combination of two simple models: the dCTR model and the rCTR model (Chuklin et al., 2015), the latter making a document-independence assumption (i.e. the click probability only depends on the rank of the document).

$$\begin{aligned} P(C_{d_r} = 1|q) &= \frac{N^c_{q,d,r}}{N^i_{q,d,r}}, \\ \text{rel}(q, d) &= \sum_r \frac{1}{r\text{CTR}(r)} \times \frac{N^c_{q,d,r}}{N^i_{q,d,r}}, \end{aligned} \tag{4.3}$$

with $r\text{CTR}(r) = \left(\sum_{\tau \in \mathcal{D}} c_\tau \mathbf{1}_{(r_\tau = r)}\right) / \left(\sum_{\tau \in \mathcal{D}} \mathbf{1}_{(r_\tau = r)}\right)$.

TOPPOP: popularity-based model, based on the number of clicks:

$$\text{rel}(q, d) = N^c_{q,d}. \tag{4.4}$$

Of course, the relevance estimate of this model is strongly influenced by the logging policy, because the latter will determine the number of impressions of a document and, therefore, the number of opportunities to be clicked. The following two models further exacerbate this bias, by giving more importance to the number of impressions of a document. They are not intended to be used as realistic click models, but as a way of detecting and quantifying the potentially detrimental effect of such bias in our experiments.

TOPPOPOBS: popularity-based model, based on the number of clicks and impressions:

$$\text{rel}(q,d) = N^c_{q,d} \times N^i_{q,d}. \tag{4.5}$$

RANKTOPOBS: rank-weighted popularity-based model, based only on the number of impressions:

$$\text{rel}(q,d) = \sum_r r\text{CTR}(r) N^i_{q,d,r}. \tag{4.6}$$

Note that TopPop, TopPopObs and RankTopObs are not well-defined click models, as they cannot be used for click prediction. Additionally, for dCTR and drCTR, we add Bayesian smoothing with a Dirichlet prior with parameter $\alpha = 1$ when predicting clicks. This allows us to avoid arbitrarily high perplexities on rarely seen items.

As-is click models

We select from the existing literature four widely used *types* of click models that make different modeling assumptions. Here, we first list and describe the key structural assumptions of the models we include in our experiments, and then explain in more detail the selection and standardisation process, with the help of two exclusion criteria.

We select the following four types of model from the literature :

PBM: position-based model (Craswell et al., 2008). It makes the examination hypothesis by stating that a user clicks document d if and only if that document is attractive to the user and has been examined by them: $C_d = 1 \Leftrightarrow A_d = 1$ and $E_d = 1$. Moreover, it encodes this assumption such that attractiveness $\alpha_{q,d}$ only depends on the query-document pair and examination γ_r on the rank:

$$\begin{aligned} P(C_d = 1) &= P(A_d = 1) \times P(E_d = 1) \\ P(A_d = 1) &= \alpha_{q,d} \\ P(E_d = 1) &= P(E_r = 1) = \gamma_r. \end{aligned} \tag{4.7}$$

It is further assumed that the document relevance coincides with its attractiveness:

$$\text{rel}(q,d) = \alpha_{q,d}. \tag{4.8}$$

UBM: user browsing model (Dupret and Piwowarski, 2008). UBM makes the same assumptions as PBM, except that the examination probabilities also depend on the rank of the latest clicked document:

$$\begin{aligned} P(C_d = 1) &= P(A_d = 1) \times P(E_d = 1) \\ P(A_d = 1) &= \alpha_{q,d} \\ P(E_d = 1) &= P(E_r = 1 \mid C_1 = c_1, \ldots, C_{r-1} = c_{r-1}) = \gamma_{r,r'}, \end{aligned} \tag{4.9}$$

where $\gamma_{r,r'}$ is the probability that the document at rank r is examined given that the latest clicked document was located at rank r'.

DBN: dynamic Bayesian network model (Chapelle and Zhang, 2009). Comparatively to previous models, it adds the concept of satisfaction S_d that can happen with probability $\sigma_{q,d}$ after a click; note that this satisfaction probability depends on the particular query-document pair. The model assumes that the user examines the page in a top-down fashion with discount factor γ until they are satisfied:

$$\begin{aligned} P(C_d = 1) &= P(A_d = 1) \times P(E_d = 1) \\ P(A_d = 1) &= \alpha_{q,d} \\ P(E_1 = 1) &= 1 \\ P(E_r = 1 \mid E_{r-1} = 0) &= 0 \\ P(E_r = 1 \mid S_{r-1} = 1) &= 0 \\ P(E_r = 1 \mid E_{r-1} = 1, S_{r-1} = 0) &= \gamma \\ P(S_r = 1 \mid C_r = 1) &= \sigma_{q,d}. \end{aligned} \tag{4.10}$$

The relevance probability is then estimated by:

$$\text{rel}(q, d) = \alpha_{q,d} \times \sigma_{q,d}. \tag{4.11}$$

NCM: neural click model (Borisov et al., 2016). NCM does not encode the examination hypothesis but instead considers click prediction as a sequence-to-sequence problem, where the SERP is modeled as a top-down sequence:

$$P(C_{d,r_d} = 1) = f(q, d_1, \ldots, d_{r_d}, c_1, \ldots, c_{r_d - 1}), \tag{4.12}$$

where f is parameterised as a long short-term memory (LSTM) in the original implementation (Borisov et al., 2016), with ad-hoc representations (embeddings) for queries, documents and clicks. Because NCM

does not implement the examination hypothesis, it does not explicitly infer a latent variable which can be interpreted as the relevance probability. Instead, thanks to the top-down browsing assumption, Borisov et al. (2016) suggest that a relevance score can be inferred by placing the considered document at the first position of the ranking:

$$\text{rel}(q, d) = f(q, d). \tag{4.13}$$

Selection process. The selection of click models from the literature was made according to two exclusion criteria :

- **EC1 : The click model should not leverage context other than position and previous clicks on the same SERP**, such as vertical type, timestamp, previous SERPs, etc. This allows us to compare models on a fair basis, and we leave the effect of context features on the robustness of click models for future work. This criterion prevents the study of context-related biases but still allows the mitigation of the main identified sources of bias such as position bias (Joachims et al., 2005) which constitute the motivation for using click models.
- **EC2 : The click model should offer a way to compute intrinsic, decontextualized relevance scores**, i.e., relevance scores depending solely on the query-document pair. This is required in order to compute nDCG, and to leverage relevance scores in downstream tasks such as label debiasing.

As a result of **EC1**, we exclude context-aware models such as (CSM, Borisov et al., 2018), (MCM, Zheng et al., 2019), (CACM, Chen et al., 2020), (AICM, Dai et al., 2021) and (GraphCM, Lin et al., 2021). However, we include in Section 4.3.2 a variant of (CACM, Chen et al., 2020) stripped of additional context, called CACM$^{\ominus}$. When stripped of such context, AICM and MCM reduce respectively to (NCM, Borisov et al., 2016) and (DBN, Chapelle and Zhang, 2009), that we already include in our experiments, and GraphCM reduces to CACM$^{\ominus}$. It should also be noted that CTR prediction models (Cheng et al., 2016; Guo et al., 2017), which are widely used in recommendation settings, incorporate no concept of latent relevance and do not explicitly consider the ranks of the recommended items. Moreover, these models exploit the "collaborative" nature of the problem, namely that a single item is typically seen by numerous

users and users interact with a lot of common items. In our stylised setting, the queries of the datasets are associated most of the time with disjoint sets of documents and the only available information consists of their id's, their ranks and the clicks. Consequently, there is no collaborative nature we could capture and rely on. In other words, these model reduce to the key assumptions of (dCTR, Craswell et al., 2008) (included in the previous section) when stripped of additional features.

To satisfy **EC2**, we also had to modify the relevance model of our stripped variant of CACM, as we will see in Section 4.3.2, because the relevance variable depends on rank and previous clicks in the original chapter. For the same reason, we had to exclude CSM because even when stripped of the use of timestamp, its bi-GRU architecture does not allow the computation of decontextualized relevance scores.

Standardisation process. In order to ensure a fair comparison, we implement the included models using the same representation of query and documents. More specifically, for models following the examination hypothesis (PBM, UBM, DBN), we separately encode query and documents into embeddings of size 64, which are then combined using a multi-layer perceptron (MLP). In particular, this means that, instead of considering query-document attractiveness ($\alpha_{q,d}$) and satisfaction probability ($\sigma_{q,d}$) directly as model parameters typically identified by an Expectation-Maximisation algorithm, these variables are modeled as the output of an MLP whose inputs are trainable document and query embeddings. For NCM, we directly pass the concatenation of the embeddings of query and document and a binary variable representing the click value at previous rank, rather than the distributed representation introduced in the original chapter. We also replace the original long short-term memory (LSTM) with a gated recurrent unit (GRU), as in our experiments it obtained the same performance more quickly. These implementation choices allow us to isolate the contribution of the structural assumptions encoded in these models and can be considered standard with respect to recent click model literature (Chen et al., 2020; Dai et al., 2021; Borisov et al., 2018). All models are trained by minimizing the cross-entropy between the predicted click distribution and the actual one, using stochastic gradient descent. Further training and implementation details of the click models can be found in Appendix 4.A.

Modified click models

We adapt structural assumptions from the literature to our requirements, and therefore add two new types of click models encoding two different structural assumptions:

CACM$^{\ominus}$: Minimalistic, context-free variant of the context-aware click model (CACM, Chen et al., 2020). This click model relies on the examination hypothesis by having on one side attractiveness probabilities modeled with an MLP from query/document pairs, similarly to PBM/UBM/DBN, and on the other side examination probabilities computed similarly as in the original CACM paper (Chen et al., 2020): the examination score is the output of a GRU whose input at each rank is the concatenation of a position embedding and an embedding for the previous click. There are two main differences with the original CACM :

1. CACM$^{\ominus}$ is context-free, i.e., it does not leverage information from past interactions within the same session and the type of vertical. This is to ensure fair comparisons, folowing our exclusion criterion **EC1**.
2. The attractiveness probabilities are agnostic to the rank of the document as well as previous clicks, which allows us to compute non-contextual relevance scores. This is in line with our exclusion criterion **EC2**.

Even though we drop certain specificities of the original work, our variant keeps the key structural assumptions of CACM. Also, note that CACM$^{\ominus}$ differs from NCM as it implements the examination hypothesis. We have:

$$\begin{aligned} P(C_d = 1) &= P(A_d = 1) \times P(E_d = 1) \\ P(A_d = 1) &= \alpha_{q,d} \\ P(E_d = 1) &= \text{GRU}((c_1, p_1), \ldots, (c_{r_d - 1}, p_{r_d - 1})) \\ \text{rel}(q, d) &= \alpha_{q,d}. \end{aligned} \tag{4.14}$$

where c_k and p_k designate embeddings for click and position respectively, while $\alpha_{q,d}$ is derived as the output of an MLP whose inputs are trainable document and query embeddings.

ARM: auto-regressive click model. Similarly to CACM$^{\ominus}$, ARM only differs from PBM, UBM and DBN by the way it models examination: the exam-

ination probability is the output of a logistic regression f_ω on previous clicks, with one parameter for each absolute rank. We can write:

$$\begin{aligned} P(C_d = 1) &= P(A_d = 1) \times P(E_d = 1) \\ P(A_d = 1) &= \alpha_{q,d} \\ P(E_d = 1) &= f_\omega(c_1, \dots, c_{r_d - 1}) \\ \text{rel}(q, d) &= \alpha_{q,d}, \end{aligned} \tag{4.15}$$

with $f_\omega(c_1, \dots, c_k) = \omega_0 + \sum_{j=1} \omega_j c_j$ for all $j \leqslant k$ and $\alpha_{q,d}$ derived as the output of an MLP whose inputs are trainable document and query embeddings. Even though ARM is similar to CACM$^{\ominus}$ in its architecture, the logistic regression requires significantly less parameters to predict the examination probability, and non-clicked documents do not influence the examination. We also introduce a non-causal version of ARM (called ARM NC) where instead of passing only the previous clicks to the logistic regression, we pass all previous clicks as well as the current one, i.e., $P(E_d = 1) = f_\omega(c_1, \dots, c_{r_d})$. This variant involves a degenerate training tasks: learning to predict a click which has been passed as input ! Yet, we include it in our experiments in order to highlight the shortcomings of the existing evaluation protocol.

Note once again that our contribution is not to introduce new click models or to improve existing ones, but to test the robustness of several structural assumptions in the context of policy distributional shift. To do so, we adopt a setting which can be considered standard with respect to the recent click model literature (Borisov et al., 2016; Chen et al., 2020; Dai et al., 2021; Borisov et al., 2018), and we add two new click models (CACM$^{\ominus}$, ARM) in order to draw more reliable conclusions during our experiments.

4.4 NAÏVE BASELINES BEAT ADVANCED MODELS AT RELEVANCE ESTIMATION

In this section, we perform a preliminary experiment on real datasets using the widely used experimental setup described in Section 4.3.1, i.e., by computing two offline metrics: perplexity and nDCG. We first give experimental details in Section 4.4.1, and present the results in Section 4.4.2.

4.4.1 Data and evaluation protocol

We evaluate click models on two real-world datasets: the Yandex Relevance Prediction dataset (Serdyukov et al., 2012) and the CLARA dataset, which comprises logs from Naver, a major South Korean search engine,[1] and relevance labels provided by human annotators. For both datasets, we follow the same processing workflow: (i) we first break down sessions into separate SERPs as we do not wish to use the additional information contained in a session beyond the current page; then (ii) we restrict the dataset to queries that have been annotated; finally (iii) we discard all pages without clicks. After pre-processing, the Yandex dataset contains 255,467 unique documents and 4,991 unique queries and the CLARA dataset contains 1,345,880 documents and 1,507 uniques queries. Both datasets have a cutoff rank of 10.

For perplexity computation, we use a chronological split where the test and validation set both represent 1/30th of the full Yandex dataset and 1/20th of the full CLARA dataset. For the nDCG computation, we remove documents which do not appear in the dataset as well as queries whose remaining documents all have equal relevance, as they would output a nDCG of 1 regardless of the quality of the click model.

We report our results in Tables 4.1 and 4.2 and 95% confidence bounds from the t-distribution can be found in Appendix 4.D. Note that when we indicate statistical significance or absence of it, we do not correct for multiple comparisons because we are not looking for at least one test to be positive, but for baselines beating all existing click models on all nDCG metrics, i.e., for all tests to be positive at the same time. Indeed, the informal null hypothesis corresponding to *this experiment* could be defined as "*No naïve baseline beats all click models in terms of nDCG at every truncation level*". Therefore, adding comparisons to the experiment (more click models, more truncation levels) would only *increase* the likelihood of at least one test to be negative and decrease the likelihood of our hypothesis being rejected.

We also report in Appendix 4.E the results measured by two other metrics, namely area under the ROC curve for click prediction and recall for relevance estimation.

1 https://www.naver.com/

Table 4.1: Results on the CLARA dataset. ① are naive baselines, ② as-is click models, and ③ modified click models; see Section 4.3.2. The best performing model in average is reported in bold and $^{=}$ indicates a result is not significantly worse than the best performing model. Additionally, † before a model's nDCG from ② or ③ indicates it is not significantly worse than ARM NC's. ↓ : lower is better; ↑ : higher is better. Full confidence bounds can be found in Appendix 4.D.

		Perplexity ↓					**nDCG ↑**			
	Click model	Avg.	@1	@2	@5	@10	@1	@3	@5	@10
①	RankTopObs	–	–	–	–	–	**0.7911**	**0.8360**	**0.8695**	**0.9167**
	TopPopObs	–	–	–	–	–	0.7797	0.8263	0.8652	0.9137
	TopPop	–	–	–	–	–	0.7782	0.8285	0.8646	0.9136
	dCTR	1.1413	1.4000	1.2502	1.1333	1.0515	0.7445	0.7881	0.8323	0.8954
	drCTR	1.1381	1.3932	1.2413	1.1311	1.0500	0.7227	0.7762	0.8224	0.8872
②	PBM	1.1445	$^{=}$1.3955	1.2523	1.1368	1.0546	0.6801	0.7396	0.7993	0.8736
	UBM	1.1410	**1.3906**	1.2431	1.1343	1.0519	0.6825	0.7411	0.8002	0.8744
	DBN	1.1386	$^{=}$1.3936	$^{=}$1.2282	1.1283	1.0523	0.6665	0.7355	0.7967	0.8715
	NCM	**1.1371**	1.3994	**1.2274**	**1.1254**	**1.0497**	0.6554	0.7311	0.7942	0.8688
③	CACM$^{\ominus}$	1.1417	$^{=}$1.3962	1.2437	1.1337	1.0519	0.6808	0.7406	0.8001	0.8740
	ARM	1.1438	1.3994	1.2478	1.1350	1.0566	†0.6861	†0.7442	†0.8027	†0.8760
	ARM NC	–	–	–	–	–	0.6930	0.7456	0.8035	0.8768

Table 4.2: Results on the Yandex dataset. Same conventions as in Table 4.1.

		Perplexity ↓					**nDCG ↑**			
	Click model	Avg.	@1	@2	@5	@10	@1	@3	@5	@10
	RankTopObs	–	–	–	–	–	0.7138	0.7003	0.7291	0.8034
	TopPopObs	–	–	–	–	–	0.7206	0.7023	0.7319	0.8056
①	TopPop	–	–	–	–	–	0.7225	0.7110	0.7374	0.8097
	dCTR	1.3212	1.6054	1.5581	1.2901	1.1790	**0.7246**	**0.7165**	**0.7456**	**0.8155**
	drCTR	1.3146	**1.5784**	1.5439	1.2895	1.1764	0.6273	0.6531	0.6989	0.7795
	PBM	1.3177	1.5970	1.5488	1.2884	1.1781	0.5977	†0.6243	†0.6734	†0.7621
②	UBM	1.2823	1.5918	1.5366	1.2463	1.1247	†0.6017	†0.6255	†0.6734	†0.7631
	DBN	1.2803	1.5861	1.5200	1.2463	1.1271	0.5785	0.6096	0.6610	0.7529
	NCM	**1.2717**	1.5842	**1.5141**	**1.2361**	**1.1152**	0.5606	0.5987	0.6518	0.7462
	CACM$^{\ominus}$	1.2789	1.5969	1.5443	1.2405	1.1165	0.5648	0.6008	0.6546	0.7485
③	ARM	1.3147	1.6142	1.5755	1.2719	1.1807	0.5993	†0.6258	†0.6730	†0.7616
	ARM NC	–	–	–	–	–	0.6086	0.6279	0.6756	0.7634

4.4.2 Results

In Tables 4.1 and 4.2, we report the performance of the click models we described in Section 4.3.2. NCM achieves the lowest perplexity at all ranks but the first one on both datasets, while dCTR and RankTopObs achieve the highest nDCG on respectively Yandex and CLARA. Overall, models in Group ①, i.e., naïve baselines, beat well-formed models from Groups ② and ③ on the relevance estimation task.

As mentioned in the introduction, even though measuring perplexity on a test set collected by the same policy as the training set is not able to guarantee robustness to policy distributional shift, one could hypothesise that the evaluation protocol of the relevance estimation task (using nDCG) is a good proxy for ensuring effective debiasing and therefore robustness to distributional shift.

However, the high nDCG of Group ① in both tables seems to contradict this hypothesis: *naïve baselines beat all click models in terms of nDCG, while we expect most of these baselines to be strongly biased by the logging policy* (especially TopPop, TopPopObs, Weighted TopObs and dCTR). Indeed, they incorporate no mechanism for correcting common biases of the logged data such as position bias or trust bias. Perhaps an even more surprising result is that the non-causal ARM (ARM NC), despite using a degenerate training task, is able to beat most or all existing click models in terms of nDCG on both datasets.

To explain these unsettling results, we hypothesise that when the logging policy is good, the nDCG-based evaluation protocol cannot distinguish between biased models and well-debiasing models, and thus cannot ensure robustness to policy distributional shift. To illustrate this intuition, let us assume the existence of perfect relevance annotations and an optimal logging policy with respect to these annotations. A click model such as dCTR incorporating no bias mitigation mechanism will overestimate the relevance of the most exposed documents and underestimate the relevance of the least exposed documents. But since the logging policy is optimal, the most relevant documents are also the most exposed, so the ordering of relevance scores learned by the biased click model does not differ from the optimal ordering, leading to an nDCG of 1. Yet, this dCTR model is strongly biased and would not be able to accurately predict the CTR of a different policy: for example, if we consider the reverse policy ranking documents by *increasing* order of relevance, dCTR would give the same CTR estimate as for the optimal policy, while this reverse policy would clearly

lead to a lower CTR due to position bias. This counter-example shows that *achieving high nDCG is no guarantee for effective debiasing, and consequently nor for out-of-distribution robustness*. As an aside, biased models can be favored even more if the logging policy uses features that are meaningful for relevance prediction but not observable by the click model, as it is often the case in industrial settings.

Considering, again, the downstream tasks listed in the introduction, this lack of guarantee on debiasing performance and robustness to a change of policy is clearly a critical issue for the required Off-Policy Evaluation in task Groups ② and ③. But it is also problematic in tasks in Group ① because we may obtain narrow, conservative, strongly biased policies as a result of the training process, while we expect the use of click models to provide debiased and potentially diverse policies. As we hinted in the discussion above, nDCG is not a reliable indicator of click model debiasing when the logging policy itself outputs high-nDCG rankings, because one cannot distinguish high nDCG from having successfully debiased the click data and high nDCG from having replicated biases in the click data. However, one might expect that, in practical use cases, the rankings extracted from the click model should be at least as good as those of the logging policy, and that if we observe an improvement in nDCG over the logging policy, it could only be attributed to effective debiasing. On the contrary, we argue that:

(1) in many industrial settings, the logging policy can be expected to obtain higher nDCG than the click models we may train from it. Indeed, commercial search engines usually perform well because they use many additional features, while click models may not attain such performance alone. Instead, the relevance scores extracted from click models may be used as one feature of a larger learning-to-rank model and therefore be useful even without a direct improvement in nDCG over the logging policy;

(2) even if there is an improvement in nDCG over the logging policy, we cannot quantify how much of it can be attributed to effective debiasing. Indeed, nDCG being aggregated over all queries, it is possible that click models are affected by the issue we highlight above on certain queries (e.g., head or tail), even though their aggregated nDCG is higher than the logging policy's. This renders improvements in nDCG unreliable.

4.4.3 Upshot

We have shown that models that we expect to be strongly biased (i.e., naive baselines implementing no bias correction) achieve high nDCG scores. It suggests that the current evaluation protocol based on relevance labels from human annotators is not able to single out biased models from well de-biased models. This would be a critical issue, as click models require correctly debiasing the observed logs in order to perform well on downstream tasks involving policy distributional shift. We therefore formulate the hypothesis that nDCG in the current offline evaluation protocol is not a good indicator of robustness to distributional shift (hypothesis $\mathcal{H}$).

4.5 AN AUGMENTED EVALUATION PROTOCOL

The surprising results of the previous section motivate us to design an augmented evaluation protocol in a simulated environment, in order to verify hypothesis $\mathcal{H}$, to highlight the shortcomings of the nDCG-based evaluation protocol, and to allow researchers and practitioners to mitigate the risks induced by distributional shift.

4.5.1 New evaluation criteria

Robustness of click prediction.

To evaluate the robustness to policy distributional shift of the click prediction capabilities of click models, we can measure the perplexity of the model on a dataset generated using a different ranking policy, i.e., a different distribution of rankings. We call this metric the *out-of-distribution* (ood) *perplexity*, as opposed to the usual *in-distribution* (ind) *perplexity*. If the perplexity of a model significantly increases on a new policy, we can conclude that causal identification during training partly failed, leading to high sensitivity to policy distributional shift. This protocol thus evaluates the downstream performance of click models on the off-policy evaluation (OPE) task (② in the introduction). Actually, the absolute value of perplexity is affected by other factors than click model's performance: it also depends on the dataset's click distribu-

tion, which itself depends on the choice of ranking policy, meaning we cannot directly compare ind-perplexity and ood-perplexity. Therefore, in Section 4.6.2, we only look at a *normalised perplexity* bounded by the respective performance of the best and the worst click model:

$$n\text{PPL}(\text{CM}_i) = 0.2 + \log\left(1 + \frac{\text{PPL}(\text{CM}_i) - \min_k \text{PPL}(\text{CM}_k)}{\max_k \text{PPL}(\text{CM}_k) - \min_k \text{PPL}(\text{CM}_k)}\right), \quad (4.16)$$

where $\text{PPL}(\text{CM}_i)$ is the perplexity obtained by the i-th click model. The use of the logarithm and the additive constant in this formula is simply for ease of visualisation in Figure 4.1, in order to spot small absolute differences. Note that using normalised perplexity means that all models in the experiment are compared relatively to each other. nPPL is bounded by 0.2 for the best model out-of-distribution and $0.2 + \log(2)$ for the worst. More importantly, *a click model CM_i will be considered more robust than its counterparts if its ood normalised perplexity is lower than its ind normalised perplexity, i.e.,* $n\text{PPL}^{ood}(\text{CM}_i) < n\text{PPL}^{ind}(\text{CM}_i)$.

Robustness of subsequent policies

Click models are used to derive an unbiased ranking policy in four of the five tasks we identified in the introduction. In label debiasing for L2R, the policy is obtained by directly ordering documents by decreasing relevance, by a distillation process where an L2R algorithm uses relevance scores as training targets, or by using the unbiased labels as features of an L2R model. In counterfactual L2R, the propensities are extracted from the model, in order to reweight the training targets of an L2R algorithm. In the fair ranking task, relevance and exposure scores are derived to find a policy maximizing a notion of utility while satisfying fairness constraints. Finally, in offline bandits and reinforcement learning, the click model is used as a click predictor for training agents seeking to maximise the expected reward, typically the expected number of clicks. It is therefore crucial to assess the quality of the policies that we aim to derive, depending on the choice of downstream task. To do so, we study the expected number of clicks of two downstream policies for each click model:

- The *Top-Down policy*, which consists in ranking documents by decreasing relevance scores according to the probability ranking principle (Robertson, 1977). This is the policy that we aim to recover in the label debiasing task.

- The *Max-Reward policy*, i.e., the policy maximizing the expected number of clicks according to the trained click model. It is the policy we wish to recover in offline bandits. As mentioned in the introduction, this downstream task requires numerous implicit or explicit instances of OPE, which is already evaluated by the first criterion, but it is also affected by the optimiser's curse (Smith and Winkler, 2006): the maximisation process is likely to select rankings that are grossly overestimated by the click model.

Note that in a non-Top-Down environment, i.e., when exposure does not always decrease with the rank, the Max-Reward policy has the potential to lead to more clicks than the Top-Down policy. A click model whose Max-Reward policy incurs a lower CTR than its Top-Down policy would therefore be interepreted as a poorly robust model.

For certain click models (UBM, NCM, ARM and CACM$^{\ominus}$), finding the Max-Reward policy by brute force can become intractable, especially if the cut-off rank of the desired policy is large or if SERP-specific context such as vertical type or GUI presentation is added. In our experiments, we randomly sample 8! rankings from the 10! possible rankings and find the best one according to the click model by brute force. We chose this sampling-based method over guided methods such as Beam Search because it is not biased towards certain types of solution; it is notably well-known that Beam Search favors near-Top-Down solutions (Lowerre, 1976).

4.5.2 Simulator design

The evaluation protocol involving the two criteria presented in the previous section is operationalised in a simulator. Although no simulation can guarantee good online performance, it allows us to mitigate the risks of deploying the model by testing the robustness of click models in a wide range of settings before deployment.

A suitable simulator needs to include the following components:

- An *internal click model*, which emulates the click behavior of users when confronted to a SERP;
- *Ranking policies*, which present SERPs to the simulated users, in response to a query; and

- *Relevance ground truth*, in order to compute the click probabilities as well as the nDCG for the relevance estimation task.

Relevance ground truth

For the relevance ground truth, we use real-world data from the Microsoft Learning to Rank Datasets (Qin and Liu, 2013), allowing us to get relevance labels on a scale of 0 to 4. In practice, we restrict the dataset to 1000 random queries which all have at least 10 documents of not-all-equal relevance.

Ranking policies

From the dataset, we are also able to get two ranking policies: BM25 (Robertson et al., 1994), which we directly extract from the features, and a LambdaMART (Burges, 2010) policy we train from all available features. We then rescale the scores given by these policies to be between 0 and 1. We also derive a near-optimal policy (ϵ-oracle) by adding Gaussian perturbations of variance 0.15 to the rescaled ground truth relevance labels $(2^{\text{rel}_{q,d}} - 1)/15$. Finally, we derive a stochastic variant of all policies by sampling from a Plackett-Luce model (Plackett, 1975; Luce, 1959), in order to allow causal identification by the click models when the policy is used for training. In practice, the sampling is performed using the Gumbel sampling trick (Oosterhuis, 2021a) with a temperature specific to each policy: $T = 0.1$ for ϵ-oracle and lambdamart and $T = 0.03$ for BM25. The resulting policies are therefore stochastic but rather low-entropy.

Query distribution

In order to mimic a realistic query frequency distribution, we fit a power-law model on the query distribution of the CLARA dataset. We find that the k-th most frequent query appears with probability $p \propto (\alpha - 1)k^{-\alpha}$ with $\alpha = 1.12$.

Internal click model

Based on the relevance labels $\text{rel}_{q,d}$, we design three internal click models:

- DBN (Chapelle and Zhang, 2009) with the attractiveness, satisfaction and continuation parameters taken respectively as $\alpha_{q,d} = 0.95 \times (2^{\text{rel}_{q,d}} - 1)/15$, $\sigma_{q,d} = 0.9 \times (2^{\text{rel}_{q,d}} - 1)/15$ and $\gamma = 0.9$.

- A "Complex Click Model" (CoCM), which is a mixture of click models that does not follow either the examination hypothesis or the cascade hypothesis. Therefore, all click models that we evaluate in our experiments suffer from click model mismatch, i.e., their structure cannot accurately model CoCM's distribution. The complete definition of CoCM can be found in Appendix 4.B. Note that the policy placing the most relevant documents at the top is not necessarily optimal with this model as the examination is not top-down.
- CoCM mismatch: A variant of CoCM with a stronger click model mismatch (see Appendix 4.B).

4.6 EVALUATING ROBUSTNESS TO POLICY DISTRIBUTIONAL SHIFT IN A SIMULATOR

The experiment described in Section 4.6.1 below provides counter-examples that confirm hypothesis $\mathcal{H}$ formulated in Section 4.4 and justify our evaluation protocol. Then, we instantiate this protocol and perform a comparison of six click models in Sections 4.6.2 and 4.6.3, corresponding respectively to the simulated deployments of click models for the tasks of CTR prediction (②) and Offline Bandits (③) .

4.6.1 Observable metrics do not guarantee robustness

In this section, we provide counter-examples where the ood-perplexity cannot be inferred from either ind-perplexity or nDCG. In Tables 4.3 and 4.4, we study the effect of policy distributional shift on several click models, under respectively DBN and CoCM as internal click models. For this particular experiment, we design ranking policies so as to create a strong policy distributional shift. To do so, we first sample 10 documents per query using relevance-stratified sampling. Then the training rankings are obtained by sampling in a top-down fashion from a Plackett-Luce model derived from the true relevances of these documents. The ind-perplexity is computed on a randomly-split separate test set. The policy used for ood-perplexity computation consists in ranking the same 10 documents by *increasing* order of relevance scores. Consequently, the

training policy contains spurious correlations (e.g., position bias) that do not hold under the ood-testing policy, and it is near-optimal, which may lead to the behavior observed in Section 4.4 according to our hypothesis.

Table 4.3: Effect of distributional shift with DBN as internal click model (same conventions as in Table 1). DBN Oracle is a DBN model with hardcoded optimal parameters, and therefore shows a lower bound of ind and ood perplexity.

Click model	ind PPL ↓	nDCG@3 ↑	ood PPL ↓
DBN Oracle	1.2856 (+- 0.0000)	1.0 (+- 0.0000)	1.3002 (+- 0.0000)
DBN	1.3230 (+- 0.0005)	0.7784 (+- 0.0136)	1.3355 (+- 0.0016)
dCTR	1.3428 (+- 0.0000)	0.9219 (+- 0.0015)	1.4683 (+- 0.0000)
PBM	1.3336 (+- 0.0005)	0.8482 (+- 0.0033)	1.3561 (+- 0.0023)
UBM	1.3271 (+- 0.0005)	0.8580 (+- 0.0000)	**1.3413 (+- 0.0011)**
NCM	**1.3248 (+- 0.0002)**	0.7851 (+- 0.0124)	1.3501 (+- 0.0027)
CACM$^{\ominus}$	1.3270 (+- 0.0005)	0.8119 (+- 0.0116)	$^{=}$1.3414 (+- 0.0010)
ARM NC^2	(1.2429 (+- 0.0003))	**0.9315 (+- 0.0021)**	1.6100 (+- 0.0074)

Table 4.4: Effect of distributional shift with CoCM as internal click model (same conventions as in Table 1). CoCM Oracle is a CoCM model with hardcoded optimal parameters, and therefore shows a lower bound of ind and ood perplexity.

Click model	ind PPL ↓	nDCG@3 ↑	ood PPL ↓
CoCM Oracle	1.2670 (+- 0.0000)	1.0 (+- 0.0000)	1.2611 (+- 0.0000)
dCTR	1.3025 (+- 0.0000)	0.8476 (+- 0.0018)	1.3485 (+- 0.0000)
PBM	1.3036 (+- 0.0003)	0.7475 (+- 0.0135)	1.3030 (+- 0.0011)
UBM	1.2945 (+- 0.0004)	0.7677 (+- 0.0137)	1.2949 (+- 0.0007)
DBN	1.2973 (+- 0.0003)	0.6674 (+- 0.0111)	1.3040 (+- 0.0006)
NCM	1.2948 (+- 0.0003)	0.6723 (+- 0.0105)	1.3067 (+- 0.0009)
CACM$^{\ominus}$	**1.2928 (+- 0.0002)**	0.7019 (+- 0.0151)	**1.2934 (+- 0.0011)**
ARM NC^2	(1.1891 (+- 0.0003))	**0.8765 (+- 0.0029)**	1.7012 (+- 0.0108)

dCTR and ARM NC turn out exhibiting very poor ood-PPL, especially with a near-optimal logging policy (in Table 4.3), despite achieving high nDCG. This collapse under the test policy shows that they were unable to learn meaningful relationships, and are instead biased by the logging policy. More importantly,

PBM, UBM, DBN, NCM and CACM$^{\ominus}$, which were designed to be unbiased with respect to the logging policy, show varying levels of robustness, and it does not seem possible to accurately predict their ood-perplexity from ind-PPL and nDCG. *Consequently, we cannot rely on results in either ind-PPL or nDCG to make statements about the success of click models at debiasing the logged data and their robustness to policy distributional shift.*

Our hypothesis regarding why dCTR and ARM NC can achieve such a high nDCG while showing poor performance out-of-distribution is that nDCG indistinguishably rewards biased and well-debiasing model (see Section 4.4). dCTR, incorporating no bias mitigation mechanism, and amplifies the biases present in the logged data. Regarding ARM NC, it seems surprising that a model using such a degenerate training task is even able to get a high nDCG. But it did not collapse to a trivial solution during training because it does not have a parameter specifically assigned to the current click (parameters are associated to absolute ranks). The functional structure of the examination branch makes it impossible to correctly estimate the examination probabilities and this incentivises the relevance branch to directly predict clicks, in a way reminiscent of what the dCTR model is doing. This behaviour consequently leads to strongly biased relevance scores.

It is also worth noting that the ood-PPL of oracle models can be higher or lower than their ind-PPL, even though these models perfectly match the internal click model. This is not surprising because, as explained in Section 4.5, the policy used to generate the data determines the expected click probabilities at each rank, and, therefore, how hard the click prediction task is. As an illustrating example, under DBN, the training policy is near optimal: top documents are very likely to be clicked and bottom documents are almost never clicked, leading to an easy task. But under the test policy, all ranks have either low relevance or low examination probabilities, making the prediction task harder. This is the effect we want to alleviate in our protocol for evaluating the robustness of click prediction by considering normalised instead of absolute perplexity.

2 Note that the perplexity of ARM NC cannot be compared to other click models as it takes the click label as input, but we include it in Tables 4.3 and 4.4 to show that despite having access to the ground truth at test time, the correlations learned by this model during training were so spurious that its ood-perplexity is extremely high.

Upshot

This experiment showed that the observable offline metrics commonly used in click model evaluation cannot guarantee robustness to distributional shift. Our evaluation protocol described in Section 4.5 allows us to observe the effect of distributional shift on simulated deployments for common downstream tasks.

4.6.2 Robustness of click prediction

This section serves three purposes: (i) we instantiate our evaluation protocol for mitigating the risk of a click model being affected by policy distributional shift so that it can be easily reproduced, (ii) we compare the robustness of several click models on the click prediction task, and (iii) we highlight how policy distributional shift can affect click models differently depending on training and test configurations.

Figure 4.1 illustrates the robustness of each click model compared to the other models in the experiment. In this graph, the perplexity is normalised according to our protocol (see Section 4.5). Therefore, when the blue line (ood-perplexity) is inside the red dashed line (ind-perplexity), this means that the model being considered is comparatively more robust than the others, and vice-versa.

PBM usually comes with poor ind-perplexity, but its robustness is far greater than most other models under a near-optimal policy like PL-oracle and strong distributional shift (when tested on BM25 or a random policy), making it a competitive choice for ood-click prediction in this case, despite its poor ind-performance. Note that this setting is common in practice as logging policies from commercial search engines usually have high performance. UBM is also quite robust overall, especially under a near-optimal policy and strong distributional shift. However, when trained on sub-optimal policies and mild click model mismatch, it does not match the robustness of some other models. Unsurprisingly, DBN almost always has the best performance both in-distribution and out-of-distribution when a DBN is also used as internal click model. But under click model mismatch, its robustness is very poor as ind-perplexity is a very unreliable indicator of ood-perplexity and it is constantly worse than other models when trained on a PL-oracle policy. NCM is also particularly brittle to distributional shift under better policies. However, its relatively better

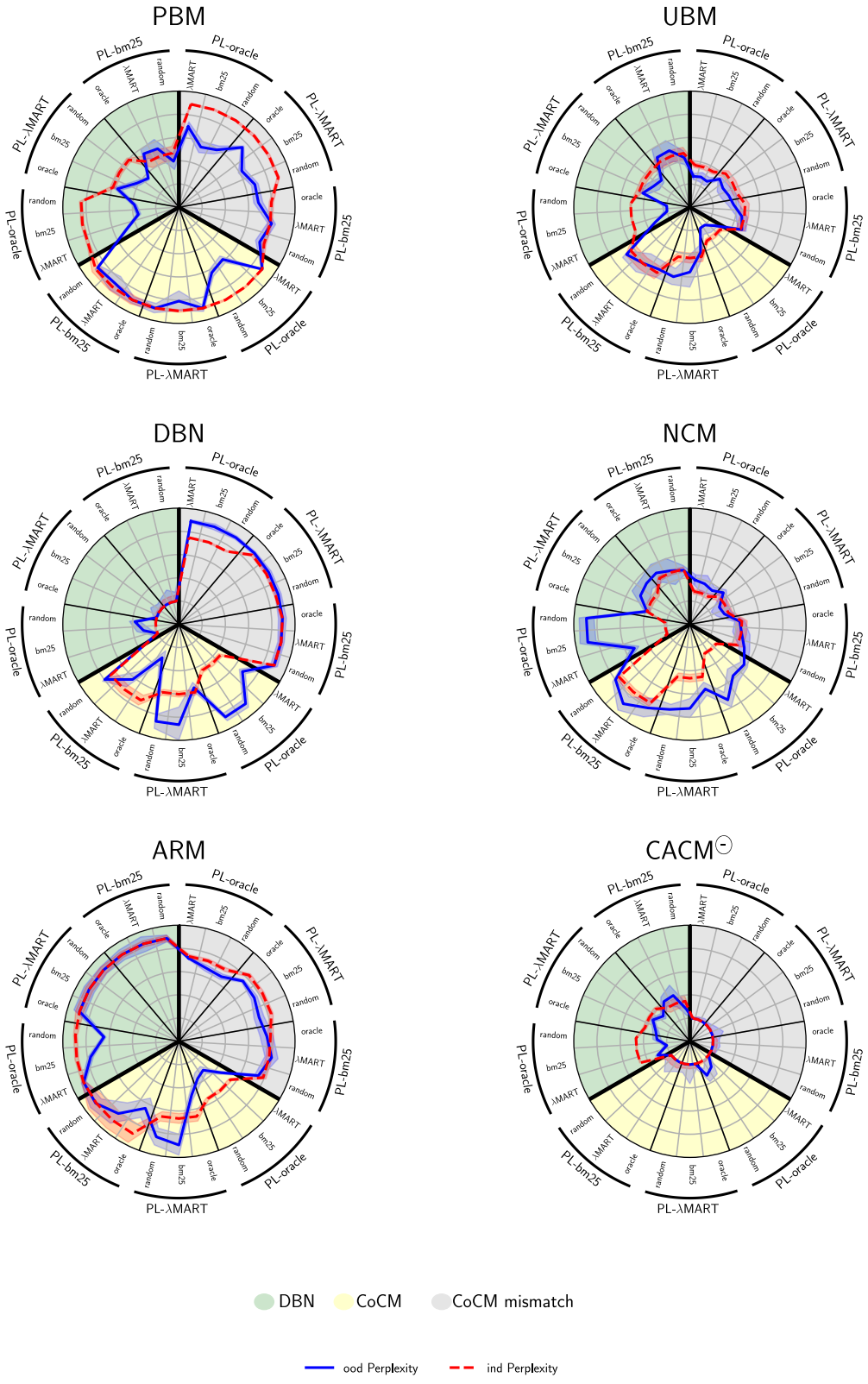


Figure 4.1: (Continued on the following page.)

Figure 4.1: Spider charts showing the level of robustness in a wide range of simulated environments. Perplexities represented on each graph are log-normalised perplexities where the best model is close to the center and the worst model is close to the border (see Section 4.5 for the complete formula of log-normalised perplexity). The background colors represent the type of internal click model, the outer circle of labels indicates the training policy and the inner circle indicates the test policy for ood-perplexity computation. Confidence bounds appear in shaded areas.

representativeness makes it a competitive choice under strong click mismatch. ARM is comparatively more robust than DBN or NCM under PL-oracle, but its ood-performance is quite unreliable and its ind-performance is usually too poor to make it competitive. Finally, CACM$^{\ominus}$ usually retains most of its very good ind-performance when tested out-of-distribution, which also makes it a good candidate for the CTR prediction task.

Upshot

In summary, each click model shows strengths and weaknesses in different settings, but PBM, UBM, and CACM$^{\ominus}$ are generally more robust than ARM, DBN and NCM, especially under strong policy distributional shift and a near-optimal logging policy, which are common real-world conditions. We therefore consider the former three as safer choices for the CTR prediction task ②.

4.6.3 Robustness of subsequent policies

Here, we instantiate the second criterion of our proposed evaluation protocol: measuring the CTR of policies produced by the click model. As explained in Section 4.5, we derive two policies for each click model and simulator configuration: Top-Down and Max-Reward. Under CoCM and CoCM mismatch, the user does not always examine the page in a top-down fashion. Therefore, Max-Reward has the potential to lead to higher click-through rate than Top-Down. Conversely, spurious correlations in the data and high uncertainty on rarely seen SERP configurations may lead to high expected CTR according to the model but poor performance when facing the true internal click model.

In Figure 4.2, we report the observed click-through rate of the Top-Down and the Max-Reward policy extracted from six click models, using CoCM

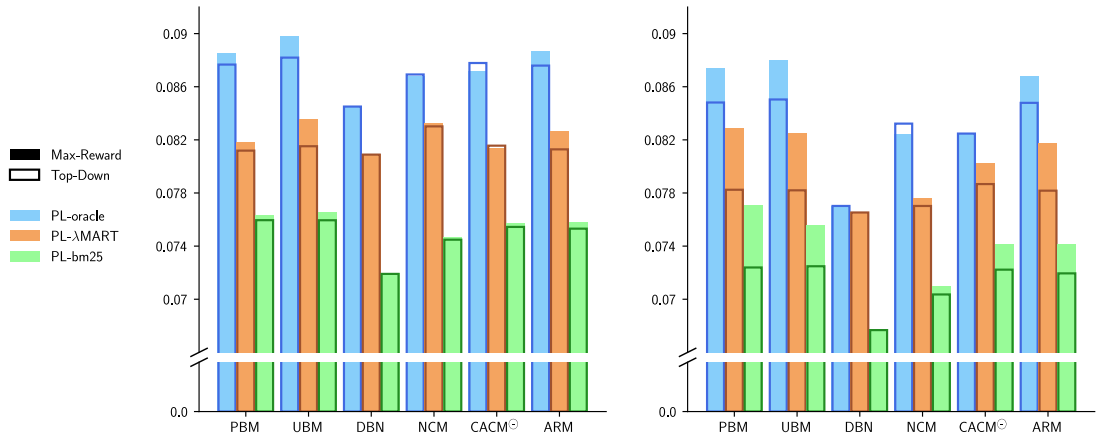


Figure 4.2: Click-through rate obtained by the Top-Down and Max-Reward policies of click models. A figure including confidence bounds is provided in Appendix 4.C. Left: CoCM as internal click model. Right: CoCM mismatch as internal click model.

and CoCM mismatch as internal models. Confirming our intuition, the Max-Reward policy can perform better than the Top-Down policy, especially with CoCM mismatch as internal model (right plot). However, certain models sometimes achieve worse CTR than with Top-Down, demonstrating a critical lack of robustness to policy distributional shift.

By looking more closely at the relative performance of Top-Down and Max-Reward in Figure 4.2, we observe that complex and expressive models like NCM and CACM$^{\ominus}$ are poorly robust since they achieve no-better or worse performance with Max-Reward than with Top-Down in certain configurations. On the contrary, simpler models with fewer parameters, like PBM and UBM, can lead to highly rewarding policies and robustly increase their performance by applying Max-Reward instead of Top-Down.

We hypothesise that by relying on simpler structural assumptions and having fewer but more frequently used parameters to be trained, these models are less likely to suffer from spurious correlations as well as high uncertainty. It is striking to see that models with poor ind click prediction performance can produce highly rewarding policies.

Moreover, CACM$^{\ominus}$ is clearly more affected than ARM by the maximisation of the expected CTR, despite having shown better performance and robustness than ARM on the CTR prediction task (see Section 4.6.2). This suggests that tasks involving maximisation of the expected CTR, such as Fair Ranking and Offline Bandits, are more demanding regarding robustness to policy dis-

tributional shift. The expressivity of CACM$^{\ominus}$ and NCM makes them better at fitting the click distribution, but it also critically exposes them to the optimiser's curse, i.e., they are likely to grossly overestimate the expected CTR of at least one ranking which is going to be selected by the maximisation process.

Upshot

In summary, this experiment shows (i) that distributional shift critically impacts the policies recovered by poorly robust models in certain downstream tasks, e.g., Fair Ranking and Offline Bandits, and (ii) that simple models should not be overlooked as they can produce highly rewarding policies and be robust under distributional shift.

4.7 DISCUSSION

In the previous section, we have instantiated our proposed evaluation protocol with a selection of six types of click models in order to analyse how well-known models perform in practical scenarios that can be encountered in multiple downstream tasks, which have in common requiring out-of-distribution predictions. Our results allow us to identify general trends, e.g., that fairness and bandits tasks require stricter robustness than CTR prediction tasks or that simpler models are usually more robust out-of-distribution. In this section, we discuss how this protocol can be leveraged by practitioners in a real-world deployment scenario. Note that our protocol allows us to compare different click models with respect to each other. It can be used to do that in two different ways:

A first use case would involve a practitioner wishing to quickly assess a new candidate click model before taking the risk of deploying policies based on it for online evaluation. The instantiation on MSLR data that we describe in our experiments can be used out of the box. In order to make this process easier, we provide code and result files that can be readily used when testing a new candidate click model on MSLR annotations and features, without re-training models included in our experiments.

A second possibility is to adapt the protocol described in Section 4.5 to a given search engine. Indeed, it relies on semi-synthetic simulators that can be

derived from graded relevance annotations (to define click probabilities) and document features (to build realistic logging policies). Because of this semi-synthetic setup, the scenario can be made to fit any real-world search engine, with only the internal user click model left to be controlled by the practitioner. In this setup, comparing candidate click models across a wide range of internal click models is key in order to assess how their robustness is affected by click model mismatch.

Also, in both settings, our experimental setup can be enriched with available context features to fit the scenario at hand. Even though our protocol cannot replace online evaluation, it constitutes a way to reduce the cost of deploying new learning-to-rank models by mitigating the risk of under-performance once deployed.

4.8 CONCLUSION

In this work, we have highlighted the limitations of the traditional offline evaluation protocol for click models, specifically that it fails to detect a lack of robustness of click models to policy distributional shift. To do so, we have implemented several types of click models encoding different structural assumptions of user behavior, and have augmented the evaluation of these models by using simulations that aim to mimic real-world deployment for different downstream tasks involving policy distributional shift.

4.8.1 Main findings

Our experiments highlight the existence of a critical issue with click models: the existing offline evaluation protocol cannot guarantee effective debiasing and robustness to distributional shift. We show that it can cause click models to underperform on the target downstream task because of poor out-of-distribution policy evaluation capabilities, whether it is when predicting the CTR of unknown policies or when training policies based on the click model's parameters.

Three major take-aways emerge from our experiments:

- They show that certain training configurations (strong click model mismatch, near-optimal training policies) are more likely to be negatively affected by distributional shift than others,

- They allow us to identify risky models (DBN, NCM) as well as relatively safer ones (PBM, UBM), and

- They provide practitioners with a way of mitigating the risks of deploying policies based on candidate click models by detecting the lack of robustness before deployment.

4.8.2 Broader implications

Our findings indicate that counterfactual models and estimators must be carefully evaluated in order to make them trustworthy for practical use in downstream tasks and that, despite being convenient, offline metrics can miss important robustness issues in certain settings.

On a more actionable note, our work suggests that getting more diverse test sets, i.e., from different logging policies, should be considered whenever possible. Moreover, simulations can play a role in an offline evaluation protocol by measuring otherwise unobservable metrics, as long as we evaluate on a wide range of simulations so as to mitigate the influence of simulator design. Developing high-quality, learnable simulators matching the dynamics of real-world deployment could also further mitigate the risks associated with it.

4.8.3 Limitations

We implemented click models in a standardised, context-free fashion in order to fairly compare their resilience to policy distributional shift, but many improvements of these models leveraging context features have been proposed in recent years (Borisov et al., 2018; Zheng et al., 2019; Dai et al., 2021; Chen et al., 2020). We expect the general trends identified in this work to generalise to these context-aware models, but the precise effect of data enrichment with abundant side information and historical behavior remains unaddressed.

4.8.4 Future work

Generalising predictions out-of-distribution is a hard problem exhibiting no theoretical guarantees without further assumptions (Shen et al., 2021). In this work we showed that click models, which aim to lift the in-distribution requirement by encoding such assumptions in their architecture, cannot be simply evaluated with traditional offline metrics and human relevance annotations. Future work should therefore investigate under which assumptions we can derive theoretical results for the generalisation capabilities of click models, and how their offline results relate to their online performance when such assumptions are satisfied.

As we have seen in the experiments, the most robust models also tend to be the simplest, whose in-distribution performance is generally subpar compared to more advanced models. Therefore, future work should also investigate strategies to counter the effect of distributional shift in order to attain similar levels of robustness with more complex click models, such as training on multiple logging policies, enforcing invariances or penalizing the epistemic uncertainty.

SUPPLEMENTARY MATERIALS

To facilitate reproducibility of the results in this chapter, we share the code along with guidelines for reproduction at github.com/naver/dist-shift-click-models.

4.9 REFLECTIONS ON THE CHAPTER

4.9.1 Research outcomes

In this chapter, we investigated my second research question:

Research Question 2. *Can we predict in a fully offline manner the performance of models learned on biased data?*

This chapter clearly answered this question negatively, at least when considering existing metrics. More downstream tasks, more complex models with richer content and context for the logged data could certainly be considered. Yet, the core issue of the existing ranking metrics not being adapted to evaluating models trained on biased data remains. We single out this issue in the next section, and propose a method that brings us closer to answering this research question positively.

4.9.2 Additional thoughts

The observation that simpler models are typically more robust than complex ones, especially under click model mismatch, does not come as a surprise. With more predictive power, complex models can improve their goodness-of-fit to the data, but are also more likely to capture spurious correlations lying in the data. This is exacerbated as in a typical web search or recommender system training scheme, the datasets are highly imbalanced and show very little variabilty: the same query almost always leads to the same ranking. Click models are usually operating at the limit of causal identifiability, and adding more bias variables (i.e., modeling more confouders) further restricts the effective support of the data for a given query-document-bias triplet, making estimation even riskier.

This highlights a typical dilemma in confounder modeling, which is especially prevalent in information retrieval systems: simpler causal models underfit the data and do not capture the user behavior well, but complex models leave very little space for de-confounding (in the extreme, if every possible variable was passed to the model, there would be no data variability at all, and therefore no notion of causality. . . things would just happen).

CHAPTER APPENDIX

4.A TRAINING AND IMPLEMENTATION DETAILS

All click models are implemented in PyTorch with PyTorch-Lightning. They are trained using the Adam Optimiser with the ReduceLROnPlateau scheduler with a factor of 0.5 and patience of 2. We use Early Stopping with patience of 3 to stop the training when the validation loss does not improve and restore the best checkpoint for evaluation on the test set. We trained these models on a single nVIDIA V100 GPU and no models required more than 10 minutes to be trained on the simulated datasets.

4.B DEFINITION OF COCM

We consider three modes: the user browses the page in a top-down fashion ($\bigtriangledown$), in a bottom-up fashion ($\bigtriangleup$), or does not look at the page at all and clicks on documents without examining them ($\bigcirc$). In our experiments, we take the respective probabilities of each mode to be $(0.6, 0.3, 0.1)$ for CoCM and $(0.2, 0.7, 0.3)$ for CoCM mismatch.

- In the top-down mode, the click probability of document d at rank k depends on its attractiveness $\alpha_{q,d}$, whether the preceding document has

been clicked c_{k-1}, and the attractiveness of the next document $\alpha_{q,d_{k+1}}$. With A_k, A_{k+1}, C_{k-1} and $E_k|C_{<k}$ jointly independent, we have:

$$\begin{aligned}
P(C_k = 1 \mid q, d_k, d_{k+1}, c_{k-1} = 1, \bigtriangledown) &= 0 \\
P(C_k = 1 \mid q, d_k, d_{k+1}, c_{k-1} = 0, c_{<k-1}, \bigtriangledown) &= \alpha_{q,d_k} \times (1 - \alpha_{q,d_{k+1}}/2) \times \\
&\quad P(E_k = 1 \mid c_{k-1} = 0, c_{<k-1}) \\
P(E_k = 1 \mid c_{k-1} = 1, c_{<k-1}) &= (1 - \sigma) \times \gamma \times P(E_{k-1} = 1 \mid c_{<k-1}) \\
P(E_k = 1 \mid c_{k-1} = 0, c_{<k-1}) &= \gamma \times P(E_{k-1} = 1 \mid c_{<k-1}).
\end{aligned} \tag{4.17}$$

We take $\alpha_{q,d} = (2^{\text{rel}(q,d)} - 1)/15$, $\sigma = 0.7$ and $\gamma = 0.9$. Note that even the top-down mode does not follow the cascade hypothesis as the click probability depends on the relevance of the following document.

- The order of next and previous documents and clicks is simply reversed in the bottum-up mode.
- In the no-look mode, the probability of the document at rank k being clicked is $P(C_k = 1 \mid \bigcirc) = \epsilon_k$ with $\epsilon_k = 0.2 \times 0.9^k$.

4.C FIGURE 4.2 WITH CONFIDENCE BOUNDS

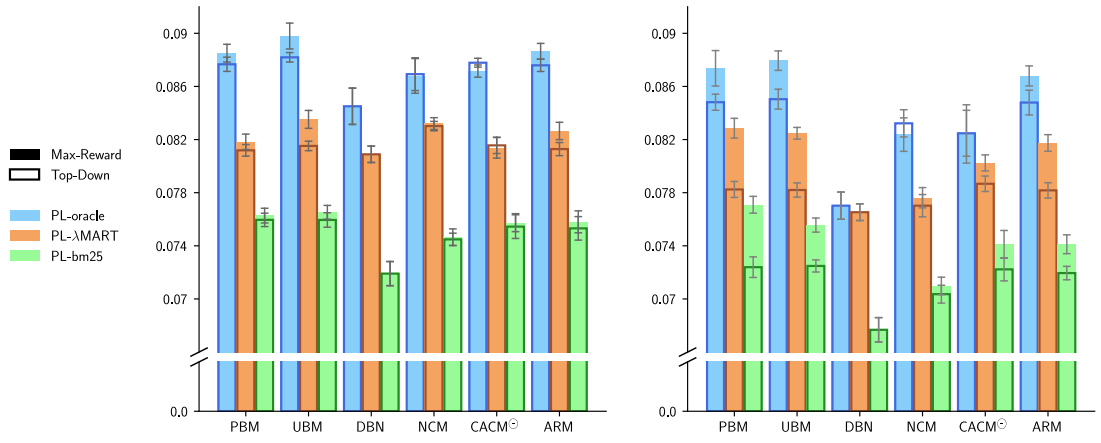


Figure 4.C.1: Click-through rate obtained by the Top-Down and Max-Reward policies of click models. Left: CoCM as internal click model. Right: CoCM mismatch as internal click model.

4.D TABLES 4.1 (LEFT) AND 4.2 (RIGHT) WITH CONFIDENCE BOUNDS

Table 4.D.1: Results on CLARA with 95% confidence bounds.

Click Model	PPL	PPL@1	PPL@2	PPL@5	PPL@10	nDCG@1	nDCG@3	nDCG@5	nDCG@10
RankTopObs	–	–	–	–	–	**0.7911±0.0002**	**0.8360±0.0000**	**0.8695±0.0001**	**0.9167±0.0000**
TopPopObs	–	–	–	–	–	0.7797±0.0009	0.8263±0.0008	0.8652±0.0007	0.9137±0.0003
TopPop	–	–	–	–	–	0.7782±0.0014	0.8285±0.0011	0.8646±0.0008	0.9136±0.0004
dCTR	1.1413±0.0000	1.4000±0.0000	1.2502±0.0000	1.1333±0.0000	1.0515±0.0000	0.7445±0.0010	0.7881±0.0005	0.8323±0.0004	0.8954±0.0002
drCTR	1.1381±0.0000	1.3932±0.0000	1.2413±0.0000	1.1311±0.0000	1.0500±0.0000	0.7227±0.0000	0.7762±0.0001	0.8224±0.0000	0.8872±0.0000
PBM	1.1445±0.0014	$^{=}$1.3984±0.0084	1.2523±0.0009	1.1368±0.0006	1.0546±0.005	0.6801±0.0051	0.7396±0.0029	0.7993±0.0019	0.8736±0.0016
UBM	1.1410±0.0000	**1.3906±0.0003**	1.2431±0.0002	1.1343±0.0001	1.0519±0.0002	0.6825±0.0039	0.7411±0.0023	0.8002±0.0016	0.8744±0.0012
DBN	1.1386±0.0013	$^{=}$1.3936±0.0061	$^{=}$1.2282±0.0009	1.1283±0.0008	1.0523±0.0005	0.6665±0.0067	0.7355±0.0030	0.7967±0.0023	0.8715±0.0016
NCM	**1.1371±0.0002**	1.3994±0.0034	**1.2274±0.0005**	**1.1254±0.0007**	**1.0497±0.0002**	0.6554±0.0047	0.7311±0.0027	0.7942±0.0022	0.8688±0.0011
CACM$^{\ominus}$	1.1417±0.0018	$^{=}$1.3962±0.0085	1.2437±0.0009	1.1337±0.0010	1.0519±0.0009	0.6808±0.0042	0.7406±0.0017	0.8001±0.0010	0.8740±0.0010
ARM	1.1438±0.0014	1.3994±0.0066	1.2478±0.0007	1.1350±0.0007	1.0566±0.0010	†0.6861±0.0076	†0.7442±0.0027	†0.8027±0.0018	†0.8760±0.0015
ARM NC	–	–	–	–	–	0.6930±0.0048	0.7456±0.0015	0.8035±0.0012	0.8768±0.0012

Table 4.D.2: Results on Yandex with 95% confidence bounds.

Click Model	PPL	PPL@1	PPL@2	PPL@5	PPL@10	nDCG@1	nDCG@3	nDCG@5	nDCG@10
RankTopObs	–	–	–	–	–	0.7138±0.0002	0.7003±0.0001	0.7291±0.0001	0.8034±0.0001
TopPopObs	–	–	–	–	–	0.7206±0.0009	0.7023±0.0009	0.7319±0.0005	0.8056±0.0005
TopPop	–	–	–	–	–	0.7225±0.0007	0.7110±0.0010	0.7374±0.0004	0.8097±0.0005
dCTR	1.3212±0.0000	1.6054±0.0000	1.5581±0.0000	1.2901±0.0000	1.1790±0.0000	**0.7246±0.0010**	**0.7165±0.0006**	**0.7456±0.0005**	**0.8155±0.0003**
drCTR	1.3146±0.0000	**1.5784±0.0000**	1.5439±0.0000	1.2895±0.0000	1.1764±0.0000	0.6273±0.0002	0.6531±0.0001	0.6989±0.0001	0.7795±0.0001
PBM	1.3177±0.0013	1.5970±0.0039	1.5488±0.0016	1.2884±0.0009	1.1781±0.0011	0.5977±0.0076	†0.6243±0.0044	†0.6734±0.0038	†0.7621±0.0028
UBM	1.2823±0.0011	1.5918±0.0035	1.5366±0.0018	1.2463±0.0009	1.1247±0.0011	†0.6017±0.0083	†0.6255±0.0050	†0.6734±0.0033	†0.7631±0.0024
DBN	1.2803±0.0008	1.5861±0.0019	1.5200±0.0012	1.2463±0.0006	1.1271±0.0005	0.5785±0.0065	0.6096±0.0030	0.6610±0.0024	0.7529±0.0018
NCM	**1.2717±0.0011**	1.5842±0.0023	**1.5141±0.0025**	**1.2361±0.0011**	**1.1152±0.0008**	0.5606±0.0057	0.5987±0.0043	0.6518±0.0036	0.7462±0.0027
CACM$^{\ominus}$	1.2789±0.0013	1.5969±0.0054	1.5443±0.0032	1.2405±0.0009	1.1165±0.0004	0.5648±0.0105	0.6008±0.0063	.6546±0.0048	0.7485±0.0036
ARM	1.3147±0.0023	1.6142±0.0069	1.5755±0.0033	1.2719±0.0012	1.1807±0.0042	0.5993±0.0048	†0.6258±0.0030	†0.6730±0.0022	†0.7616±0.0018
ARM NC	–	–	–	–	–	0.6086±0.0084	0.6279±0.0040	0.6756±0.0050	0.7634±0.0033

4.E OFFLINE METRICS ON CLARA AND YANDEX : AUROC AND RECALL

Table 4.E.1: Area under the receiving operating characteristic curve (AUROC) and Recall on the CLARA dataset. ① are naive baselines, ② as-is click models, and ③ modified click models; see Section 4.3.2. The best performing model in average is reported in bold and $^{=}$ indicates a result is not significantly worse than the best performing model. ↓ : lower is better; ↑ : higher is better. AUROC@k is computed only on documents at rank k while Recall@k is computed on documents up to rank k, similarly to respectively PPL@k and nDCG@k. Also, AUROC "Full" is computed over the whole dataset.

		AUROC ↑					**Recall ↑**			
	Click model	Full	@1	@2	@5	@10	@1	@3	@5	@10
①	RankTopObs	–	–	–	–	–	$^{=}$0.2521	**0.5814**	0.7908	0.9830
	TopPopObs	–	–	–	–	–	0.2502	0.5789	**0.7929**	$^{=}$0.9838
	TopPop	–	–	–	–	–	**0.2525**	0.5799	$^{=}$0.7927	**0.9842**
	dCTR	0.9214	0.8970	0.7924	0.7902	0.7888	0.2393	0.5445	0.7600	0.9802
	drCTR	0.9255	0.9004	0.8028	0.8097	0.8337	0.2345	0.5376	0.7540	0.9790
②	PBM	0.9161	$^{=}$0.9002	0.7929	0.7998	0.8160	0.1997	0.4958	0.7354	0.9819
	UBM	$^{=}$0.9255	$^{=}$0.9027	$^{=}$0.8100	$^{=}$0.8210	$^{=}$0.8531	0.2020	0.4984	0.7358	0.9819
	DBN	$^{=}$0.9244	0.8961	**0.8110**	0.8195	$^{=}$0.8384	0.1891	0.4903	0.7311	0.9828
	NCM	**0.9292**	**0.9037**	$^{=}$0.8098	**0.8257**	**0.8576**	0.1873	0.4889	0.7320	0.9819
③	CACM$^{\ominus}$	$^{=}$0.9242	$^{=}$0.9010	$^{=}$0.8094	$^{=}$0.8219	$^{=}$0.8546	0.1899	0.4900	0.7320	0.9819
	ARM	0.8906	0.8815	0.7630	0.7724	0.7702	0.1975	0.4936	0.7337	0.9821
	ARM NC	–	–	–	–	–	0.1995	0.4955	0.7342	0.9817

Table 4.E.2: AUROC and Recall on the Yandex dataset. Same conventions as in Table 4.E.1.

		AUROC ↑					Recall ↑			
	Click model	Full	@1	@2	@5	@10	@1	@3	@5	@10
	RankTopObs	–	–	–	–	–	0.7138	0.7003	0.7291	0.8034
	TopPopObs	–	–	–	–	–	0.2139	0.4693	0.6572	0.9127
①	TopPop	–	–	–	–	–	=0.2176	0.4791	0.6643	0.9149
	dCTR	0.8766	0.8014	0.7547	0.7624	0.7697	**0.2185**	**0.4864**	**0.6752**	**0.9207**
	drCTR	0.8819	**0.8135**	0.7670	0.7614	0.7675	0.1778	0.4496	0.6547	0.9157
	PBM	0.8798	0.8035	0.7603	0.7622	0.7575	0.1670	0.4284	0.6364	0.9106
②	UBM	0.9149	0.8057	0.7749	0.8557	0.9194	0.1692	0.4289	0.6345	0.9108
	DBN	0.9166	0.8068	=0.7932	0.8586	0.9199	0.1588	0.4199	0.6283	0.9076
	NCM	**0.9222**	0.8090	**0.7952**	**0.8689**	**0.9264**	0.1539	0.4133	0.6233	0.9061
	CACM$^{\ominus}$	0.9182	0.8008	0.7675	0.8630	0.9245	0.1559	0.4153	0.6256	0.9067
③	ARM	0.8865	0.7988	0.7648	0.8102	0.8111	0.1656	0.4288	0.6348	0.9105
	ARM NC	–	–	–	–	–	0.1681	0.4286	0.6362	0.9108

5

AN OFFLINE METRIC FOR THE DEBIASEDNESS OF CLICK MODELS

The previous chapter gave us an idea of how robust click models are to the distribution shift induced by deploying the newly learnt policy, and how sevral click models compare on their downstream tasks. While a semi-simulated empirical analysis like the one we performed helps understand how new click models will perform in practice, it requires a lot of design choices and a large amount of work to set up.

Therefore, in this chapter, we try to design a single offline metric that would convey roughly the same information about click model robustness as the empirical analysis. In order to do so, we had to restrict the scope of the robustness analysis to the core issue that the previous chapter revealed: that click models can reach high performance on the relevance annotations simply by replicating certain biases present in the offline data, rather than correcting for them. We end up with a metric that, taken together with traditional likelihood and ranking metrics, correlates much better with the actual performance after deployment than the traditional metrics alone.

Our metric, CMIP, only requires ground truth relevance scores (from annotations or randomized traffic) and scores from the logging policy (known or estimated). It is based on the following simple idea: if you want to spot cheaters in a class of student, i.e., those who copied on their classmates but won't be able to generalize to a new test, you cannot simply compare the grades of the students as those who have cheated can get very good grades, sometimes

even better than the classmate they copied on. Instead, you must compare the mistakes that the students made: if two students consistently made the same mistakes, one of them has likely cheated. We use this principle to design a metric that spots the non-robust click models which simply replicated the biases of the data.

This chapter is based on the following publication: **Romain Deffayet**, Philipp Hager, Jean-Michel Renders, and Maarten de Rijke. 2023. An Offline Metric for the Debiasedness of Click Models. In *SIGIR'23: the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

5.1 INTRODUCTION

Search and recommender systems aim to rank items in order of relevance to a given search query or user context (Liu, 2009). Operational search engines have access to large logs of user behavior that are valuable sources for improving ranking systems (Serdyukov et al., 2012; Qin and Liu, 2013; Chapelle and Chang, 2011). However, implicit user feedback in the form of clicks is well-known to be biased (Craswell et al., 2008; Joachims et al., 2005). E.g., clicks can only occur on items exposed to users, introducing selection bias (Ovaisi et al., 2020; Oosterhuis and de Rijke, 2020a). Also, the rank at which a document is displayed greatly impacts the number of users seeing and clicking an item, leading to position bias (Joachims et al., 2005; Joachims et al., 2017). And trust bias arises when users rely on their search engine to place relevant documents at the top leading to clicks on top-ranked items regardless of their relevance (Agarwal et al., 2019; Vardasbi et al., 2020a).

Click models Click models have a long history in web search for modeling user behavior by learning to predict how a user would interact with a given list of items (Craswell et al., 2008; Dupret and Piwowarski, 2008; Chapelle and Zhang, 2009; Chuklin et al., 2015; Borisov et al., 2016). Click models explicitly model effects that impact a user's click decision, such as item relevance, position bias, or trust bias, and are thus a valuable tool for understanding users (Craswell et al., 2008), predicting ad clicks (Zhu et al., 2010; Chen and Yan, 2012; McMahan et al., 2013), as offline evaluation metrics (Chuklin et al., 2013), or estimating

biases that the field of unbiased learning-to-rank aims to mitigate (Joachims et al., 2017; Vardasbi et al., 2020a; Ai et al., 2021).

Commonly, two aspects of click models are evaluated (Grotov et al., 2015; Chuklin et al., 2015). First, a model's ability to accurately predict clicks is commonly measured using the perplexity of the model on a hold-out test set of clicks (Dupret and Piwowarski, 2008). Second, if a model estimates document relevance, metrics such as nDCG or MRR can be computed using relevance annotations gathered by human experts (Chapelle and Zhang, 2009). Recently, Deffayet et al. (2023b), included here as Chapter 4, have shown that the current evaluation protocol of perplexity and nDCG does not guarantee that the best-performing model generalizes well to predicting clicks on unseen rankings. By simulating a variety of user behaviors on rankings created by different ranking policies, the authors show that the best-performing models on one ranking policy are not guaranteed to perform well when presented with the same documents in a different order. This setting simulates a covariate shift in the ranking distribution (also called policy shift).

Failure of generalization Deffayet et al. (2023b) identify two cases where the current evaluation protocol breaks down. First, they find that biased click prediction methods can achieve high nDCG scores, especially when the policy that collected the click data is already near-optimal and tends to generate similar rankings. Picture the case in which all documents are ranked in order of relevance to the user. In this setting, position bias perfectly correlates with document relevance, and naive methods such as using a document's average click-through rate (CTR) as relevance and click prediction will lead to strong nDCG and perplexity scores. But this method fails to predict clicks on the inverted ranking in which the most relevant item is displayed at the bottom and is, thus, highly affected by position bias. In this case, predicting the average CTR of a document as inferred from the original dataset is not a good prediction of user behavior on the inverted ranking, and the model fails to generalize. This model is not invariant under policy shift because it gives different predictions depending on the train rankings.

Second, the authors find that click model mismatch, a case in which the assumptions of the click models do not match the user behavior in the collected dataset, can lead to wrong conclusions about which models generalize well to unseen rankings. While Deffayet et al. (2023b) evaluate a variety of click models and identify trends about which click models tend to generalize better,

we still lack a principled approach to reliably select a click model from a set of candidates for deployment in downstream applications.

Debiasedness of a click model In this work, we introduce the notion of *debiasedness* of a click model w.r.t. the logging policy, the concept that the inferred relevance of a newly trained click model should not be correlated with the relevance predictions of the policy that was used to collect the training data, beyond the true relevance signal. First, we prove that debiasedness is a necessary condition (i) for obtaining consistent and unbiased estimations of document relevance, and (ii) for the invariance of click prediction under policy shift. Secondly, we present *conditional mutual information with the logging policy* (CMIP), a method based on conditional independence testing, that measures the degree of debiasedness of a newly trained model with regard to the logging policy.

In our semi-synthetic experiments, we first reproduce the findings in (Oosterhuis, 2023b) on strong but narrow logging policies. Then, we verify, on a wide array of training configurations, that CMIP helps to predict the performance of models outside of their training distribution. Lastly, we show that off-policy selection strategies based on CMIP incur lower regret than those based on perplexity and nDCG only.

Contributions Our contributions can be summarized as follows:

- We introduce the notion of debiasedness of a click model and show that it is necessary for unbiasedness, consistency, and invariance under policy shift of click models.
- We propose CMIP, a metric using relevance annotations that measures debiasedness of a click model.
- We show in semi-synthetic experiments that CMIP improves predicting the downstream performance of click models as well as the regret of off-policy model selection strategies.

To support the reproducibility of this work, we release the code for this chapter[1] and a standalone implementation of our metric.[2] Below, we first introduce related work on click models and conditional independence testing (Section 5.2). Then, we present the current evaluation protocol for click models and its deficiencies (Section 5.3) before introducing the concept of debiased-

1 https://github.com/philipphager/sigir-cmip
2 https://github.com/philipphager/cmip

ness and our proposed metric (Section 5.4). We end by evaluating our metric in extensive semi-synthetic experiments (Section 5.5 and 5.6).

5.2 RELATED WORK

5.2.1 Click models and their evaluation

Click models emerged to model user behavior in web search (Craswell et al., 2008; Dupret and Piwowarski, 2008; Chapelle and Zhang, 2009; Liu et al., 2016). Early methods use probabilistic graphical models to encode assumptions about user behavior in order to disentangle the influence of the presentation of a search result and its intrinsic relevance. The examination hypothesis, for example, introduced with the position-based model (PBM) (Craswell et al., 2008), assumes that the user examine and perceive the document as relevant in order to click on it. The cascade model (Craswell et al., 2008) assumes that users browse results from top to bottom, click on the first relevant result and then leave the page. For an overview of common click models, see (Chuklin et al., 2015).

More recently, click models based on neural architectures have emerged (Borisov et al., 2016; Borisov et al., 2018; Zheng et al., 2019; Lin et al., 2021; Chen et al., 2020; Dai et al., 2021) to model more complex browsing behavior (Borisov et al., 2018) and user preferences across sessions (Chen et al., 2020; Lin et al., 2021). Neural click models typically also use more expressive representations of queries, documents, and other meta-data (Chen et al., 2020; Borisov et al., 2018; Lin et al., 2021). Combined with recent optimization techniques, these models enable efficient training on large-scale click logs. In this work, we use three click models originally proposed as probabilistic graphical models and implement them with current gradient-based optimization techniques. We also include two neural click models, and two baselines based on click statistics. These models are presented in Section 5.5.2.

Click models are commonly evaluated using the log-likelihood of clicks in a test set, measuring how well a model approximates the observed data (Grotov et al., 2015). Craswell et al. (2008) evaluate models by measuring cross-entropy. More widely used nowadays is the perplexity metric, which measures how surprised a model is to observe a click on a given document and rank (Dupret and

Piwowarski, 2008). Another line of work compares predicted and actual click-through-rates (CTRs) on a test set, typically using MAE or RMSE (Chapelle and Zhang, 2009; Zhu et al., 2010; Grotov et al., 2015). Dai et al. (2021) introduced distributional coverage, a metric quantifying whether the distribution of click sequences predicted by a model matches the true distribution of clicks. Moving beyond click prediction, Chapelle and Zhang (2009) evaluate the ranking performance of click models by computing retrieval metrics (e.g., MAP or nDCG) on an additional test set of human relevance annotations. We introduce perplexity, nDCG, and their limitations in Section 5.3.2.

5.2.2 Conditional independence testing

This work introduces the concept of debiasedness, which requires a test for conditional independence (Dawid, 1979). Given a set of three random variables X, Y, and Z, conditional independence assesses if, given Z, knowing X is helpful for predicting Y (and vice versa). Conditional independence tests are widely applied in statistics and causal inference, e.g., to verify edges in Bayesian networks (Koller and Friedman, 2009), to discover causal graphs (Pearl, 2009), or for feature selection (Koller and Sahami, 1996).

We require a non-parameteric conditional independence test for continuous random variables. Approaches include binning continuous variables to apply tests for discrete data (Margaritis, 2005), reframing the problem as measuring the distance between two conditional densities (Su and White, 2007), estimating conditional mutual information (Mukherjee et al., 2019), or using kernel-based methods (Fukumizu et al., 2004; Doran et al., 2014). We use methods from (Sen et al., 2017; Mukherjee et al., 2019) to estimate conditional mutual information. Their approach is inspired by the use of model-powered independence testing (Lopez-Paz and Oquab, 2017; Sen et al., 2017; Mukherjee et al., 2019), reformulating statistical tests as supervised learning problems, which can be solved using standard classification or regression models. We introduce the approach we follow in Section 5.4.3.

5.3 BACKGROUND

5.3.1 Notation and assumptions

Notation. Let $d \in D$ be a document. A ranking y is an ordered list of documents: $y = [d_1, d_2, \ldots, d_K]$. Note that y is a list of length K and $y[k]$ is the document displayed at position k. We retrieve the position of a document in y using $\text{rank}(d \mid y)$. A policy π serves a ranking y in response to a search query q. We consider stochastic ranking policies $\pi(y \mid q)$, which are probability distributions over rankings, given a query. For each ranking displayed to a user, we observe a vector of binary feedback c of length K, with each entry denoting a click or no click on the displayed item: $c[k] \in \{0, 1\}$. Thus, our final dataset contains observations of a user query, the displayed ranking, and the recorded clicks: $\mathcal{D} = \{(q_i, y_i, c_i)\}_{i=1}^{N}$. The production ranker that collects this training dataset is commonly called the logging policy, which we denote as π_l.

Assumptions We follow a common assumption in L2R, that user clicks are a noisy and biased indicator of how relevant a document is to a given query (Joachims et al., 2017; Oosterhuis, 2022). We denote the relevance of an item to a query as $r(d, q)$. As explained before, clicks are usually influenced by bias factors such as the item's position or the user's trust in the system. Depending on the specific click model, bias factors can depend only on the position of a document or even on other documents in the same ranking. We refer to the vector of bias factors for documents in a given ranking as $o(y)$.

Our theory considers the family of click models that follow the structure of the examination hypothesis (Craswell et al., 2008; Chuklin et al., 2015), which assumes that to be clicked, a document has to be observed by a user and deemed as relevant. In a general form, the examination hypothesis demands that relevance r and bias factors o factorize as:

$$\forall k \in \{1, \ldots, K\}, \;\; P(c[k] = 1 \mid y, q) = r(y[k], q) \times o(y)[k]. \tag{5.1}$$

This generic formulation of the examination hypothesis, also used in (Zhuang et al., 2021), can account for users observing an item based on its position or even based on the relevance of surrounding documents. For simplicity, we assume that $o(y)[k] > 0, \forall y \in \mathcal{D}, k \in \{1, \ldots, K\}$, excluding any bias that leads to an item having no chance of being clicked (Joachims et al., 2017), such as

item selection bias. However, our work can be extended to this case. Lastly, our discussions below consider only a single query q to simplify our notation. All statements can be extended to a setting with multiple queries.

5.3.2 Evaluating click models

Click models are trained on an objective quantifying the quality of their click prediction. However, their primary goal, arguably, is to recover accurate estimates of the latent factors of user feedback. Hence, the literature on click models has adopted metrics for both of these objectives, respectively perplexity (PPL) and normalized discounted cumulative gain (nDCG).

The click prediction quality on a test set is measured by the perplexity at each rank k and by the average perplexity over all ranks:

$$\text{PPL@}k = 2^{-\frac{1}{N}\sum_{(y,c)\in\mathcal{D}} c[k]\log_2 \tilde{c}[k] + (1-c[k])\log_2(1-\tilde{c}[k])}, \tag{5.2}$$

$$\text{PPL} = \frac{1}{K}\sum_{k=1}^{K} \text{PPL@}k, \tag{5.3}$$

where $\tilde{c} = P(c \mid y)$ is a vector of click probabilities predicted by the model for a ranking y. Perplexity measures how surprised a model is to observe a given click behavior in the test set, given the model's parameters (Dupret and Piwowarski, 2008). Perplexity is at least one and can be arbitrarily high. However, since a model predicting clicks at random has a perplexity of two, a realistic click model should achieve a perplexity between one and two (Chuklin et al., 2015).

The quality of relevance estimates $\tilde{r}$ for documents is measured by the ranking metric nDCG, comparing predicted relevance scores against human annotations of relevance:

$$\text{nDCG} = \frac{\text{DCG}(\tilde{y})}{\text{DCG}(y^{\text{true}})}, \text{ with } \text{DCG}(y) = \sum_{k=1}^{K} \frac{2^{r(y[k])} - 1}{\log_2(k+1)}, \tag{5.4}$$

where $y^{\text{true}} = \arg\text{sort}^{\downarrow}_{d\in y}\, r(d)$ and $\tilde{y} = \arg\text{sort}^{\downarrow}_{d\in y}\, \tilde{r}(d)$ are obtained by ranking documents in order of relevance, as predicted by human annotators and the click model, respectively.

These two metrics are complementary in the sense that perplexity quantifies the goodness-of-fit of the model to the logged data while nDCG quantifies the

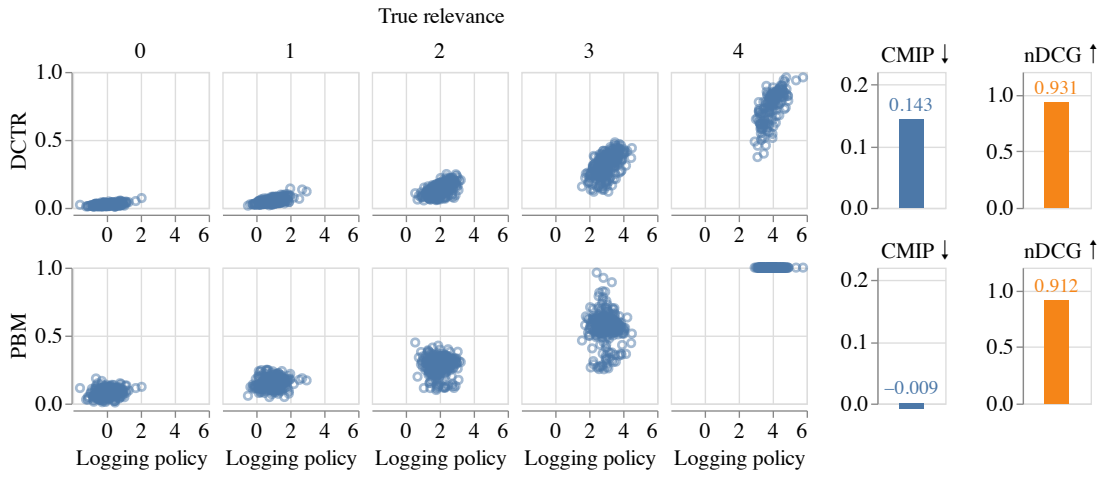


Figure 5.1: Comparing the relevance estimates of two click models (DCTR and PBM) against the relevance estimates of an almost optimal logging policy (NoisyOracle, defined in Section 5.5.1) for 1.5k documents, grouped by their true relevance. Clicks follow a PBM user model. The DCTR model achieves a higher nDCG but correlates notably with the logging policy, resulting in a high CMIP. In contrast to the PBM, the DCTR model is not debiased in this setup. Note that CMIP is in theory a non-negative metric but approximations can make it slightly negative.

quality of the rankings produced by the recovered relevance estimates. However, as we recall below, previous work has warned about the poor generalizability of these two metrics in many practical scenarios.

5.3.3 *Perplexity fails to generalize, especially under model misfit*

Perplexity measures how well a model fits the conditional distribution of clicks given rankings observed in the dataset. However, the performance measured by perplexity only holds on a separate test set as long as the i.i.d. assumption is satisfied (Shen et al., 2021), which notably requires that the rankings in the test set are sampled from the same distribution as rankings in the training set. This assumption is often violated when using click models to predict and evaluate ranking policies that differ from the one used for training (e.g., (Chen and Yan, 2012; Joachims et al., 2017)). This mismatch creates a covariate shift in the input distribution of the model. In such cases, no guarantee can be derived on the out-of-distribution performance of the trained models, and we are likely to observe a drop in performance.

Deffayet et al. (2023b) show the empirical effect of covariate shift on click model prediction. Model misfit, searching within model classes that do not contain the Bayes-optimal function, likely aggravates the performance drop on click predictions out-of-distribution. Model misfit is not the only cause for the lack of robustness. E.g., deep neural networks, that have high capacity and potentially include the Bayes-optimal predictor, can also suffer from covariate shift due to training sets covering only a narrow space of all possible rankings. *Hence, PPL measured in-distribution is not a good predictor of click model performance out-of-distribution in many practical scenarios, including model misfit and narrow logging policies.*

5.3.4 nDCG fails to generalize when the logging policy is good

nDCG assesses the ordering of documents based on their predicted relevance scores. As a list-wise metric, nDCG does not evaluate the accuracy of the estimated relevance probabilities but only how ordering by these relevance estimates correlates with rankings obtained through expert annotations. This ranking task is a use case for click models, but many scenarios require accurate estimation of relevance and examination probabilities, such as off-policy evaluation (Oosterhuis and de Rijke, 2020b), counterfactual learning-to-rank with inverse propensity scoring (Agarwal et al., 2019), or click maximization in reinforcement learning (Slivkins, 2019). Performance measured by nDCG can be misleading for these tasks since highly biased and poorly predictive click models can obtain high nDCG scores (Deffayet et al., 2023b). Indeed, when the logging policy already achieves a high nDCG, one cannot use nDCG to differentiate between a model predicting accurate relevance probabilities from a model replicating the logging policy, e.g., by sorting documents by their number of impressions. In more realistic scenarios, this misleading behavior of nDCG might manifest itself only for a group of queries (e.g., tail queries), enabling a model to achieve an improved nDCG score at the cost of biased relevance estimates for these queries. *Consequently, nDCG is not a good predictor of click model debiasing capabilities in many realistic settings.*

Faced with the lack of metrics evaluating the robustness of click models to shifts in the input rankings, we propose the idea of measuring debiasedness in the next section.

5.4 TOWARDS HEALTHY BENCHMARKS: A METRIC TO QUANTIFY DEBIASEDNESS

To establish CMIP, which measures the robustness of click models to covariate shift, we first introduce the notion of *debiasedness* in Section 5.4.1. We then explain how to test for this property in Section 5.4.2, and finally instantiate our proposed metric in Section 5.4.3.

5.4.1 Debiasedness in click modeling

Let $\mathcal{R}$ be a set containing three relevance scores for each document: $\mathcal{R} = \{(\tilde{r}(d), r_l(d), r(d))\}_{d \in \mathcal{D}}$, where r is the true relevance as annotated by human experts and r_l the relevance estimate of the logging policy, which we define as the expected rank of a document: $r_l(d) = \mathbb{E}_{y \sim \pi_l}[\text{rank}(d \mid y)]$. We assume this quantity to be known in our experiments but it can easily be estimated otherwise. Finally, $\tilde{r}$ is a set of relevance scores as estimated, e.g., by a new model.

We consider the following random experiment. We draw triplets of relevance scores from $\mathcal{R}$ at random and with replacement, revealing a document's relevance scores but not the document itself. We write R, R_l and $\tilde{R}$ for the random variables that return the respective relevance scores over all documents.[3] We can now define the debiasedness of a set of scores $\tilde{r}$:

Definition 2. *A set of scores $\tilde{r}$ is* debiased w.r.t. the logging policy *if its corresponding random variable $\tilde{R}$ is independent of the relevance of the logging policy R_l, conditioned on the true relevance R:*

$$\tilde{R} \perp\!\!\!\perp R_l \mid R. \tag{5.5}$$

Intuitively, debiasedeness means that the score of a document cannot be predicted by knowing where the logging policy placed it. Click models aim to disentangle the factors influencing user behavior, thereby also alleviating biases induced by the logging policy. Therefore, a natural property that we may expect of a well-behaved click model is that its estimated relevance of an item cannot be predicted by revealing the relevance of that same item according to the logging policy, i.e., *debiasedness*:

3 In the remainder, we will use a set and its corresponding random variable interchangeably.

Definition 3. *A click model is* debiased w.r.t. the logging policy *if its estimated relevance after training is independent of the relevance of the logging policy, conditionally on the true relevance as well as the dataset it has been trained on:*

$$\tilde{R}^{\mathcal{D}} \perp\!\!\!\perp R_l \mid (R, \mathcal{D}), \tag{5.6}$$

where $\tilde{R}^{\mathcal{D}}$ is the set of relevance scores estimated by a click model trained on dataset $\mathcal{D}$.

We give a visual intuition of debiasedness and our proposed metric CMIP in Figure 5.1. We display the relevance estimates obtained by a PBM click model and a model predicting the average CTR of each document (DCTR) as relevance. In the plot, we group all documents by their true annotated relevance. In contrast to the PBM, the DCTR model does not account for position bias simulated in the click data. We can observe a clear correlation of the relevance estimates of the DCTR model with those of the logging policy, meaning we can predict the estimated relevance of a randomly drawn document by knowing where the logging policy placed it. Thus, the DCTR model is not debiased in this setting, which is captured in a higher score of CMIP.

Debiasedness alone does not guarantee high performance of a click model, e.g., a model assigning random relevance scores is trivially debiased. Below, we show that debiasedness is a necessary condition for common goals in unbiased learning-to-rank.

Debiasedness is a necessary condition for both consistency and unbiasedness Unbiasedness has been introduced in different subfields as a common goal of unbiased learning-to-rank (Oosterhuis, 2022; Joachims et al., 2017). Extending the meaning of unbiasedness from estimators to click models, *an unbiased click model recovers the true relevance parameters for each document, in expectation over possible training datasets*. Using $\tilde{r}^{\mathcal{D}}(d)$ to denote the relevance of document d, as predicted by a click model after being trained on dataset $\mathcal{D}$, a click model is unbiased if, and only if:

$$\mathbb{E}_{\mathcal{D} \sim \pi_l} \left[\tilde{r}^{\mathcal{D}}(d) \right] = r(d) \ , \tag{5.7}$$

where we use the notation $\mathcal{D} \sim \pi_l$ to illustrate that the dataset is generated by users interacting with the logging policy. Consistency has been introduced more recently (Oosterhuis, 2022), as a more attainable goal for click models. A *consistent* click model recovers the true relevance parameters in the limit of infinite data:

$$\lim_{|\mathcal{D}| \to \infty} \tilde{r}^{\mathcal{D}}(d) = r(d) \ . \tag{5.8}$$

Since the relevance scores recovered by a consistent click model in the limit of infinite data are equal to the true relevance R, a consistent click model is trivially debiased: $R \perp\!\!\!\perp R^{\pi_l} \mid R$. Similarly, the expected relevance scores of an unbiased click model are debiased since $\mathbb{E}_{\mathcal{D}\sim\pi_l}[\tilde{R}^{\mathcal{D}}] = R$.

Debiasedness is a necessary condition for invariance under policy shift In this work, we evaluate the click prediction capabilities of click models outside of their training distribution. Therefore, we define the notion of invariance under policy shift as a model predicting the same click probabilities for a document, regardless of the dataset it was trained on:

Definition 4. A click model is said to be *invariant under policy shift* if its estimated click probabilities are the same regardless of the training set, i.e., for every ranking y and any two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$:

$$\tilde{c}^{\mathcal{D}_1}(y) = \tilde{c}^{\mathcal{D}_2}(y) = \tilde{c}(y), \tag{5.9}$$

where $\tilde{c}^{\mathcal{D}}(\cdot)$ are the click predictions obtained after training a click model on $\mathcal{D}$.

This definition allows us to introduce our main theorem that for every click model following the examination-hypothesis (Eq. 5.1), *debiasedness is a necessary condition for invariance under policy shift*:

Theorem 1. *A click model that is invariant under policy shift is debiased. For every dataset $\mathcal{D}$ and ranking y:*

$$\tilde{c}^{\mathcal{D}}(y) = \tilde{c}(y) \Rightarrow \tilde{R}^{\mathcal{D}} \perp\!\!\!\perp R_l \mid (R, \mathcal{D}). \tag{5.10}$$

Proof. First, a model such that $\tilde{r}^{\mathcal{D}} = 0$ is debiased since $0 \perp\!\!\!\perp R_l \mid (R, \mathcal{D})$, and therefore satisfies Eq. 5.10. Next, assume $\tilde{r}^{\mathcal{D}} \neq 0$ in the remainder. For a model following the examination-hypothesis, we can write, for any rank k, ranking y and training dataset $\mathcal{D}$:

$$c^{\mathcal{D}}(y)[k] = r^{\mathcal{D}}(y[k]) \times o^{\mathcal{D}}(y)[k].$$

Consider two documents d_1 and d_2 with $\tilde{r}^{\mathcal{D}}(d_2) \neq 0$. Consider also two rankings y_1, y_2 that differ only by their first document: $y_1[1] = d_1$, $y_2[1] = d_2$, and $y_1[k] = y_2[k]$ for $k > 1$. Let these rankings share the same examination probability on their first position: $\tilde{o}^{\mathcal{D}}(y_1)[1] = \tilde{o}^{\mathcal{D}}(y_2)[1]$, then we can write down the two click probabilities:

$$\begin{cases} \tilde{c}^{\mathcal{D}}(y_1)[1] = \tilde{r}^{\mathcal{D}}(d_1) \times \tilde{o}^{\mathcal{D}}(y_1)[1] \\ \tilde{c}^{\mathcal{D}}(y_2)[1] = \tilde{r}^{\mathcal{D}}(d_2) \times \tilde{o}^{\mathcal{D}}(y_2)[1] \end{cases}.$$

Because of the equality of examination probabilities, we have:

$$\frac{\tilde{r}^{\mathcal{D}}(d_1)}{\tilde{r}^{\mathcal{D}}(d_2)} = \frac{\tilde{c}^{\mathcal{D}}(y_1)[1]}{\tilde{c}^{\mathcal{D}}(y_2)[1]} \overset{\text{(LHS)}}{=} \frac{\tilde{c}(y_1)[1]}{\tilde{c}(y_2)[1]} .$$

When the left-hand side of Eq. 5.10 is true, this ratio does not depend on π_l and relevance scores are determined up to a document-independent constant, so knowing R_l does not help predict the relevance of a newly picked document: $P(\tilde{R}^{\mathcal{D}} \mid \mathcal{D}, R) = P(\tilde{R}^{\mathcal{D}} \mid \mathcal{D}, R, R_l)$. Thus, the model is debiased: $\tilde{R}^{\mathcal{D}} \perp\!\!\!\perp R_l \mid (R, \mathcal{D})$. □

A key observation is that the estimated relevance of a document may depend on the training dataset and ultimately on the logging policy, yet revealing $R_l \mid (R, \mathcal{D})$ might not help to predict $\tilde{R}^{\mathcal{D}} \mid (R, \mathcal{D})$. Take, for example, the PBM model. Without explicit constraints, we can scale its inferred relevance scores up or down, and by adjusting the examination scores accordingly, the click probabilities stay constant. Then a perfectly fitted PBM may be invariant under policy shift, while the exact set of parameters it recovers depends on the training dataset. However, for a fixed dataset, knowing R_l does not help to predict $\tilde{R}^{\mathcal{D}}$ in this setting.

In conclusion, while debiasedness alone does not guarantee that a click model inferred the correct parameters during training, it is a necessary condition for consistency, unbiasedness, and invariance under policy shift. We, therefore, propose to systematically verify that this property holds, using a protocol based on relevance annotations, which we describe in the following sections.

5.4.2 Testing for debiasedness with mutual information

Given a set of annotated documents and the expected rank of these documents under the logging policy, we can test for the debiasedness of a candidate click model using any conditional independence test for continuous variables. Using independence tests, given some significance level, would yield a binary answer to whether the click model is debiased. In practice, however, we may be more interested in picking the best model from a set of candidates for deployment, i.e., off-policy selection (OPS). For example, which model should we select if multiple candidates are debiased, or none are? Therefore, we choose

to quantify the degree of debiasedness using the effect size of an independence test: conditional mutual information (CMI).

We first recall the concept of mutual information (MI), which measures the average reduction in uncertainty of a random variable X obtained when knowing the value of a second random variable Y. Mutual information, usually expressed as $I(X;Y)$, can capture non-linear relationships between variables. Conditioning the mutual information between two variables on a third variable is strongly connected to conditional independence testing:

$$\tilde{R} \perp\!\!\!\perp R_l \mid R \iff I(\tilde{R}; R_l \mid R) = 0\,, \tag{5.11}$$

where $\tilde{R}$ are the relevance scores predicted by a click model, R_l the implicit relevance scores of the logging policy, and R the human annotations of relevance. This means that, conditional on R, knowing R_l does not reduce the uncertainty of predicting $\tilde{R}$ and vice-versa. Thus, conditional independence testing is a special use case of CMI and we can interpret a lower value of CMI as a higher degree of debiasedness, with a CMI of zero indicating that the relevance scores of a newly trained click model are independent of the policy that collected the dataset, conditional on the true relevance. We refer to the CMI when computed w.r.t. the logging policy as CMIP.

5.4.3 *Estimating conditional mutual information with the logging policy (CMIP)*

In this section, we cover how to estimate the CMIP metric to quantify debiasedness. First, we note that CMI can be expressed as the Kullback-Leibler divergence between two distributions:

$$\begin{aligned} I(X;Y \mid Z) &= \mathcal{D}_{\mathrm{KL}}\left(p \| q\right) \\ &\text{with } p = P(\tilde{R}, R_l, R) \\ &\text{and } q = P(R)\; P(\tilde{R} \mid R)\; P(R_l \mid R)\,, \end{aligned} \tag{5.12}$$

which is a pseudo-distance between the joint distribution p of all three variables occurring together and the distribution q in which the predicted relevance scores $\tilde{R}$ and the relevance of the logging policies R_l are independent, conditional on R. If the divergence between both distributions is zero, the joint distribution (which we actually observe) is equivalent to the distribution on which conditional independence holds. Given this divergence-based formulation of CMI, we employ a two-step approach suggested in (Mukherjee et al.,

2019). First, we obtain samples from the marginal distribution q on which conditional independence holds. Second, we estimate the KL-divergence between the observed dataset and the generated samples, which is the estimate of our CMIP metric.

Sampling from the marginal distribution q

How can we obtain samples from the conditional independence distribution q given our observational dataset p? For a proof that this methodology actually approximates q, we refer to (Sen et al., 2017, Theorem 1). We use a knn-based approach suggested in (Sen et al., 2017); its simplicity and computational speed make it suitable for an evaluation metric. Given a dataset of observed relevance labels for each document, $\mathcal{R} = \{(\tilde{r}(d), r_l(d), r(d))\}_{d \in \mathcal{D}}$, we split the data into two equal parts $\mathcal{R}_i$ and $\mathcal{R}_j$. For each document in $\mathcal{R}_i$, we find the nearest neighbor document in $\mathcal{R}_j$ with the most similar true relevance. In the case of using relevance annotations, this method simplifies to sampling any document from $\mathcal{R}_j$ with the same relevance label. By exchanging the relevance estimates of the logging policy between the two documents, the resulting dataset $\mathcal{R}_q = \{(\tilde{r}(d_i), r_l(d_j), r(d_i))\}$, is now a sample from q.

Estimating KL-divergence

Given samples from the original relevance dataset $\mathcal{R}_p \sim p$ and samples from the marginal distribution $\mathcal{R}_q \sim q$, we can compute the CMI as the KL-divergence between both distributions. We follow Mukherjee et al. (2019) and frame the task of divergence estimation between two continuous joint distributions as a binary classification problem. The main idea is to label samples from p with $m = 1$ and samples from q with $m = 0$. After shuffling the two datasets into one, we train a binary classifier to predict to which distribution a given triplet of relevance values $(\tilde{r}, r_l, r)$ belongs to. The better the classifier can assign samples to their original distribution, the higher the divergence between the two distributions. Using the Donsker-Varadhan reformulation of KL-divergence (Mukherjee et al., 2019, Definition 3), we use the classifier's pre-

dictions of $P(m = 1)$ on a test set to compute the *conditional mutual information with the logging policy* (CMIP) as:

$$\begin{aligned} &\text{CMIP} = \mathcal{D}_{\text{KL}}\left(p \parallel q\right) \\ &\approx \frac{1}{|\mathcal{R}_p|} \sum_{i \in \mathcal{R}_p} \log \frac{P(m = 1 \mid i)}{P(m = 0 \mid i)} - \log \left(\frac{1}{|\mathcal{R}_q|} \sum_{j \in \mathcal{R}_q} \frac{P(m = 1 \mid j)}{P(m = 0 \mid j)} \right). \end{aligned} \tag{5.13}$$

The above procedure requires a well-calibrated classifier and we clip predictions $P(m = 1) \in [0.01, 0.99]$ to avoid extremely large likelihood ratios when dividing by predictions close to zero. Lastly, we bootstrap the metric, performing multiple repetitions of k-nn sampling and KL divergence estimation, reporting the average over five repetitions. In order to simplify the usage of our metric, we release a standalone implementation of CMIP.[4]

5.5 EXPERIMENTAL SETUP

We test if CMIP helps to predict which click models are robust to covariate shift by performing experiments using a semi-synthetic click simulation setup prevalent in unbiased learning-to-rank (Joachims et al., 2017; Oosterhuis and de Rijke, 2020a; Vardasbi et al., 2020a; Vardasbi et al., 2020b). The setup is semi-synthetic since we generate synthetic user clicks on real search queries and documents. To simulate shifts in the ranking distribution, we train models on click data collected under one logging policy and evaluate the model on clicks obtained under a different policy. Below, we introduce our simulation setup and the click models used in our experiments.

5.5.1 Semi-synthetic click simulation

Overview

We generate click datasets by repeatedly: (i) sampling a query and its candidate documents from a preprocessed real-world dataset; (ii) sampling a ranking of the candidate documents using a stochastic logging policy; and (iii) presenting the ranked search results to a synthetic user model to sample clicks. In the following, we cover each step in more detail.

4 https://github.com/philipphager/cmip

Dataset and preprocessing

Our click simulation is based on the MSLR-WEB10K dataset (Qin and Liu, 2013). We use the training dataset of the first fold, containing 6,000 search queries, each with a set of candidate documents. Each query-document pair was judged by experts on a five point relevance scale: $r(d) \in \{0, 1, 2, 3, 4\}$, which we use as ground-truth in our experiments. During preprocessing, we reduce the number of documents per query to ten using stratified sampling on the human relevance annotation. Thereby, we reduce the number of candidate documents, while keeping a similar distribution of relevance grades. After discarding all queries with less than ten documents, we obtain a total of 5,888 queries.

Stochastic policies

For each new simulated user session, we first pick a query from the preprocessed dataset uniformly at random, a common practice in simulation for unbiased learning-to-rank (Joachims et al., 2017), to avoid high variance on rare queries in this study. After sampling a query, we sample rankings of the candidate set of documents. For that, we first obtain relevance estimates for each document using one of three policies of different quality:

UNIFORM: A policy assigning the same relevance to all documents.

LAMBDAMART: A LambdaMART ranker (Burges, 2010) trained on feature vectors and relevance annotations provided in MSLR-WEB10K.[5]

NOISYORACLE: A near-optimal policy using perturbed human relevance annotations after adding Gaussian noise of variance 0.5.

After using one of these three policies to obtain relevance estimates for each document, we sample stochastic rankings using a Plackett-Luce model (Plackett, 1975; Luce, 1959). We sample multiple rankings per query to observe documents in different positions since a deterministic ranking would not allow our click models to disentangle effects such as position bias or relevance during training. We use the Gumbel Softmax trick to efficiently sample rankings from a Plackett-Luce distribution (Bruch et al., 2020; Oosterhuis, 2021a) and control the degree of stochasticity in the sampled rankings using the temperature parameter of the softmax. We sample rankings with a low degree of stochasticity in our experiments, using a temperature of $T = 0.1$ by default.

5 LightGBM version 3.3.2, using 100 trees, 31 leafs, and learning rate 0.1.

User models

After sampling rankings, we generate synthetic clicks on our documents. We define how relevant each document is to the synthetic user based on the expert relevance annotations (Chapelle et al., 2009): $R_d = \epsilon + (1 - \epsilon) \frac{2^{r(d)} - 1}{2^4 - 1}$, with noise $\epsilon = 0.1$ to also sample clicks on irrelevant documents. To examine our metric under a variety of click behaviors, we simulate four different users:

PBM: A user behaving according to the examination hypothesis, clicking only on observed and relevant documents. The observation probability depends only on the document position (Craswell et al., 2008). Following (Joachims et al., 2017), we define the observation probability at rank k as: $O_k = \frac{1}{k}$.
DBN: A user for whom relevance is split into two concepts: attractiveness and satisfaction. Attractiveness measures how likely a user is to click on a document after observing it in the ranking, while satisfaction estimates how likely a user is satisfied with the document after opening it. Documents are examined from top to bottom until the user is satisfied or abandons the list (Chapelle and Zhang, 2009). Thus, examination of a document not only depends on its rank, but also on the documents examined before. We use the probability of relevance R_d to define the attractiveness of a document as $A_d = R_d$ and its satisfaction as $S_d = \frac{R_d}{2}$, so that even on near-optimal policies, fulfilling a user's information need sometimes requires more than one click.
MIXTUREDBN: This setting simulates platforms presenting results horizontally, where users may not inspect the document in order (Borisov et al., 2018). This mixture is composed of two DBN users: 70% of the time, the user inspects the results in the usual order from first to last rank, but 30% of the time, the user inspects the ranking in reverse order.
CAROUSEL: This last user simulates settings in which documents are grouped in vertical rows or carousels (Rahdari et al., 2022). For this specific setting only, we use 25 documents per query instead of ten. The user chooses one of five carousels, according to a PBM, where the relevance of a carousel is taken to be the average relevance of the five documents that compose it. Then the user clicks on documents within the chosen carousels according to a DBN.

Next, we introduce the click models that we use, but we note that none fits the MixtureDBN and Carousel user behavior. We include these complex click behaviors to evaluate the usefulness of our metric under model mismatch.

5.5.2 Click model overview

We compare seven click models in our simulations; two are naive baselines using click statistics; five are prominent click models from the literature. In line with earlier studies, the models we compare do not input document features beyond the document's id. Below, we only summarize the main idea for each approach. For details, we refer to (Deffayet et al., 2023b), which we follow closely in our implementation.

Naive baselines

DCTR: The document CTR model uses the mean click-through-rate of a document as both click and relevance prediction. Since the CTR is averaged over all document positions, this model naively assumes that users examine all ranks equally.

RDCTR: The ranked document CTR model predicts the mean click-through-rate of a document at a given rank as click probability. We follow (Deffayet et al., 2023b, Eq. 3) and estimate relevance as the sum of a document's CTR at each rank, weighted by the inverse of the average CTR of all documents at the given rank.

In both methods, rarely examined documents can cause extreme click predictions, such as predicting a click probability of zero for a document that was never clicked. To mitigate predictions that lead to arbitrarily high perplexity values, we use the empirical Bayes method and initialize each prediction with Beta priors estimated on our training data as suggested in (Chapelle and Zhang, 2009).

Click models

We implement a **PBM** and a **DBN** click model matching the user behaviors introduced in Section 5.5.1. In addition, we implement three other models:

UBM: Extending the PBM, the user browsing model assumes that examining an item depends in addition to its position also on the position of the latest clicked document (Dupret and Piwowarski, 2008).

NCM: The neural click model uses an RNN to iterate over the list of documents and predicts clicks at every step. While the model only predicts clicks and does not explicitly model relevance, Borisov et al. (2016) suggest to use the click probability of an item when placed on top of a ranking as its relevance.

Table 5.1: Adjusted R^2 score of predicting the out-of-distribution perplexity using combinations of in-distribution perplexity, nDCG, and CMIP using regression (higher is better). We mark significantly higher ▲ or lower performance ▼ compared to using *ind PPL, nDCG* at a significance level of $\alpha = 0.0001$. Adding CMIP improves predictions of ood PPL, achieving the best performance when combining all three metrics, and combinations including CMIP are more consistent across different experimental setups and covariate shifts.

User model	**Logging policy**	**Test policy**	ind PPL	nDCG	CMIP	**ind PPL, nDCG**	CMIP, ind PPL	CMIP, nDCG	Joint
PBM	NoisyOracle	LambdaMART	0.276	0.256	0.439	0.395	$0.938^{\blacktriangle}$	$0.972^{\blacktriangle}$	$0.963^{\blacktriangle}$
		Uniform	0.277	0.346	$0.725^{\blacktriangle}$	0.304	$0.966^{\blacktriangle}$	$0.969^{\blacktriangle}$	$0.970^{\blacktriangle}$
	LambdaMART	NoisyOracle	$0.369^{\blacktriangledown}$	0.907	0.904	0.861	0.883	$0.962^{\blacktriangle}$	$0.944^{\blacktriangle}$
		Uniform	$0.347^{\blacktriangledown}$	$0.965^{\blacktriangle}$	0.931	0.889	$0.974^{\blacktriangle}$	$0.977^{\blacktriangle}$	$0.975^{\blacktriangle}$
DBN	NoisyOracle	LambdaMART	$0.235^{\blacktriangledown}$	$0.978^{\blacktriangle}$	$0.517^{\blacktriangledown}$	0.693	$0.787^{\blacktriangle}$	$0.977^{\blacktriangle}$	$0.897^{\blacktriangle}$
		Uniform	$0.254^{\blacktriangledown}$	$0.984^{\blacktriangle}$	$0.545^{\blacktriangledown}$	0.688	$0.821^{\blacktriangle}$	$0.983^{\blacktriangle}$	$0.963^{\blacktriangle}$
	LambdaMART	NoisyOracle	$0.941^{\blacktriangledown}$	$0.997^{\blacktriangle}$	$0.997^{\blacktriangle}$	0.974	0.970	$0.997^{\blacktriangle}$	0.983
		Uniform	$0.958^{\blacktriangledown}$	$0.999^{\blacktriangle}$	$0.999^{\blacktriangle}$	0.983	0.982	$0.999^{\blacktriangle}$	0.982
MixtureDBN	NoisyOracle	LambdaMART	$0.805^{\blacktriangledown}$	$0.149^{\blacktriangledown}$	0.843	0.869	$0.910^{\blacktriangle}$	$0.906^{\blacktriangle}$	$0.914^{\blacktriangle}$
		Uniform	$0.618^{\blacktriangledown}$	$0.266^{\blacktriangledown}$	$0.859^{\blacktriangle}$	0.767	$0.884^{\blacktriangle}$	$0.909^{\blacktriangle}$	$0.880^{\blacktriangle}$
	LambdaMART	NoisyOracle	0.985	$0.167^{\blacktriangledown}$	$0.625^{\blacktriangledown}$	0.988	0.990	$0.862^{\blacktriangledown}$	0.985
		Uniform	$0.951^{\blacktriangledown}$	$0.167^{\blacktriangledown}$	$0.735^{\blacktriangledown}$	0.987	0.984	$0.902^{\blacktriangledown}$	0.987
Carousel	NoisyOracle	LambdaMART	0.974	$0.811^{\blacktriangledown}$	$0.993^{\blacktriangle}$	0.970	0.977	0.978	0.973
		Uniform	0.954	0.941	$0.994^{\blacktriangle}$	0.947	0.968	$0.995^{\blacktriangle}$	0.942
	LambdaMART	NoisyOracle	0.993	$0.129^{\blacktriangledown}$	$0.688^{\blacktriangledown}$	0.990	0.991	$0.773^{\blacktriangledown}$	0.994
		Uniform	0.993	$0.113^{\blacktriangledown}$	$0.722^{\blacktriangledown}$	0.996	0.994	$0.759^{\blacktriangledown}$	0.998
Average			0.683	0.574	0.782	0.831	0.939	0.933	0.959

CACM: We implement another RNN-based model iterating over rankings; instead of predicting clicks it predicts the user's probability of examination at each rank. The resulting examination probability is multiplied with an estimated relevance probability to obtain the click prediction. The model is a variant of the context-aware click model introduced in (Chen et al., 2020) as proposed in (Deffayet et al., 2023b, Eq. 14).

All models are implemented using PyTorch, and are trained by minimizing a binary cross-entropy loss between the predicted clicks and the observed clicks in the training dataset. Further implementation details are openly accessible in our code.[6]

5.5.3 *Experiments*

In our experiments, we evaluate whether CMIP helps to predict the performance of click models under covariate shift. Therefore, we first generate 5M training, 1M validation, and 1M test clicks on a strong baseline policy (LambdaMART or NoisyOracle). We use the training/validation sets to train models and the test set to compute the in-distribution perplexity (ind PPL). We simulate a covariate shift with a second test set of 1M clicks generated by a different policy, called test policy, and report the out-of-distribution perplexity (ood PPL). Lastly, we use the human-annotated relevance labels from the MSLR-WEB10K dataset to compute nDCG and CMIP.

5.6 RESULTS

5.6.1 *Evaluation with CMIP: A visual example*

We introduce our experimental results by giving a visual intuition of CMIP. In Figure 5.2, we use a near-optimal logging policy (NoisyOracle) and generate clicks according to a PBM user model. We train seven click models and observe their in-distribution perplexity (ind PPL), nDCG, and our proposed metric CMIP, as well as the models' performance under policy shift as measured by the out-of-distribution perplexity (ood PPL). We can see that neither

6 https://github.com/philipphager/sigir-cmip

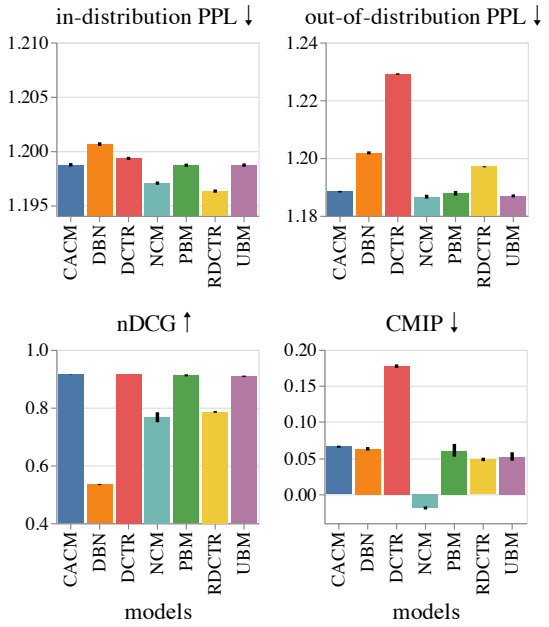


Figure 5.2: Comparing the performance of click models. Our proposed metric, CMIP, helps predict out-of-distribution results. All models are trained on a PBM user model and a NoisyOracle logging policy and evaluated under a uniform policy. We average results over ten independent runs and we display the 95% confidence interval.

ind PPL nor nDCG are sufficient to predict the downstream performance in ood PPL. E.g., the competitive nDCG score of the DCTR model fails to capture the model's poor generalizability. In contrast, CMIP correctly identifies this model as biased by assigning it a high value. Similarly, despite a low nDCG, NCM retains a good performance out-of-distribution as indicated by a low debiasedness score.

This visual example helps understand how CMIP can be generally useful to evaluate click models, but the results are dependent on the configuration used for training and evaluating the models. Below, we assess the predictive power of CMIP more systematically.

5.6.2 CMIP helps predict out-of-distribution perplexity

Next, we systematically evaluate whether adding CMIP to the existing metrics (ind PPL and nDCG) helps to predict the out-of-distribution perplexity of click models. To do so, we test click models in a total of 16 configurations, where

the user model, logging policy, and test policy vary. Moreover, we evaluate each model configuration over ten independent runs of our click simulation.

To quantify how well ind PPL, nDCG, CMIP, and their combinations predict ood PPL, we use the metrics as input to a decision tree regressor predicting ood PPL. Since the metric ranges depend on the policy and user model, we train separate regressors for each of the 16 user/policy configurations and seven metric combinations. We report predictive performance using the adjusted R^2 score on 2-fold cross validation with a thousand repetitions. We test differences between each metric combination and the current evaluation protocol of jointly using nDCG and ind PPL using a two-tailed Welch's t-test with a significance level of $\alpha = 0.0001$, accounting for multiple comparisons using the Bonferroni correction.

We report the results in Table 5.1. First, neither ind PPL, nDCG, nor CMIP are sufficient on their own to predict the ood PPL well, even though CMIP has a better mean predictive power than the other two metrics across all configurations. Second, using a combination of multiple metrics leads to better predictions of ood performance. However, we can observe that combining CMIP with any other of the two metrics leads to an average R^2 greater than 0.9 which is not attainable just using ind PPL and nDCG. More importantly, by inspecting the performance across configurations, we see that metric combinations including CMIP are notably more consistent across different configurations, with their R^2 score rarely dropping below 0.8. This suggests that CMIP improves the safety and reliability of click model evaluation for deployments in downstream tasks. Lastly, the joint usage of all three metrics is either significantly better or on-par with the usage of nDCG and ind PPL. These trends are consistent when using linear regression and other regression metrics such as MSE. Our results strongly indicate that adding CMIP to click modeling benchmarks should lead to more reliable predictions of downstream performance, and therefore help practitioners to mitigate the risks of deploying policies based on click models.

5.6.3 Strategies based on CMIP incur lower regret in off-policy selection problems

Next, we use CMIP in an off-policy selection (OPS) problem where we have a set of candidate click models and need to decide which one to use for downstream applications. To quantify how CMIP helps practitioners select the best

Table 5.2: Average regret incurred by different OPS strategies (lower is better). The regret is the difference in out-of-distribution click prediction performance with the best-performing model. We mark the best strategy in bold and underline the second best, and we report 95% confidence intervals inside parentheses.

Policies	Average regret
PPL↓	2.5499 ($\pm$0.0506)
nDCG↑	5.2406 ($\pm$0.8480)
top-4 nDCG: PPL↓	1.6423 ($\pm$0.1417)
top-4 PPL: nDCG↑	2.5199 ($\pm$0.3168)
CMIP↓	2.5268 ($\pm$0.2722)
top-4 CMIP: PPL↓	2.5505 ($\pm$0.0507)
top-4 CMIP: nDCG↑	<u>0.9493</u> ($\pm$0.1175)
top-4 CMIP, top-4 nDCG: PPL↓	1.6404 ($\pm$0.1430)
top-4 CMIP, top-4 PPL: nDCG↑	**0.9176** ($\pm$0.1221)

model, we design simple OPS strategies based on the three metrics and compare the amount of regret they incur, i.e., how much click prediction performance, measured by ood PPL, is lost by following a given strategy instead of selecting the optimal model. Every selection strategy is based on the maximization or minimization of a metric among the set of candidates. E.g., "nDCG↑" is the strategy that selects the model with the highest nDCG. In addition to the three basic strategies defined this way, we define conditional strategies: e.g., "top-4 CMIP: PPL↓" selects the model with the lowest perplexity among the four models with the lowest CMIP. The strategy "top-4 CMIP, top-4 PPL: nDCG↑" selects the model with the highest nDCG among the intersection of the four models with lowest CMIP and the four models with the lowest PPL. If this intersection is empty, the model with highest nDCG is selected.

Table 5.2 reports the average regret incurred by these strategies over the same configurations of user model, logging policy, and test policy as in Table 5.1. For better readability, we multiply the obtained regret in terms of difference in ood PPL by a factor 1000. Also, the regret in OPS is very sensitive to the exact set of candidates in the comparison, so in order to obtain more robust results, we apply each strategy on all possible combinations of five, six, or seven models from our set of seven candidates and report the average regret over these combinations. We observe that most strategies based on CMIP

outperform those without it, and that the lowest average regret is obtained by "top-4 CMIP, top-4 PPL: nDCG↑", confirming that CMIP is useful in off-policy selection problems.

5.7 CONCLUSION

We propose *conditional mutual information with the logging policy* (CMIP), an evaluation metric for click modeling benchmarks in unbiased learning-to-rank. CMIP addresses the problem that existing metrics do not ensure that click models are robust to shifts in the ranking policy and therefore fail to predict their performance on downstream tasks. CMIP evaluates how relevance scores of trained models correlate with relevance scores of the logging policy beyond the true relevance signal, i.e., measuring how biased a new model is by the model that was used to collect the training data.

Findings and broader impact We gave visual interpretations of CMIP and its use for selecting best-performing click models. We quantified its usefulness in click modeling benchmarks by showing that it (i) improves the prediction of downstream performance when coupled with existing metrics, and (ii) lowers the regret incurred by off-policy selection strategies. The effectiveness of CMIP suggests that distributional approaches to offline evaluation, i.e., that consider the distribution of model outputs instead of individual predictions, may be useful to derive generalization properties.

Limitations and future work First, CMIP uses pointwise relevance annotations, but pairwise or listwise annotations could also be used. Second, we have assumed that annotations are a perfect predictor of relevance. It remains unclear how to interpret nDCG and CMIP in case annotator disagreement and biases render annotations less reliable. Finally, click feedback collected on fully randomized rankings could replace the need for expert annotations; we leave an analysis of CMIP in that case for future work.

5.8 REFLECTIONS ON THE CHAPTER

5.8.1 Research outcomes

This chapter provides progress towards answering the following two research questions:

Research Question 2. *Can we predict in a fully offline manner the performance of models learned on biased data?*

Research Question 3. *When do we need assumptions on user behavior, and how can we test for the validity of these assumptions?*

The CMIP metrics brings us closer to answering RQ2 positively, and helps select the correct assumptions about user behavior that lead to bias in the data. At least, it provides a more reasonable selection of candidates for an online A/B test, than the ranking metrics alone. Now, multiple potential refinements and open questions remain: can we extend CMIP to content-based and context-aware models? How to port the CMIP approach to training de-biased models? Finally, as we showed in Chapter 3 that click models do not work as well with dynamic users, how robust is the metric when user behavior is dynamic, and preference must be estimated?

5.8.2 Additional thoughts

Beyond simply making the process of deploying click-model-based systems safer, I believe metrics like CMIP open an interesting avenue for model evaluation. Indeed, we learned in this chapter that point predictions of performance, i.e., comparing one prediction with one label and then aggregating these results in some way, give less information than distributional metrics like CMIP, i.e., metrics that compare the whole distribution of predictions to the distribution of labels. In particular, when we are dealing with issues such as bias, which are distributional in nature, distributional metrics offer a range of new features to measure: shape, modes, covariances, etc

Part III

Challenges of Reinforcement Learning for Recommender Systems

6

GENERATIVE SLATE RECOMMENDATION WITH REINFORCEMENT LEARNING

This chapter is concerned with the following question: given that making assumptions about how users behave on a recommendation platform restricts the flexibility of the agent and that choosing the right assumptions is very tricky as shown in the previous two chapters, can we train an assumption-free reinforcement learning agent that works for recommending slates (i.e., lists) of items?

We propose a generic approach that involves pre-training a generative model on logged data from a previous version of the recommender system in order to create a suitable action-space for training RL agents. Rather than claiming state-of-the-art performance thanks to this method, which would require a lot more evidence in diverse scenarios than given in this chapter, I wish to showcase how end-to-end approaches can match and sometimes surpass approaches that incorporate strong inductive biases from decades of research in information retrieval, therefore making them strong contenders for future improvements in RL-based recommender systems.

This chapter is based on the following publication: **Romain Deffayet**, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke. 2023. Generative Slate Recommendation with Reinforcement Learning. In *WSDM'23: the 16th ACM International Conference on Web Search and Data Mining*.

6.1 INTRODUCTION

Ubiquitous in online services, recommender systems (RSs) play a key role in personalization by catering to users' identified tastes. Ideally, they also diversify their offerings and help users discover new interests (Jannach et al., 2021). In the latter case, RSs take on an active role, which means that recommendations influence future user behavior, and therefore their effects on users must be explicitly controlled. Such effects can be detrimental: users may get bored if too many similar recommendations are made, and it has been well-documented that users can end up in so-called filter bubbles or echo chambers (Pariser, 2011; Bakshy et al., 2015; Flaxman et al., 2016). From the perspective of the online platform or the content provider, user boredom leads to poor retention and conversion rates (Hohnhold et al., 2015), while filter bubbles raise fairness and ethical issues for which providers can be held accountable (Masrour et al., 2020). Conversely, RSs can also positively impact users, for example, when users get interested in new, unexpected topics or when the RS offers a fair representation of available options (Anderson et al., 2020). It is natural, therefore, to balance exploitation (i.e., sticking to the known interests of the user) and exploration (i.e., further probing the user's interests) so as to avoid always recommending similar items, and encourage recommendations that boost future engagement. The reinforcement learning (RL) literature has proposed models and algorithms that aim to optimize long-term metrics by acknowledging the causal effect of recommendations on users (Chen et al., 2019b; Zou et al., 2019).

In this work we consider the common scenario of slate recommendation (Sunehag et al., 2015; Ie et al., 2019b; Chen et al., 2019b), which comes with specific challenges. At each interaction turn, a slate recommender system recommends a list of items from the collection, and the user interacts with zero, one or several of those items. As a consequence, users may not examine all the recommended items, which leads to biases in the observed interactions along with a complex interplay between items in the same slate (McInerney et al., 2020). More importantly, the size of the action space, i.e., the number of possible slates, prohibits the use of off-the-shelf RL approaches (Dulac-Arnold et al., 2015). Indeed, as slate recommendation is a combinatorial problem, the evaluation of all actions by the RL agent through trial and error is simply intractable: even with as few as $1,000$ items in the collection, the number of possible slates

of size 10 is approximately 9.6×10^{29}. We propose to tackle this problem in the context of a practical scenario, **(S)**, which fits the second-stage ranking phase (Dang et al., 2013) of many content recommendation platforms:

(**S**) *The collection contains around a thousand items, and at each turn of interaction the proposed model must select and rank 10 items to be presented to the user.*

All our tractability and feasibility statements in this chapter must therefore be understood through the lens of this scenario **(S)**.

To reduce the prohibitively large size of the combinatorial action space, previous studies have proposed to decompose slates in a tractable manner (Sunehag et al., 2015; Ie et al., 2019b; Chen et al., 2019b) – but at the cost of restrictive assumptions, e.g., concerning mutual independence of items in the slate, knowledge of the user click model, availability of high-quality item embeddings, or that at most one item per slate is clicked.

In contrast, we propose to first learn a continuous, low-dimensional latent representation of actions (i.e., slates), and then let the agent take actions within this latent space during its training phase. In practice, we obtain the latent representations by introducing a *generative modeling of slates* (GeMS) based on a variational auto-encoder (VAE) pre-trained on a dataset of observed slates and clicks, collected from a previous version of the recommender system. Such a dataset is usually available in industrial recommendation settings. Therefore, we do not rely on restrictive assumptions, and the fact that we represent full slates enables the agent to improve the quality of its recommendations, instead of using individual item representations.

Our contributions can be summarized as follows:

- We propose GeMS, a novel way to represent actions in RL for slate recommendation, by pre-training a VAE on slates and associated clicks. Unlike previous methods, GeMS is free of overly restrictive assumptions and only requires logged interaction data.
- We provide a unified terminology to classify existing slate recommendation approaches based on their underlying assumptions.
- We show on a wide array of simulated environments that previous methods underperform when their underlying assumptions are lifted (i.e., in practical settings), while GeMS allows us to recover highly rewarding policies without restrictive assumptions.

- To support the reproducibility of this work, we publicly release the code for our approach, baselines and simulator.[1]

6.2 RELATED WORK

Long-term user engagement Several studies have documented the misalignment between short-term benefits and long-term user engagement (Anderson et al., 2020; Hohnhold et al., 2015), as well as the tendency of traditional recommender systems to be detrimental to long-term outcomes (Rossi et al., 2021). Such myopic behavior is known to cause boredom and decrease user retention (Anderson et al., 2020), which is prejudicial for both users and content providers. This behavior also raises concerns such as the rich-get-richer issue (Chen et al., 2019b) and feeding close-mindedness (Rossi et al., 2021). Some previous studies tried to counter this effect by explicitly maximizing diversity (Waller and Anderson, 2019) or by finding metrics correlated with long-term outcomes (Chandar et al., 2022; Athey et al., 2019). In contrast, in our work we directly optimize long-term metrics by using reinforcement learning algorithms (Chen et al., 2019b; Zou et al., 2019; Hansen et al., 2021).

Reinforcement learning for slate recommendation The problem of slate recommendation with reinforcement learning (RL) has been tackled in several previous studies, although the settings in which solutions were tested vary and are sometimes not applicable to our scenario **(S)**. Chen et al. (2019b) and Bai et al. (2019) assume a simple user click model and independence of items within a slate in order to reduce the problem to choosing individual items, which they solve with the REINFORCE algorithm on a SoftMax policy. Ie et al. (2019b) assume knowledge of the user's click model and item relevance, which allows them to perform combinatorial optimization for the computation of Q-values. Sunehag et al. (2015) take a continuous action in the product space of item embeddings, i.e., one embedding per slot in the slate, and pre-select nearest-neighbor items for full-slate Q-function evaluation. Chen et al. (2019c) use properties of the optimal Q-function to propose an elegant decomposition of it and generate optimal slates autoregressively. We detail the assumptions made by each of these approaches in Section 6.4, but we had to discard (Chen

1 https://github.com/naver/gems.

et al., 2019c) due to its prohibitively heavy computation: it requires a number of neural network forward passes proportional to the slate size times the number of items in the collection (i.e., 10,000 passes in scenario **(S)**), for each training or inference step.

Our proposed approach differs from previous work because we do not manually decompose the slates using tractable heuristics based on restrictive assumptions, but instead approximate the slate generation process with a deep generative model. Our proposed framework only has a single requirement, viz. the availability of logged data with slates and associated clicks, as we will detail in Section 6.4. The latter assumption is by no means restrictive as such logged data is readily available in common industrial recommendation settings.

Latent action representations While learning a latent representation of states is very common in the RL literature (Stooke et al., 2021; Ha and Schmidhuber, 2018), few studies have tackled the problem of latent action representation. Chandak et al. (2019) train an action generation function in a supervised manner, by learning to predict the action taken from a pair of successive states. This is not directly applicable in our case, because the true user state is not observable and successive observations are simply clicks that appear to be too weak of a signal to infer the slates leading to these clicks. Botteghi et al. (2021) learn a state-action world model and jointly train latent state and action representations in a model-based fashion.

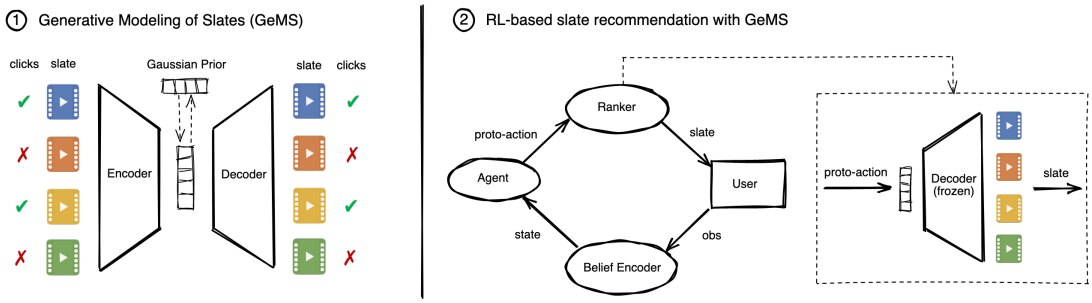


Figure 6.1: Our proposed framework for slate recommendation with reinforcement learning. We first pretrain our GeMS model on previously collected logged data composed of slates and associated clicks (left), then we use the frozen decoder of GeMS to decode the RL agent's low-dimensional proto-action vector into a slate (right).

Learning a world model in our setting essentially amounts to the latent modeling of slates and clicks (similar to our approach), while also conditioning on

an internal hidden state.[2] The work by Zhou et al. (2020) is perhaps the closest work to ours, as it uses a variational auto-encoder (VAE) to embed actions into a controllable latent space before training an RL agent. However, it does not consider slates but only simple, atomic actions. In contrast, Jiang et al. (2019) and Liu et al. (2021) train VAEs to represent slates and their associated clicks, but they do not investigate training an RL agent from the learned latent representation.

To the best of our knowledge, we are the first to learn a latent representation of slates for RL-based recommendation.

6.3 METHOD

6.3.1 Notations and problem definition

We consider a slate recommendation scenario in which a user interacts with a recommender system (RS) throughout an episode of T turns. At every turn $t \in \{1, \dots, T\}$, the system recommends a slate $a_t = (i_t^1, \dots, i_t^k)$ where $(i_t^j)_{1 \leqslant j \leqslant k}$ are items from the collection $\mathcal{I}$ and k is the size of the slate set by the RS designer. The user can click on zero, one or several items in the slate and the resulting click vector $c_t = (c_t^1, \dots, c_t^k), c_t^j \in \{0, 1\}$ is returned to the RS.

The problem of maximizing the cumulative number of clicks over an episode can be modeled as a partially observable Markov decision process (POMDP) $\mathcal{M}^P = (\mathcal{S}, \mathcal{O}, \mathcal{A}, R, T, \Omega)$ defined by:

- A set of states $\mathcal{S}$, which represent the unobservable state of the user's mind;
- A set of observations $\mathcal{O}$ accessible to the system. Here, observations are clicks from the previous interaction ($o_t = c_{t-1}$) and therefore lie in the space of binary vectors of size k: $\mathcal{O} = \{0, 1\}^k$;
- A set of actions $\mathcal{A}$, which is the set of all possible slates composed of items from the collection, i.e., $|\mathcal{A}| = \frac{|\mathcal{I}|!}{(|\mathcal{I}|-k)!}$;
- A reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which we set to $R(s_t, a_t) = r_t = \sum_{j=1}^{k} c_t^j$ in order to reflect our long-term objective of maximizing the cumulative number of clicks; and

2 We tried a similar method in pilot experiments, but the additional conditioning only deteriorated the results, so we only present the condition-free method in this chapter.

- A set of unknown transition and observation probabilities, respectively $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and $\Omega : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$, as well as a distribution over initial states $S^1 : \mathcal{S} \to [0, 1]$.

Due to the unobserved nature of the true user state in the POMDP, it is common to train agents by relying on a proxy of the state inferred from available observations. The function that provides such a proxy is traditionally referred to as the *belief encoder* (Kaelbling et al., 1998). We also define the concepts of a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and trajectory $\tau = (o_t, a_t, r_t)_{1 \leqslant t \leqslant T}$. In the remainder, we write $\tau \sim \pi$ to signify that we obtain a trajectory by first sampling an initial state s_1 from S^1 and then recursively sampling actions $T - 1$ times from the policy π. The goal can now be formulated as finding an optimal policy, i.e., a policy maximizing the *expected return* $\pi^* \in \arg\max_\pi \mathbb{E}_{\tau \sim \pi} [\mathcal{R}(\tau)]$ with $\mathcal{R}(\tau) = \sum_{t=1}^{T} r_t$. Finally, given a state s and action a, we define the Q-function $Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi, s_1 = s, a_1 = a} [\mathcal{R}(\tau)]$ and V-function $V^\pi(s) = \mathbb{E}_{a \sim \pi(s)} [Q^\pi(s, a)]$.

6.3.2 Overview of the framework

In our proposed framework, the interactions with the environment, i.e., the user, can be described by the following repeated steps:

1. The *belief encoder* summarizes the history of interactions with the user into a state vector;

2. The *agent* selects a proto-action based on this state; and

3. The *ranker* (here resulting from a VAE model) decodes this proto-action into a slate that is served to the user.

In the remainder of this section, we first detail our proposed *generative modeling of slates* (GeMS). GeMS is a deep generative model that learns a low-dimensional latent space for slates and associated clicks – thus constituting a convenient proto-action space for the RL agent and allowing for tractable RL without resorting to restrictive assumptions as in prior work (Chen et al., 2019b; Bai et al., 2019; Ie et al., 2019b; Sunehag et al., 2015). Then we describe how GeMS is integrated as a ranker in our RL framework and we briefly discuss the remaining RL components. This two-step process is depicted in Figure 6.1.

6.3.3 Generative Modeling of Slates (GeMS)

In order to instantiate our GeMS model, we propose to train a variational auto-encoder (VAE) on a precollected dataset $\mathcal{D}$ of logged interactions, as illustrated in Figure 6.1 (left). A VAE aims to learn a joint distribution over data samples (i.e., slates and clicks denoted as a and c, respectively) and latent encodings (i.e., proto-actions denoted as z) (Kingma and Welling, 2014). To do so, a parameterized distribution $p_\theta(a, c, z)$ is trained to maximize the marginal likelihood of the data $p_\theta(a, c) = \int_z p_\theta(a, c, z)dz$. In practice, due to the intractability of this integral, a parameterized distribution $q_\phi(z|a, c)$ is introduced as a variational approximation of the true posterior $p_\theta(z|a, c)$ and the VAE is trained by maximizing the evidence lower bound (ELBO):

$$\mathcal{L}^{\text{ELBO}}_{\theta,\phi} = \mathbb{E}_{a,c\sim\mathcal{D}}\Big[\mathbb{E}_{z\sim q_\phi(\cdot|a,c)}[\log p_\theta(a, c|z)] - \text{KL}\left[q_\phi(z|a, c)\|p(z)\right]\Big],$$

where $p(z)$ is the prior distribution over the latent space, KL is the Kullback-Leibler divergence (Kullback and Leibler, 1951), and z is a sample from a Gaussian distribution obtained using the reparameterization trick (Kingma and Welling, 2014). The distributions $q_\phi(z|a, c)$ and $p_\theta(a, c|z)$ are usually referred to as the encoder and the decoder, respectively.

The downstream performance of the RL agent we wish to ultimately learn clearly depends on the upstream ability of the VAE to properly reconstruct slates. However, as Liu et al. (2021) observe, an accurate reconstruction of slates may limit the agent's capacity to satisfy the user's interests. Indeed, finding high-performance continuous control policies requires smoothness and structure in the latent space, which may be lacking if too much emphasis is given to the reconstruction objective in comparison to the prior matching objective enforced by the KL-divergence. Therefore, it is necessary to balance reconstruction and controllability, which is done by introducing an hyperparameter β as weight for the KL term in Eq. 6.1. Moreover, in order to promote additional structure in the latent space, we add a click reconstruction term in the loss: slates with similar short-term outcomes (i.e., clicks) are grouped together during pre-training. Yet, we may want to avoid biasing the learned representations towards click reconstruction too much, as it may come at the cost of quality of the slate reconstruction. Therefore, we introduce a hyperparameter λ to adjust this second trade-off. We show the empirical impact of β and λ in Section 6.6.3.

In our implementation, the prior $p(z)$ is set as a standard Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The encoder $q_\phi(z|a, c)$ is a Gaussian distribution with diagonal

covariance $\mathcal{N}(\mu_\phi(a,c), \text{diag}(\sigma^2_\phi(a,c)))$, parameterized by a multi-layer perceptron (MLP). This MLP inputs the concatenation of learnable item embeddings and associated clicks over the whole slate, and outputs $(\mu_\phi(a,c), \log \sigma_\phi(a,c))$. For the decoder $p_\theta(a,c|z)$, another MLP takes as input the latent sample z, and outputs the concatenation of reconstructed embeddings $\mathbf{e}^j_\theta(z)$ and click probabilities $p^{j,c}_\theta(c_j|z)$ for each slot j in the slate. We then derive logits for the item probabilities $p^{j,a}_\theta(a_j|z)$ by taking the dot-product of the reconstructed embedding $\mathbf{e}^j_\theta(z)$ with the embeddings of all items in the collection. For collection items, we use the current version of embeddings learned within the encoder, but we prevent the gradient from back-propagating to them using the stop-gradient operator to avoid potential degenerate solutions.

In summary, the VAE is pre-trained by maximizing the ELBO on the task of reconstructing slates and corresponding clicks, i.e., by minimizing $\mathcal{L}^{\text{GeMS}}_{\theta,\phi} = \mathbb{E}_{a,c\sim\mathcal{D}}[\mathcal{L}^{\text{GeMS}}_{\theta,\phi}(a,c)]$ with:

$$\begin{aligned}\mathcal{L}^{\text{GeMS}}_{\theta,\phi}(a,c) = &\overbrace{\sum_{j=1}^{k} \log p^{j,a}_\theta(a_j|z_\phi(a,c))}^{\text{slate reconstruction}} + \\ &\overbrace{\lambda \sum_{j=1}^{k} \log p^{j,c}_\theta(c_j|z_\phi(a,c))}^{\text{click reconstruction}} + \\ &\overbrace{\beta \sum_{i=1}^{d} \left(\sigma^2_{\phi,i} + \mu^2_{\phi,i} - \log \sigma_{\phi,i} - 1\right)}^{\text{KL-divergence}} .\end{aligned} \tag{6.1}$$

where $z_\phi(a,c) = \mu_\phi(a,c) + \text{diag}(\sigma^2_\phi(a,c)) \cdot \epsilon$, for $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Here, d is the dimension of the latent space, and β and λ are hyperparameters controlling the respective weight of the KL term and the click reconstruction term. Note that the KL term takes this simple form due to the Gaussian assumption on $q_\phi(z|a,c)$ and the $\mathcal{N}(\mathbf{0}, \mathbf{I})$ prior.

6.3.4 RL agent and belief encoder

After the pre-training step described in Section 6.3.3, the parameters of GeMS are frozen and we use its decoder as the ranker in our RL framework. The RL agent can then be trained to maximize the discounted return by taking proto-actions within the VAE's latent space. To generate a slate $(i^1, \ldots, i^k)$ from the

agent's proto-action z, we take for each slot $j \in \{1, \dots, k\}$ the most likely item according to the decoder: $i^j = \arg\max_{i \in \mathcal{I}} p_\phi^{j,a}(i|z)$.

Since our focus within the RL framework is on the choice of the ranker, we adopt a standard implementation of the belief encoder and the agent: the former is modeled by a gated recurrent unit (GRU) (Cho et al., 2014) taking as input the concatenation of item embeddings and respective clicks from each slate, and the latter is a soft actor-critic (SAC) (Haarnoja et al., 2018) algorithm. We chose SAC because it is a well-established RL algorithm, known for its strong performance and data-efficiency in continuous control. Additionally, SAC adds an entropy term incentivizing exploration which we have noticed during our experiments to be important to attain high performance in highly stochastic recommendation environments.

6.4 BASELINES AND THEIR ASSUMPTIONS

We evaluate our proposed method against four main baselines derived from prior work. In this section, we describe these baselines as well the assumptions on user behavior that they formulate in order to make the combinatorial problem of slate recommendation tractable. By doing so, we are able to compare the assumptions made by these baselines and highlight the generality of our method in Table 6.1. Note that we only report from previous studies the mechanism used for slate generation, which is the topic of this study, and ignore other design choices.

SoftMax In (Chen et al., 2019b; Bai et al., 2019), the authors reduce the combinatorial problem of slate optimization to the simpler problem of item optimization: the policy network output is a softmax layer over all items in the collection, and items are sampled with replacement to form slates.
Doing so requires the mild assumption that *the Q-value of the slate can be linearly decomposed into item-specific Q-values* (**DQ**). But more importantly, it also requires two strong assumptions, namely *users can click on at most one item per slate* (**1CL**) and *the returns of items in the same slate are mutually independent* (**MI**). Together, these assumptions are restrictive, because their conjunction means that the click probability of an item in the slate does not depend on the item itself. Indeed, having dependent click probabilities (to enforce the single click)

Table 6.1: Comparison of assumptions made by prior work. Our method only requires access to logged interaction data.

	1CL	**DQ**	**MI**	**CM**	**SP**	**EIB**	**LD**
SoftMax (Chen et al., 2019b; Bai et al., 2019)	✓	✓	✓	✗	✗	✗	✗
SlateQ (Ie et al., 2019b)	✓	✓	✗	✓	✗	✗	✗
WkNN (Sunehag et al., 2015)	✗	✓	✗	✗	✓	✓	✓
TopK	✗	✗	✗	✗	✓	✗	✓
GeMS (Ours)	✗	✗	✗	✗	✗	✗	✓

and independent items in the slate is compatible only if click probabilities do not depend on items.

SlateQ Ie et al. (2019b) propose a model-based approach in which the click behavior of the user is given, and Q-learning (Watkins and Dayan, 1992) is used to plan and approximate users' dynamic preferences. On top of the earlier DQ and 1CL, it requires *access to the true relevance and click model* (**CM**), which is an unfair advantage compared to other methods. For computational efficiency reasons, we adopt the faster variant referred to as QL-TT-TS in the original paper.

TopK Even though, to the best of our knowledge, no work has proposed this approach, we include it in our set of baselines as it is a natural way to deal with slate recommendation. The agent takes continuous actions in the space of item embeddings, and we generate slates by taking the k items from the collection with the closest embeddings to the action, according to a similarity metric (the dot-product in practice). This method therefore assumes the *availability of logged data of past interactions* (**LD**), in order to pre-train item embeddings. In our experiments, we evaluate two variants of this baseline: *TopK (MF)*, where item embeddings are learned by matrix factorization (Koren et al., 2009), and *TopK (ideal)*, which uses ideal item embeddings, i.e., the embeddings used internally by the simulator (see Section 6.5.1). The latter version clearly has an unfair advantage. Also, because ranking items this way assumes that the most rewarding items should appear on top, it makes the *sequential presentation* (**SP**) assumption from (Sunehag et al., 2015) that *the true click model is top-down and fading*, i.e., if $c(i)$ indicates that item i has been clicked and $l \leqslant k$ is the posi-

tion of i in slate a, then $P(c(i)|s,a) = P(c(i)|s,a_{\leqslant l}) \leqslant P(c(i)|s,\tilde{a}_{\leqslant l-1})$, where $a_{\leqslant l} = (i^1,\dots,i^{l-1},i)$ and $\tilde{a}_{\leqslant l-1} = (i^1,\dots,i^{l-2},i)$.

WkNN In (Sunehag et al., 2015), the authors propose a finer-grained and potentially more capable variant of TopK referred to as *Wolpertinger* (Dulac-Arnold et al., 2015): the agent takes actions in the product-space of item embeddings over slate slots, i.e., continuous actions of dimension $k \times d$, where d is the dimension of item embeddings. Then, for each slot in the slate, p candidate items are selected by Euclidean distance with embeddings of items from the collection, and every candidate item's contribution to the Q-value is evaluated in a greedy fashion. Besides LD and DQ, WkNN requires two strong assumptions to ensure submodularity of the Q-function: sequential presentation SP and *execution is best* (**EIB**), i.e., *recommendations that are risky on the short term are never worth it*. Formally, this translates as: $\mathbb{P}(R(s,\pi_1(s)) = 0) \geqslant \mathbb{P}(R(s,\pi_2(s)) = 0) \Rightarrow V^{\pi_1}(s) \leqslant V^{\pi_2}(s)$ for any policies π_1, π_2. Note that it partly defeats the purpose of long-term optimization.

In Table 6.1, we summarize the assumptions made by each baseline. In comparison to prior work, our proposed framework has a single assumption: the availability of logged data with slates and associated clicks (LD), as Table 6.1 indicates. This assumption is by no means restrictive as such logged data is readily available in common industrial recommendation settings.

On top of these baselines, we also include a **random** policy and a **short-term oracle** as reference points. The short-term oracle has access to the true user and item embeddings, enabling it to select the items with the highest relevance probability in each slate. Therefore, at each turn of interaction, it gives an upper bound on the immediate reward but it is unable to cope with boredom and influence phenomena.

6.5 EXPERIMENTAL SETUP

6.5.1 Simulator

We design a simulator that allows us to observe the effect of lifting the assumptions required by the baselines, and we experiment with several simulator variants to ensure generalizability. We summarize our main design choices

Table 6.2: Average cumulative number of clicks on the test set for our 6 simulated environments. Bold: best method; underlined: 2nd-best method; †: statistically significantly better than all other methods. 95% confidence intervals are given in parentheses. Methods grouped under "Disclosed env." have access to privileged information about the environment and can therefore not be fairly compared with "Undisclosed env." methods.

		Focused item embeddings			Diffuse item embeddings		
	Method	**TopDown**	**Mixed**	**DivPen**	**TopDown**	**Mixed**	**DivPen**
Disclosed env.	Short-term oracle	107.7	101.6	85.4	96.7	94.6	78.8
	SAC+TopK (ideal)	429.0 (±5.9)	384.1 (±13.5)	386.3 (±15.5)	373.9 (±25.0)	371.9 (±36.4)	341.3 (±55.3)
	SlateQ	206.5 (±4.1)	202.7 (±3.4)	119.0 (±3.9)	209.5 (±5.4)	192.7 (±5.1)	117.8 (±5.8)
Undisclosed env.	Random	33.8 (±0.2)	33.9 (±0.2)	33.6 (±0.2)	33.3 (±0.2)	33.2 (±0.2)	32.9 (±0.2)
	REINFORCE+SoftMax	248.1 (±19.3)	<u>233.5</u> (±18.5)	<u>249.1</u> (±11.6)	249.5 (±15.3)	<u>214.7</u> (±25.0)	213.8 (±27.1)
	SAC+WkNN	98.5 (±8.9)	97.7 (±10.8)	95.5 (±9.9)	107.2 (±8.9)	89.8 (±7.4)	92.5 (±5.0)
	SAC+TopK (MF)	<u>254.4</u> (±17.1)	232.7 (±19.4)	242.2 (±15.4)	<u>249.7</u> (±10.3)	184.1 (±1.3)	<u>231.4</u> (±13.3)
	SAC+GeMS (Ours)	**305.3**† (±**21.9**)	**242.6** (±**21.5**)	**254.1** (±**27.7**)	**300.0**† (±**42.8**)	**260.6**† (±**27.2**)	**249.6** (±37.6)

below and refer the reader to our code available online[3] for a more detailed description.

Item and user embeddings Following scenario **(S)**, our simulator includes 1,000 items. We consider a cold-start situation where users are generated on-the-fly for each new trajectory. Items and users are randomly assigned embeddings of size 20, corresponding to ten 2-dimensional topics: $\mathbf{e} = (\mathbf{e}^1, \dots, \mathbf{e}^{10})$. Each 2-dimensional vector $\mathbf{e}^t$ is meant to capture the existence of subtopics within topic t. The embedding of a user or item x is generated using the following process: (i) sample topic propensities $w_x^t \sim \mathcal{U}(0,1)$ and normalize such that $\sum_t w_x^t = 1$; (ii) sample topic-specific components $\mathbf{ffl}_x^t \sim \mathcal{N}(\mathbf{0}, 0.4 \cdot \mathbf{I}_2)$ and rescale as $\mathbf{e}_x^t = w_x^t \cdot \min(|\mathbf{ffl}_x^t|, 1))$; and (iii) normalize the embedding $\mathbf{e}_x = (\mathbf{e}_x^1, \dots, \mathbf{e}_x^{10})$ such that $\|\mathbf{e}_x\| = 1$. Each item is associated to a main topic, defined as $t(i) = \arg\max_{1 \leqslant t \leqslant 10} \|\mathbf{e}_i^t\|$.

To accomodate different types of content and platforms, we derive two variants of item embeddings in the simulator: one with embeddings obtained as described above, and one with embeddings for which we square and re-normalize each component. In Section 6.6, we highlight this difference in peakedness by referring to the former as *diffuse embeddings* and the latter as *focused embeddings*.

Relevance computation The relevance probability of item i for user u is a monotonically increasing function of the dot-product between their respective embeddings: $\text{rel}(i,u) = \sigma({\mathbf{e}_i}^T \mathbf{e}_u)$, where σ is a sigmoid function.

Boredom and influence effects User embeddings can be affected by two mechanisms: *boredom* and *influence*. Each item i clicked by user u influences the user embedding in the next interaction turn as: $\mathbf{e}_u \leftarrow \omega \mathbf{e}_u + (1-\omega)\mathbf{e}_i$, where we set $\omega = 0.9$ in practice. Additionally, if in the last 10 items clicked by user u five have the same main topic t^b, then u gets bored with this topic, meaning we put $\mathbf{e}_u^{t^b} = \mathbf{0}$ for 5 turns. These mechanisms have been defined to penalize myopic behavior and encourage long-term strategies.

Click model Users click on recommended items according to a position-based model, i.e., the click probability is the product of item-specific attractiveness and rank-specific examination probabilities: $\mathbb{P}(c|i,r) = A_i \times E_r$. Specifically, we define for an item located at rank r: $E_r = \nu\varepsilon^r + (1-\nu)\varepsilon^{k+1-r}$ with $\varepsilon = 0.85$.

3 https://naver/github/gems

It is a mixture of the terms ϵ^r and ϵ^{k+1-r}, which capture the top-down and bottom-up browsing behaviors, respectively. We use two variants of this click model in our experiments: *TopDown* with $\nu = 1.0$ and *Mixed* with $\nu = 0.5$. The attractiveness of an item is set to its relevance in TopDown and Mixed. In addition, we consider a third variant *DivPen* which also penalizes slates that lack diversity: A_i is down-weighted by a factor of 3 if more than 4 items from the slate have the same main topic (as in Mixed, we also set $\nu = 0.5$ for DivPen).

In summary, our experiments are performed on 6 simulator variants defined by the choice of item embedding peakedness (*diffuse item embeddings* or *focused item embeddings*) and the choice of click model (*TopDown*, *Mixed*, or *DivPen*).

6.5.2 Implementation and evaluation details

Our implementation aims to be as standard as possible, considering the literature on RL, in order to ensure reproducibility. All baselines are paired with SAC (Haarnoja et al., 2018), except SlateQ which is based on Q-Learning (Watkins and Dayan, 1992), and SoftMax, which we pair with REINFORCE (Sutton and Barto, 2018b) because it requires a discrete action space and a discretized variant of SAC led to lower performance in our experiments. We implement all agents using two-layer neural networks as function approximators, and use target networks for Q-functions in Slate-Q and SAC. For hyperparameters common to baselines and our method, we first performed a grid search over likely regions of the space on baselines, and re-used the selected values for our method. For all methods we use the Adam optimizer with learning rates of 0.001 for Q-networks and 0.003 for policy networks when applicable, as well as a discount factor $\gamma = 0.8$ and a Polyak averaging parameter $\tau = 0.002$. For the hyperparameters specific to our method (d, β and λ), we perform a grid search on the TopDown environment with focused item embeddings and select the combination with the highest validation return. This combination is then re-used on all other environments. The searched ranges were defined as $d \in \{16, 32\}$, $\beta \in \{0.1, 0.2, 0.5, 1.0, 2.0\}$ and $\lambda \in \{0.0, 0.2, 0.5, 1.0\}$.

For methods making the (LD) assumption, we generated a dataset of 100K user trajectories (with 100 interactions turns each) from an ϵ-greedy oracle policy with $\epsilon = 0.5$, i.e., each recommended item is selected either uniformly randomly or by an oracle, with equal probabilities. The VAE in GeMS is trained on this dataset for 10 epochs with a batch size of 256 and a learning rate of 0.001.

For approaches requiring pre-trained item embeddings (TopK and WkNN), we learn a simple matrix factorization model on the generated dataset by considering as positive samples the pairs composed of the user in the trajectory and each clicked item in their recommended slates.

In all of our experiments, we compare average cumulative rewards over 10 seeded runs, corresponding to ten initializations of the agent's parameters. In the case of GeMS, the seed also controls the initialization of the VAE model during pre-training. We train agents for 100K steps. Each step corresponds to a user trajectory, composed of 100 interaction turns (i.e., 100 slates successively presented to the user) for a unique user. Every $1,000$ training steps, we also evaluate the agents on 200 validation user trajectories. Finally, the agents are tested by selecting the checkpoint with the highest validation return and applying it on 500 test user trajectories. Confidence intervals use Student's t-distribution, and statistical tests are Welch's t-test. Both are based on a 95% confidence level.

6.6 RESULTS

In our experiments, we investigate the following research questions:

(**RQ1**) How does our slate recommendation framework based on GeMS compare to previous methods when the underlying assumptions of the latter are lifted?

(**RQ2**) Does the proposed GeMS framework effectively balance immediate and future rewards to avoid boredom?

(**RQ3**) How do the balancing hyperparameters β and λ in GeMS impact the downstream RL performance?

6.6.1 Comparison of our method against baselines (RQ1)

In this section, we compare the performance of our method and baselines on a wide array of simulated environments, corresponding to the six environments described in Section 6.5.1.

Overview of the results Table 6.2 shows the average test return (i.e., cumulated reward or cumulated number of clicks) after training on 100K user trajectories. We group methods into two categories: *Disclosed env.*, i.e., methods leveraging hidden environment information, and *Undisclosed env.*, i.e., methods that consider the environment as a black-box and are therefore practically applicable. A first observation we can draw, regardless of the specific environment used, is that the short-term oracle is easily beaten by most approaches. Indeed, the simulator penalizes short-sighted recommendations that lead to boredom: in these environments, *diversity is required to reach higher returns.* We can also observe the superiority of SAC+TopK (Ideal). This is not surprising, as this method benefits from an unfair advantage – access to true item embeddings – but it suggests that practically applicable methods could be augmented with domain knowledge to improve their performance. However, despite having access to privileged information, SlateQ's performance is subpar, especially in DivPen environments. Its lower performance might be explained by its approximate optimization strategy and restrictive single-click assumption.

Overall comparison of methods *The proposed SAC+GeMS compares favorably to baselines across the range of environments we simulate.* Out of the 6 tested environments, SAC+GeMS obtained the best average results on all of them, among which 3 show a statistically significant improvement over all other methods. SAC+WkNN performs very poorly: we hypothesize that the approach suffers from the curse of dimensionality due to the larger action space (200 dimensions in our experiments) and the assumption made by the approach that candidate items need to be close to target item embeddings according to the Euclidean distance. SAC+TopK (MF) is more competitive, but the large difference with SAC+TopK (ideal) suggests that TopK is very sensitive to the quality of item embeddings. Despite its very restrictive assumptions and lack of theoretical guarantees in our setup, REINFORCE+SoftMax was a very competitive baseline overall. However, while its best checkpoint had high return, its training was unstable and failed to converge in our experiments, which suggests it may be unreliable.

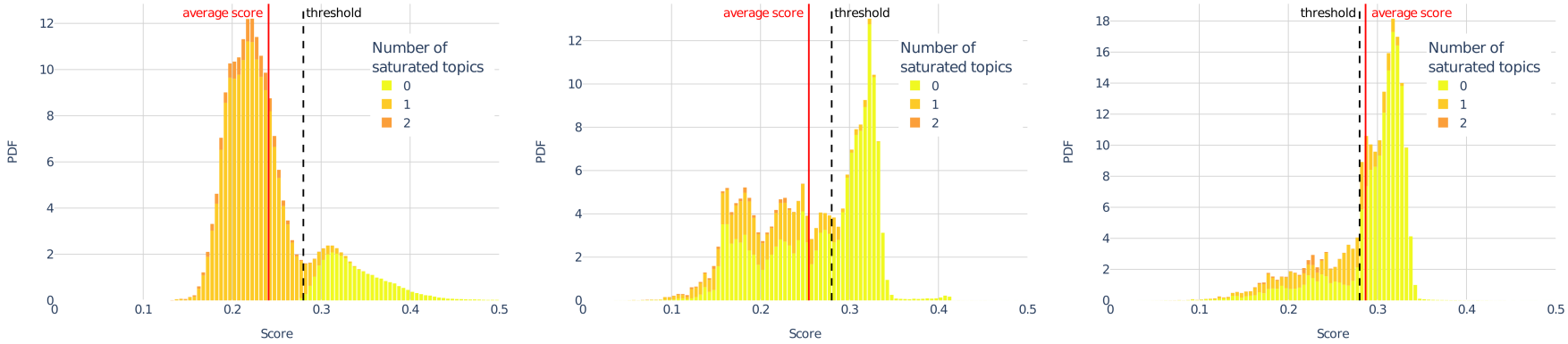
Comparisons across environments The TopDown environment is the easiest for most methods, regardless of the type of item embeddings. This is not surprising as all methods besides Random either assume a top-down click model, sample items in a top-down fashion or rely on data from a top-down

logging policy. However, it is worth noting that other factors can dominate the performance, such as sub-optimality of item embeddings for SAC+TopK (MF). Conversely, DivPen was harder for most methods, because it requires a strong additional constraint to obtain high returns: intra-slate diversity must be high. SAC+GeMS was also affected by these dynamics, but remained able to beat other methods by generating diverse slates. Finally, the use of diffused item embeddings does not appear to cause lower returns for GeMS, compared with focused ones, but is associated with larger confidence intervals for SAC+GeMS: indeed, pivot items spanning multiple topics are more likely to be attractive, at the expense of more fine-grained strategies, making the training process uncertain.

6.6.2 GeMS overcomes boredom to improve its return (RQ2)

In Section 6.1 we highlighted that long-term optimization with RL can penalize myopic behavior such as recommending only highly relevant but similar items, which may lead to boredom. In this section, we verify that SAC+GeMS is able to adapt its slate selection to cope with boredom. We recall that in our simulated environments (detailed in Section 6.5.1), users get bored of a particular topic whenever 5 of their latest 10 clicks were on items from that topic. When a topic is saturated, its corresponding dimensions in the user embedding are set to $\mathbf{0}$, which has the effect of diminishing the attractiveness of future items presented to the user. It is therefore necessary to avoid boredom in order to reach higher returns, even if it comes at the cost of lower immediate rewards.

We compare three approaches on the TopDown environment with focused item embeddings: (i) the short-term oracle (STO) always maximizing the immediate reward, (ii) SAC+GeMS with $\gamma = 0.8$ (i.e., our proposed method) where γ is the discount factor of the RL algorithm, and (iii) SAC+GeMS with $\gamma = 0$ which does not explicitly include future rewards in its policy gradient. In this environment, SAC+GeMS$^{\gamma=0.8}$ achieves an average test return of 305.3, while SAC+GeMS$^{\gamma=0}$ reaches 194.3, and STO only obtains 107.7. These results suggest that long-term optimization is indeed required to reach higher returns. It may seem surprising that SAC+GeMS$^{\gamma=0}$ gets better returns than STO, but its training objective incentivizes *average* immediate rewards, which implicitly encourages it to avoid low future rewards. However, adopting an explicit mech-



(a) Short-term oracle.

(b) SAC+GeMS with $\gamma = 0$.

(c) SAC+GeMS with $\gamma = 0.8$.

Figure 6.2: Distribution of the relevance scores of items recommended by (a) a short-term oracle, (b) SAC+GeMS with $\gamma = 0$ and (c) SAC+GeMS with $\gamma = 0.8$. Boredom penalizes item scores and is visualized by orange areas. The myopic approaches (left, center) lead to more boredom than the long-term approach (right), and therefore to lower average item scores (solid red lines).

anism to account for its causal effect on the user (i.e., setting $\gamma = 0.8$) allows SAC+GeMS to improve its decision-making.

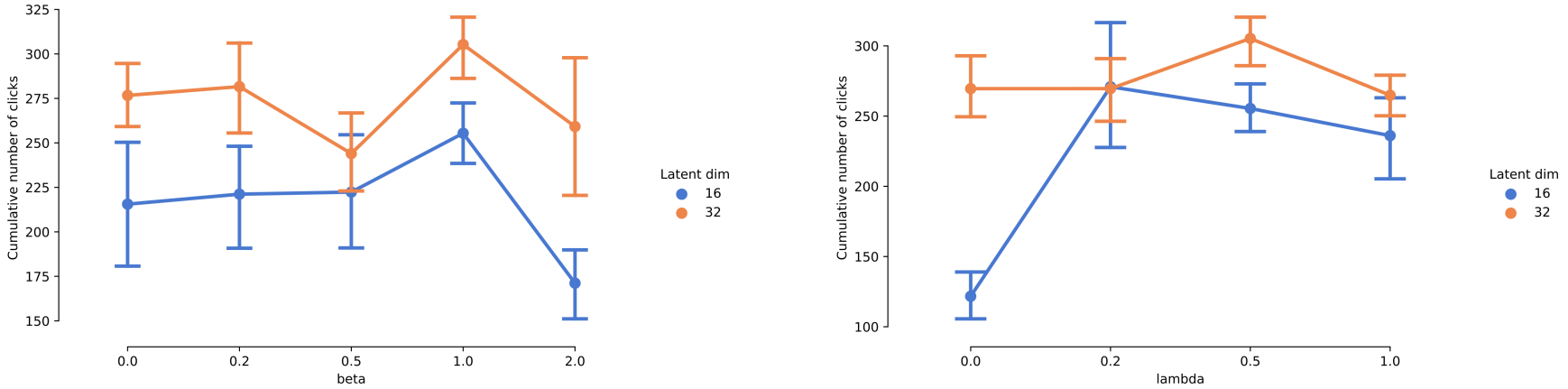
In Figure 6.2, we plot the distribution of item scores (i.e., the dot-product between internal user and item embeddings as defined in Section 6.5.1) for the items recommended in slates by each of the three methods, with the same seed for all three plots. The dashed vertical line shows the score threshold of 0.28 needed to reach a relevance probability of 0.5. Therefore, items on the left of this line have a lower click probability while items on the right have a higher click probability. The color indicates how many topics were saturated when the agent recommended that particular item whose score is plotted: one can see that when the user is bored of at least one topic, items become less attractive as scores are reduced.

When no topic is saturated (i.e., yellow distribution), STO recommends items with excellent scores (above the threshold and up to 0.45): as a consequence, STO gets high immediate rewards. However, by doing so it incurs a lot of boredom (large orange areas). Overall, it leads to lower expected scores (solid red line) and therefore fewer clicks. Conversely, SAC+GeMS$^{\gamma=0.8}$ sacrifices some immediate reward (yellow distribution shifted to the left) but causes very little boredom (small orange area). Overall, *by trading off relevance and diversity, SAC+GeMS$^{\gamma=0.8}$ yields good immediate rewards while limiting boredom.* It therefore gets higher average scores. SAC+GeMS$^{\gamma=0}$ exhibits an intermediate behavior due to its limited capabilities: it recommends items of varying relevance, yet leads to substantial boredom (larger orange area than for $\gamma = 0.8$).

6.6.3 Balancing hyperparameters β and λ (RQ3)

In Section 6.3.3, we suggested that the choice of β and λ leads to trade-offs that may impact the downstream performance of SAC+GeMS. As a reminder, β adjusts the importance of accurate reconstruction versus smoothness and structure in the latent space (i.e., controllability), while λ weights the click reconstruction with respect to the slate reconstruction. Next, we verify our intuition on the importance of these trade-offs by reporting (in Figure 6.3) the best validation return obtained for different values of said hyperparameters, on the TopDown environment with focused item embeddings.

Figure 6.3a suggests that, indeed, there exists a "sweet spot" in the selection of β. It confirms the intuition described in Section 6.3.3 and the observation of



(a) Impact of β for $\lambda = 0.5$.

(b) Impact of λ for $\beta = 1.0$.

Figure 6.3: Average cumulative number of clicks on the validation set obtained by SAC+GeMS with its best validation checkpoint, for different values of β and λ (defined in Section 6.3.3). We also display 95% confidence intervals.

Liu et al. (2021): *β must be appropriately balanced in order to ensure high performance on the downstream RL task.* Specifically, we found that choosing $\beta = 1.0$ leads to the highest return overall, regardless of whether a latent dimension of 16 or 32 is used.

The impact on the downstream performance of the trade-off between slate and click reconstruction (Figure 6.3b) is less prominent but can still be observed. It justifies our choice to add the click reconstruction term in the loss (Eq. 6.1), even though clicks output by GeMS' decoder are not used during RL training. This also confirms the importance of introducing and adjusting the hyperparameter λ: *modeling clicks jointly with slates improves the final performance of SAC+GeMS, but properly weighting the click reconstruction objective with respect to the slate reconstruction objective is necessary.*

6.7 CONCLUSION

We have presented GeMS, a slate representation learning method based on variational auto-encoders for slate recommendation with reinforcement learning. This method has the notable advantage of being flexible, allowing full-slate modeling and lightweight assumptions, in contrast with existing approaches.

Findings and broader impact Our experiments across a wide array of environments demonstrate that GeMS compares favorably against existing slate representation methods in practical settings. Moreover, our empirical analysis highlights that it effectively balances immediate and future rewards, and that the trade-offs imposed by β and λ significantly impact the RL downstream performance, indicating that properly balancing these hyperparameters is critical. Our work suggests that generative models are a promising direction for representing rich actions such as slates.

Limitations Our simulated experiments demonstrate the effectiveness of GeMS for representing slates in an RL framework. However, it is well-known that online training of RL agents is too expensive and risky, and that in practice agents must be trained offline, i.e., directly from logged data (Chen et al., 2019b). We did not address here the specific challenges of offline RL, as we wished to isolate the contribution of the slate representation to downstream performance.

Future work In future work, we will investigate how generative models can be leveraged in the offline setting, in different scenarios, or with even richer actions. We also plan to look into improvements of the architectures used for structured action representations, for example by using domain knowledge and user models.

6.8 REFLECTIONS ON THE CHAPTER

6.8.1 Research outcomes

In this chapter, we primarily investigated the fourth research question:

Research Question 4. *How to train reinforcement learning agents that recommend slates of items to users effectively and efficiently?*

We have proposed an alternative way of treating the slate recommendation problem, based on a data-driven representation of the structure of the slate-feedback space, rather than hardcoded, strong inductive biases.

With this chapter, we also gained a different perspective on the third research question:

Research Question 3. *When do we need assumptions on user behavior, and how can we test for the validity of these assumptions?*

Indeed, and in contrast to common practices in the literature, it turns out that the size and complexity of the slate space does not necessarily require simplifying assumptions on user feedback, at least not for agent training. This comes in contrast to the offline, item-centric approach of Chapters 4 and 5, where assumptions were required to correctly identify and mitigate presentation biases.

6.8.2 Additional thoughts

GeMS offers the possibility not only to perform a continuous relaxation of the discrete slate selection problem, but also to make that latent space semantically meaningful for the recommendation task. We showed that adding a click prediction objective helps structure the latent space and lets the agent reach

higher performance during RL training. It opens a wider alley of research into the type of representations needed for effective and efficient RL training in recommendation.

It is particularly interesting for offline reinforcement learning, where we have to learn from a fixed dataset. Then, agents can potentially improve on the data-generating policy thanks to two mechanisms: stitching parts of trajectories and generalizing across states and actions (Levine et al., 2020). Adding side objectives to GeMS can allow generalisation to users and slates that are distinct, but semantically close, in the sense that they yield similar outcomes (clicks, boredom, ...). However, offline RL training of an agent based on GeMS would come with additional challenges: the support of the training must be correctly estimated in order to identify out-of-distribution user-slate pairs. Lee et al. (2022b) proposed an action representation similar to GeMS, where the distribution of the latent proto-action is estimated thanks to a normalizing flow, which guarantees that only in-distribution actions are being selected by the agent.

7

DISTRIBUTIONAL REINFORCEMENT LEARNING WITH DUAL EXPECTILE-QUANTILE REGRESSION

Recommendation environments, because they imply interactions with humans, are highly stochastic. Such uncertainty can create noise that destabilizes the learning process, and may create trade-offs between risk and reward. Developing better ways of estimating and acting with respect to it is thus certainly a promising research direction for reinforcement learning-based recommendation systems, such as the one presented in the previous chapter.

I found the area of uncertainty-aware reinforcement learning agents to be underexplored, so in this final research chapter, I chose to investigate in a more general and theoretical manner the estimation of uncertainty in dynamic and interactive environments. I chose to focus on distributional RL, as it is often presented as the go-to approach for learning full value distributions and therefore providing uncertainty estimation.

Quantile-based approaches for distribution RL are very popular because of their performance and flexibility, but after initial experiments, we found that their performance is limited by the use of the L_1-based quantile loss function. The usual fix for this bottleneck is to replace the original quantile loss with a hybrid L_1 - L_2 Huber loss that can leverage the efficiency of L_2 based losses. But this comes at the cost of distributional estimation guarantees, and the value

distribution often ends up collapsing to its mean, which defeats the purpose of distributional RL. We wanted to find a way to obtain the best of both worlds, thereby giving rise to the hybrid expectile-quantile approach presented in this chapter.

This chapter is based on the following publication: Sami Jullien, **Romain Deffayet**, Jean-Michel Renders, Paul Groth, and Maarten de Rijke, 2024. Distributional Reinforcement Learning with Dual Expectile-Quantile Regression. *Under review*.

7.1 INTRODUCTION

Distributional reinforcement learning (RL) (Bellemare et al., 2023) aims to maintain an estimate of the full distribution of expected return rather than only the mean. Compared to a mean-based approach, it can be used to better capture the uncertainty in the transition matrix of the environment (Bellemare et al., 2017), as well as the stochasticity of the policy being evaluated, which may enable faster and more stable training by making better use of the data samples (Mavrin et al., 2019).

Non-parametric approximations of the return distribution learned by quantile regression have proven very effective in several domains (Dabney et al., 2018a; Dabney et al., 2018b; Yang et al., 2019), when combined with deep RL agents such as deep Q-networks (DQN) (Mnih et al., 2013) or soft actor-critic (SAC) (Haarnoja et al., 2018). They come with the major advantage of providing guarantees for the convergence of distributional policy estimation (Dabney et al., 2018b), and in certain cases, of convergence to the optimal policy (Rowland et al., 2023), all while requiring few assumptions on the shape of the return distribution and demonstrating strong empirical performance (Dabney et al., 2018a; Yang et al., 2019). However, the best-performing quantile-based agents are often obtained by replacing the original quantile regression loss function, i.e., an asymmetric L_1 loss, by an asymmetric Huber loss, i.e., a hybrid L_1-L_2 loss. By doing so, distributional guarantees vanish, as the proofs proposed in previous work relied on the L_1-based quantile regression (Bellemare et al., 2023; Dabney et al., 2018b). Critically, we show in Section 7.5.2 that the estimated distributions collapse to their mean in practice. In this chapter, *we propose a different approach, based on both quantile and expectile regression,*

that matches the performance of Huber-based agents while preserving distributional estimation guarantees and avoiding distributional collapse in practice.

We are not the first to note that asymmetric L_2 losses, i.e., that regress *expectiles* of the target distribution, tend to yield degenerate estimated distributions when training agents with temporal difference learning. Rowland et al. (2019) note that expectiles of a distribution cannot be interpreted as samples from this distribution, and therefore expectiles other than the mean cannot be directly used to compute the target values in distributional temporal difference learning. Instead, they propose to generate samples from expectiles of the distribution by adding an imputation step, that requires solving a costly root-finding problem. While theoretically justified, we found this approach to be extremely slow in practice, preventing widespread use at scale. In contrast, our dual approach only requires an additional two-layer neural network and computing a quantile loss function on top of the expectile loss function, which adds close to no computational overheads when training Atari agents on modern GPUs.

Our contributions can be summarized as follows:

- We propose a dual expectile-quantile approach to distributional dynamic programming that provably converges to the true value distribution.
- We release implicit expectile-quantile networks (IEQN),[1] a practical implementation of our dual approach based on implicit quantile networks (Dabney et al., 2018a).
- We show both on a toy example and at scale on the Atari-5 benchmark that IEQN (i) avoids distributional collapse, and (ii) matches the performance of the Huber-based IQN-1 approach.

7.2 BACKGROUND

7.2.1 Distributional reinforcement learning

We consider an environment modeled by a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, R, T, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are a state and action space, respectively, $R(s, a)$ denotes the stochastic reward obtained by taking action a in state s, $T(\cdot \mid s, a)$ is the probability distribution over possible next states after taking a in s, and γ

1 Available at `https://anonymous.4open.science/r/ieqn`.

is a discount factor. Furthermore, we write $\pi(\cdot \mid s)$ for a (potentially stochastic) policy selecting the action depending on the current state. We consider the problem of finding a policy maximizing the average discounted return, i.e.,

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right], \tag{7.1}$$

where $a_t \sim \pi(\cdot \mid s_t)$ and $s_{t+1} \sim T(\cdot \mid s_t, a_t)$. We can define the action-value random variable for policy π as $Z^\pi : (s, a) \mapsto \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$, with $s_0 = s, a_0 = a$. We will refer to action-value variables and their estimators as Z-functions in the remainder. Note that the Q-function, as usually defined in RL (Sutton and Barto, 2018a), is given by $Q^\pi(s, a) = \mathbb{E}\left[Z^\pi(s, a)\right]$. In this work, we consider approaches that evaluate policies through distributional dynamic programming, i.e., by repeatedly applying the distributional Bellman operator $\mathcal{T}^\pi$ to a candidate Z-function:

$$\mathcal{T}^\pi Z(s_t, a_t) = R(s_t^\pi, a_t^\pi) + \gamma Z(s_{t+1}^\pi, a_{t+1}^\pi). \tag{7.2}$$

This operator has been shown to be a contraction in the p-Wasserstein distance and therefore admits a unique fixed point Z^π (Bellemare et al., 2017). A major challenge of distributional RL resides in the choice of representation for the action-value distribution, as well as the exact implementation of the distributional Bellman operator. For simplicity, in the remainder and in line with previous work, we only consider empirical distributions (Bellemare et al., 2023, Definition 5.5) (i.e., whose representation can fit in finite memory), and refer to the empirical representation distributional Bellman operator (Bellemare et al., 2023, Algorithm 5.1) as $\mathcal{T}^\pi$.

7.2.2 Quantile and expectile regression

Let Z be a real-valued probability distribution. The *α-quantile* q_α of Z is defined as a value splitting the probability mass of Z in two parts of weights α and $1 - \alpha$, respectively:

$$P(z \leq q_\alpha) = \alpha. \tag{7.3}$$

Therefore, the *quantile function* $Q_Z : \alpha \mapsto q_\alpha$ is the inverse cumulative distribution function: $Q_Z = F_Z^{-1}$. Alternatively, quantiles are given by the minimizer of an asymmetric L_1 loss:

$$q_\alpha = \arg\min_{q} \mathbb{E}_{z \sim Z}\left[\left(\alpha \mathbb{1}_{z > q} + (1 - \alpha)\mathbb{1}_{z \leq q}\right)|z - q|\right]. \tag{7.4}$$

Expectiles and the *expectile function* $E_Z : \tau \mapsto e_\tau$ are defined analogously, as the τ-expectile e_τ minimizes the asymmetric L_2 loss:

$$e_\tau = \arg\min_e \mathbb{E}_{z\sim Z}\left[(\tau \mathbb{1}_{z>e} + (1-\tau)\mathbb{1}_{z\leq e})\,(z-e)^2\right]. \tag{7.5}$$

7.2.3 Quantiles and expectiles in distributional RL

Quantile regression has been used for distributional RL in many previous studies (Dabney et al., 2018a; Dabney et al., 2018b; Yang et al., 2019) where a parameterized quantile function $Q_Z^\theta(s,a,\alpha)$ is trained using a quantile temporal difference loss function derived from Eq. (7.4), i.e., for N estimated quantiles:

$$\mathcal{L}_Q\left(Q_Z^\theta(s,a,\cdot)\right) = \sum_{i=1}^N \sum_{j=1}^N l_Q(q_i, z_j) \text{ with } l_Q(q_i,z_j) = \left(\alpha_i \mathbb{1}_{z_j>q_i} + (1-\alpha_i)\mathbb{1}_{z_j\leq q_i}\right)\left|z_j - q_i\right|, \tag{7.6}$$

where the trainable quantile values $q_i = Q_Z^\theta(s,a,\alpha_i)$ are obtained by querying the quantile function at various quantile fractions α_i, which can be fixed by the designer (Dabney et al., 2018b), sampled from a distribution (Dabney et al., 2018a), or learned during training (Yang et al., 2019). In quantile-based temporal difference (QTD) learning, the target samples z_j can be obtained by querying the estimated quantile function at the next state-action pair: $z_j = r + \gamma Q_Z^\theta(s',a',\alpha_j)$.[2] Indeed, because the true quantile function is the inverse CDF of the action-value distribution, Dabney et al. (2018b) and Bellemare et al. (2023) showed that, among N-atoms representations, quantiles at equidistant fractions minimize the 1-Wasserstein distance with the action-value distribution and that the resulting projected Bellman operator is a contraction mapping in such a distance. Rowland et al. (2023) extended these results to prove the convergence of QTD learning under mild assumptions. We refer to these studies for a more detailed convergence analysis.

In contrast, expectile-based temporal difference (ETD) learning does not allow the same training loss as the one given by Eq. (7.6). We first write the generic ETD loss derived from Eq. (7.5):

$$\mathcal{L}_E\left(E_Z^\theta(s,a,\cdot)\right) = \sum_{i=1}^N \sum_{j=1}^N l_E(e_i, z_j) \text{ with } l_E(e_i,z_j) = \left(\tau_i \mathbb{1}_{z_j>e_i} + (1-\tau_i)\mathbb{1}_{z_j\leq e_i}\right)\left(z_j - e_i\right)^2, \tag{7.7}$$

2 We can have $a' \sim \pi(\cdot \mid s')$, as in actor-critic algorithms, or $a' = \arg\max_a \mathcal{Q}_Z^\theta(s',a,\alpha_j)$ as in Q-learning. This section is agnostic to that choice but we refer to (Bellemare et al., 2023) for convergence analysis in the latter case.

with $e_i = E_Z^\theta(s, a, \tau_i)$. Here, choosing $z_j = r + \gamma E_Z^\theta(s', a', \tau_j)$, analogously to QTD learning and non-distributional TD learning, would cause the update to approximate a different distribution because the expectile function is in general not the inverse CDF of the return distribution, meaning that expectiles cannot be considered as samples from the distribution. Rowland et al. (2019) formalized this idea using the concept of *Bellman-closedness*, i.e., that the projected Bellman operator yields the same statistics whether it is applied to the target distribution or to the implicit distribution given by statistics of the target distribution (i.e., in our case a uniform mixture of diracs with locations given by quantiles or expectiles).

7.3 RELATED WORK

Distributional reinforcement learning has been shown to result in several benefits – by ascribing randomness to the value of a state-action pair, an algorithm can learn more efficiently for close states and actions (Mavrin et al., 2019), as well as capture possible stochasticity in the environment (Martin et al., 2020). Multiple families of approaches have emerged.

Estimating a parameterized distribution is a straightforward approach, and has been explored from both Bayesian (Strens, 2000; Vlassis et al., 2012) and frequentist (Jullien et al., 2023) perspectives. However, this usually requires an expensive likelihood computation, as well as making a restrictive assumption on the shape of the return distribution Z. For instance, assuming a normal distribution when the actual distribution is heavy-tailed can yield disappointing results.

Thus, approaches based on non-parametric estimation are also used to approximate the distribution. C51 (Bellemare et al., 2017) quantizes the domain where Z has non-zero density (usually in 51 atoms, hence the name), and performs weighted classification on the atoms, by computing the cross-entropy between Z and $\mathcal{T}^\pi Z$. While C51 increases performance over non-distributional RL, it requires the user to manually set the return bounds and is not guaranteed to minimize any p-Wasserstein metric with the target return distribution.

Another important non-parametric approach to the estimation of a distribution is quantile regression. Quantile regression relies on the minimization of an asymmetric L_1 loss. Estimating quantiles allows to approximate the action-

value distribution without relying on a shape assumption. QR-DQN (Dabney et al., 2018b) introduced quantile regression as a way to minimize the 1-Wasserstein metric between Z and $\mathcal{T}^{\pi}Z$. ER-DQN (Rowland et al., 2019) traded the estimation of quantiles for expectiles, at the cost of a potential distribution collapse, that they prevent via a root-finding procedure. Further, implicit quantile networks (IQN) (Dabney et al., 2018a) sample and embed quantile fractions, instead of keeping them fixed, thereby improving performance. Fully parameterized quantile functions (FQF) (Yang et al., 2019) add another network generating quantiles fractions to be estimated. We build on IQN and its expectile counterpart to propose a well-performing, non-collapsing agent.

7.4 METHOD

7.4.1 Dual training of quantiles and expectiles

It has been suggested that expectiles are more efficient than quantiles for function approximation (Newey and Powell, 1987; Waltrup et al., 2015), but unlike quantiles, they cannot be directly used to generate proper samples of the estimated return distribution (z_j in Eq. (7.7)), which are required in distributional dynamic programming. Rowland et al. (2019) proposed an *imputation strategy*, i.e., a way to generate samples of a distribution that matches the current set of estimated expectiles, by solving a convex optimisation problem. In our experiments, we found that applying this imputation strategy tends to drastically increase the runtime (around 25 times slower in our setup), making experimentation with such methods close to impossible for researchers with modest computing resources. In this paper, we propose to learn a functional mapping between expectiles and quantiles and use the predicted quantiles to generate samples.

We learn a single Z-function using expectile regression. Therefore, we have $\forall(s,a) \in \mathcal{S} \times \mathcal{A}, \tau \in [0,1],\ \hat{Z}(s,a,\tau) \triangleq E_{Z(s,a)}(\tau)$, where Z is the true Z-function we wish to estimate. Then, we note that for non-deterministic $Z(s,a)$, the expectile function at a given state-action pair $E_{Z(s,a)} \in \mathbb{R}^{[0,1]}$ is a strictly increasing and continuous function that spans the entire convex hull of the distribution's support (Holzmann and Klar, 2016). Meanwhile, the quantile function $Q_{Z(s,a)} \in \mathbb{R}^{[0,1]}$ spans the distribution's support. As a consequence, every quan-

tile is a single expectile, i.e., there exists a functional mapping from quantile fractions to expectile fractions. In this work, we propose to learn such a mapper $\hat{\mathcal{M}}(s,a,\tau) \triangleq E^{-1}_{Z(s,a)} \circ F^{-1}_{Z(s,a)}(\tau)$ using the quantile regression loss function from Equation (7.6). We now have $\forall (s,a) \in \mathcal{S} \times \mathcal{A}, \tau \in [0,1],\ \hat{Z}(s,a,\hat{\mathcal{M}}(s,a,\tau)) \triangleq Q_{Z(s,a)}(\tau)$. We can then simply query our estimator of quantiles at the next state-action pair to yield a sound imputation step, while the parameters of the Z-function are learned through expectile regression.

For any triplet (s_t, a_t, τ), our proposed update step can be described as follows:

1. Generate approximate samples of the target distribution using the quantile representation :

$$\hat{z} \sim R(s_t,a_t) + \gamma \hat{Z}(s_{t+1}, a_{t+1}, \hat{\mathcal{M}}(s_{t+1}, a_{t+1}, \tau))$$

2. Use expectile regression to learn the Z-function:

$$\hat{Z}(s_t,a_t,\tau) \leftarrow \mathrm{ER}\,(\hat{e},\hat{z}) \ \text{ with } \hat{e} \sim \hat{Z}(s_t,a_t,\tau)$$

3. Use quantile regression to learn the mapper:

$$\hat{\mathcal{M}}(s_t,a_t,\tau) \leftarrow \mathrm{QR}\,(\hat{q},\hat{z}) \ \text{ with } \hat{q} \sim \hat{Z}(s_t,a_t,\hat{\mathcal{M}}(s_t,a_t,\tau))$$

The state-action embeddings of the mapper are copied form those of the Z-function. This way, the parameters of the Z-function (including the large image embedding networks and the overall scale of the rewards) are learned using expectile regression, while only the residual shape difference between the quantile and expectile function is learned by the mapper, using quantile regression.

The update step described above can be formalized as a distributional operator, that we define in Section 7.4.2. We prove that our proposed update operator converges to the distributional Bellman operator in the limit of infinite estimated quantile/expectile fractions. Then, in Section 7.4.3, we detail a practical implementation of dual expectile-quantile RL based on implicit quantile networks that we name IEQN.

7.4.2 Convergence of the dual expectile-quantile operator

In this section, we prove that our proposed update operator converges to the distributional dynamic programming operator from Equation (7.2) as the num-

ber of quantiles and expectiles kept in memory grows infinitely large, i.e., that the error incurred by our dual expectile-quantile operator vanishes in the limit of an infinite number of statistics to be evaluated. This result relies on several properties of the expectile function, including its absolute continuity that we establish in the following lemma:

Lemma 1. *Let Z be a random variable taking values in $[a, b]$ with finite second moment and whose CDF admits finitely many discontinuities. Then, the expectile function $E_Z : \tau \mapsto \arg\min_e \mathbb{E}_{z\sim Z}\left[(\tau \mathbb{1}_{z>e} + (1-\tau)\mathbb{1}_{z\leq e})\,(z-e)^2\right]$ is absolutely continuous on $[0, 1]$.*

The proof is in Appendix 7.C. We are now able to prove our main result, Theorem 2, i.e., that our dual regression projection operator approximates the target distribution well in the limit of an infinite number of quantile/expectile fractions:

Theorem 2. *Let $\tau_k = \frac{2k-1}{2K}$, for $k = 1, \dots, K$, and let $\Pi^K_{\mathcal{M}} : \mathscr{P}(\mathbb{R}) \to \mathscr{P}(\mathbb{R})$ be the dual regression projection operator defined as:*

$$\forall \eta \in \mathscr{P}(\mathbb{R}),\ \ \Pi^K_{\mathcal{M}}(\eta) = \frac{1}{K}\sum_{k=1}^{K} \delta_{E_\eta\left(\text{floor}^K\left(E_\eta^{-1}(F_\eta^{-1}(\tau_k))\right)\right)} = \frac{1}{K}\sum_{k=1}^{K} \delta_{E_\eta\left(\frac{2\lfloor K\mathcal{M}(\tau_k)+1/2\rfloor-1}{2K}\right)}\,,$$

where $E_\eta : [0, 1] \to \mathbb{R}$ is the expectile function of η, $F_\eta^{-1} : [0, 1] \to \mathbb{R}$ is the inverse CDF – i.e., the quantile function – of η, and $\text{floor}^K(x) = \tau_{\lfloor Kx+\frac{1}{2}\rfloor}$. Let $\eta \in \mathscr{P}(\mathbb{R})$ be a bounded-support probability distribution with finite second moment and whose CDF admits finitely many discontinuities, and let W_1 be the 1-Wasserstein distance. Then:

$$\lim_{K\to\infty} W_1(\Pi^K_{\mathcal{M}}\eta, \eta) = 0\,.$$

Reusing the notation from the theorem, we can formally define our dual expectile-quantile operator. Let $\pi \in \mathscr{P}(\mathcal{A})^{\mathcal{S}}$ be a policy, we have:

$$\mathcal{T}^\pi_{\mathcal{M}^K} = \Pi^K_{\mathcal{M}}\mathcal{T}^\pi\,, \tag{7.8}$$

where $\mathcal{T}^\pi : Z(s_t, a_t) = R(s_t, a_t) + \gamma\mathbb{E}_\pi\left[Z(s^\pi_{t+1}, a^\pi_{t+1})\right]$ is the distributional Bellman operator (see Section 7.2.1). We can now derive a corollary in the context of distributional RL training:

On Markov decision processes with bounded rewards and $\gamma < 1$, the dual expectile-quantile operator converges pointwise to the distributional Bellman operator:

$$\lim_{K\to\infty} \mathcal{T}^\pi_{\mathcal{M}^K} = \mathcal{T}^\pi \text{ pointwise.}$$

This results comes in contrast to the failure of the naive expectile operator (Rowland et al., 2019) to match the distributional Bellman operator. We now present a practical implementation of an agent using our dual approach.

7.4.3 A practical implementation: IEQN

We use the principle described in Section 7.4.1 to implement IEQN (Algorithm 1), a new distributional RL agent based on implicit quantile networks (IQN) (Dabney et al., 2018a). The Z-function is modeled as a neural network inputting a state and a fraction $\tau \sim \mathcal{U}(0,1)$, and outputting τ-expectile values for all actions. Its parameters are learned via an asymmetric L_2 loss, i.e., expectile regression. We also use a neural network to implement the mapper between quantile fractions and expectile fractions, and learn its parameters via an asymmetric L_1 loss, i.e., quantile regression.

Algorithm 1 Implicit expectile-quantile networks (IEQN) update

Require: Z-function Z_θ, mapper m_ϕ, fractions $(\tau_i)_{i=1,\dots,N} \sim \mathcal{U}([0,1])$, learning rate λ.

Collect experience (s, a, r, s')

for $i = 1, \dots, N$ **do**

Compute expectile values $e_i \leftarrow Z_\theta(s, a, \tau_i)$ and quantile values $q_i \leftarrow Z_\theta(s, a, m_\phi(\tau_i))$

Compute the greedy next-action $a' \leftarrow \max_{b \in \mathcal{A}} \frac{1}{N} \sum_{i=1}^{N} Z_\theta(s', b, \tau_i)$

Compute target samples $z_i \leftarrow r + \gamma \cdot \text{stop\_grad}(Z_\theta(s', a', m_\phi(\tau_i)))$

end for

Compute expectile loss $\mathcal{L}_E \leftarrow \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\tau_i \mathbb{1}_{z_j > e_i} + (1 - \tau_i) \mathbb{1}_{z_j \leq e_i} \right) \left(z_j - e_i \right)^2$

Compute quantile loss $\mathcal{L}_Q \leftarrow \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\tau_i \mathbb{1}_{z_j > q_i} + (1 - \tau_i) \mathbb{1}_{z_j \leq q_i} \right) \left| z_j - q_i \right|$

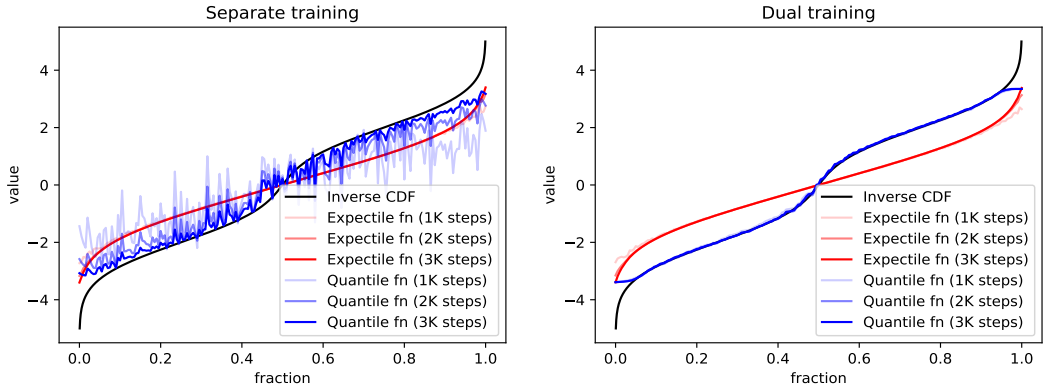
Update expectile function parameters $\theta \leftarrow \theta - \lambda \nabla_\theta \mathcal{L}_E$

Update mapper parameters $\phi \leftarrow \phi - \lambda \nabla_\phi \mathcal{L}_Q$

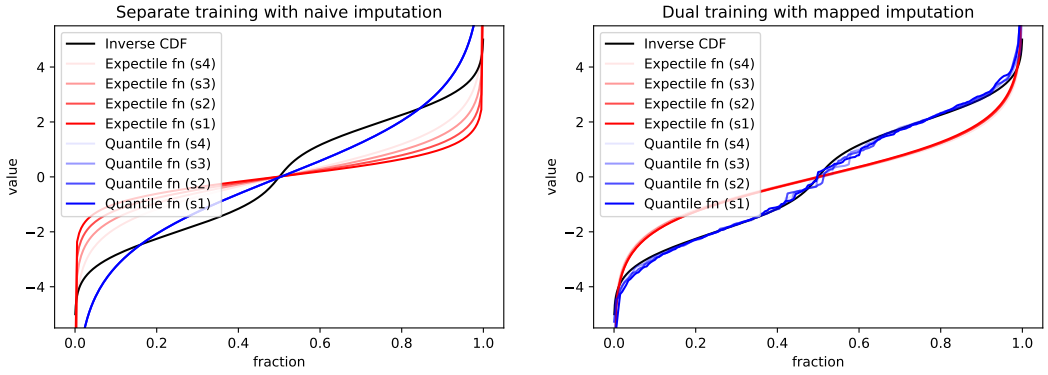
7.5 EXPERIMENTS

We first demonstrate on a toy MDP the benefits of learning quantiles and expectiles together. We then describe our experimental setup and results on the Atari Arcade Learning Environment (ALE).

7.5.1 *Chain MDP: A toy example*



(a) Approximating a distribution with separate and dual training.



(b) Tabular distributional RL with separate and dual training.

Figure 7.1: (a) Approximating a bimodal distribution with quantile and expectile regression. Quantile regression approximates the inverse CDF, albeit with high variance, especially on extreme values (left, blue curves). Expectiles converge very quickly to the expectile function (left, red curves). When training a mapper to generate quantiles from expectiles, quantile estimation becomes much more efficient (right). **(b)** Distributional RL with function approximation in a chain MDP with 4 states, and a bimodal reward distribution at the last state. The expectile function collapses as the temporal difference error propagates to previous states (left, red curves) while the quantile function is a poor approximation of the inverse CDF (left, blue curves). Our dual method solves both problems (right).

We start by observing the effect of our proposed operator in a toy environment. The MDP comprises 4 states, each pointing to the next through a unique action and without accumulating any reward, until the last state s_4, where the episode terminates and the agent obtains a reward sampled from a bimodal distribution $r \sim \left(\frac{1}{2}\mathcal{N}(-2,1) + \frac{1}{2}\mathcal{N}(+2,1)\right)$ (see Appendix 7.B for a visual description).

Figure 7.1a highlights the advantageous properties of expectile regression that were introduced in prior work (Philipps, 2021a; Philipps, 2021b; Waltrup et al., 2015). When trying to approximate the distribution of terminal rewards directly from samples (left), expectile regression yields more accurate estimates than quantile regression in the low-data regime (recall that the quantile function is the inverse CDF while the expectile function is in general not). Interestingly, coupling expectile regression with our mapper (right) allows us to recover the quantile function much more efficiently than quantile regression itself.

In Figure 7.1b (left), we illustrate the deficiencies of regular quantile and expectile dynamic programming. Quantile function learning is sample-inefficient and fails to approximate the distribution within the given evaluation budget. However, we observe that the distribution information is propagated correctly through temporal difference updates, since the quantile functions estimated at each state coincide. In contrast, the expectile function collapses to the mean as the error propagates from s_4 to s_1. This is due to the fact that expectile values at the next state-action pair cannot be used as pseudo-samples of the return distribution $Z(s_{t+1})$ (Rowland et al., 2019). Finally, Figure 7.1b (right) shows that our dual training method, where the pseudo-samples of $Z(s', a')$ are the estimated quantiles $Z_\theta(s_{t+1}, m_\phi(\tau))$, solves both issues: the expectile function does not collapse anymore and the quantile function approximation is more accurate.

7.5.2 Experiments on the Atari Arcade Learning Environment

Baselines

We experimented with the following baselines to evaluate our approach:

IQN-0, IQN-1 We approximate quantiles using the general approach described in IQN (Dabney et al., 2018a), respectively without and with a Huber loss.

IEN-Naive We use a similar approach as for IQN, but trained with an expectile loss and a naive imputation step as described in (Rowland et al., 2019), i.e., expectile values are used as target for the temporal difference loss. The solver-based implementation described by the authors was too slow on our setup, as it was approximately 25 times slower than the other baselines.

Environments

We opted to conduct our experiments with the Atari Learning Environment (ALE) (Bellemare et al., 2013), following the setup of Machado et al. (2018), notably including a 25% chance to perform a sticky action at each step, i.e., repeating the latest action instead of using the action predicted by the agent. This creates stochasticity in the environment, which should be captured by distributional RL agents. In order to accommodate for limited computing resources, we constrained ourselves to the Atari-5 subbenchmark (Aitchison et al., 2023), yet using 5 seeds to reduce the uncertainty in our results. We perform 25 validation episodes every 1M steps to generate our performance curves. As is common with ALE, we report human-normalized scores, rather than raw game scores, and we aggregate them using the interquartile mean (IQM), as it *is a better indicator of overall performance* (compared to sample median) (Agarwal et al., 2021), due to its robustness to scale across tasks and to outliers. It is especially needed, as the presence of sticky actions increases the number of outlier seeds.

Implementation details

We base all baselines and our method on the same underlying neural network, implemented in JAX (Bradbury et al., 2018). Its architecture follows the structure detailed by Dabney et al. (2018a). We used the training loop composition of CleanRL (Huang et al., 2022b). Hyperparameters can be found in Appendix 7.A.1. We implemented the Z-function for all agents as a feed-forward neural network with layer normalization. We did not use the fraction proposal network introduced with FQF (Yang et al., 2019), as our method can be seen as complementary to it, and we focus on the effect of the choice of statistics. Finally, we found that using layer normalization increased performance for both our method and baselines.

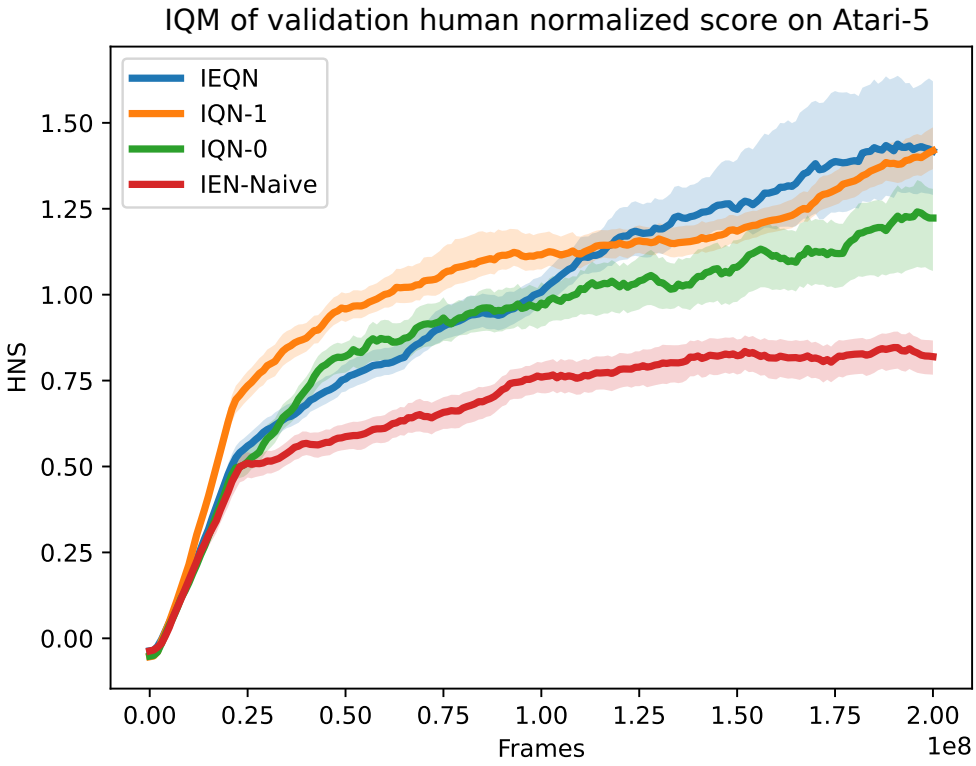


Figure 7.2: Interquartile mean of the human normalized score of distributional RL agents on the Atari5 benchmark with 5 random seeds per environment. Shaded areas correspond to the 25-th and 75-th percentiles of a bootstrap distribution. A rolling average with window size of $20M$ frames is performed to enhance readability.

As described in Algorithm 1, we only use the expectile loss to update the Z-function for our agent, while we use the quantile loss to update our mapper. The mapper is implemented as a two layer, residual fully-connected neural network with ReLU and Tanh activations. Since it is queried to obtain both the candidate and target values, we use a mapper-specific target network updated less frequently than the live network, using Polyak averaging (Polyak and Juditsky, 1992) with a weight of 0.5. We share the parameters across all states, to simplify its architecture. We detail the implications of this choice in Appendix 7.A.3.

Results

In this section, we verify that our dual approach also provides benefits at scale, on a classic benchmark.

Table 7.1: Average and standard deviation of the distance between quantile (respectively expectile) 0.1 and 0.9, relatively to the scale of the Q-function, at the end of training.

	Quantiles spread	Expectiles spread
IQN-0	1,25 ± 0,198	-
IQN-1	0,144 ± 0,072	-
IEN-Naive	-	0,174 ± 0,195
IEQN	0,721 ± 0,142	0,465 ± 0,086

We first present in Figure 7.2 the aggregated results over 5 seeds on the Atari5 benchmark. We can see that despite a slower start, IEQN ends up matching the performance of IQN-1. To get statistically stronger results, we also performed a bootstrap hypothesis test on the difference of IQMs at the end of training (we average scores from the last 5 validation epochs to be robust to instabilities). We found that our method surpasses the performance of both the quantile approach (achieved significance level 0.0117), and naive expectile approach (achieved significance level 0), thereby demonstrating the benefits of dual regression over single regression of either quantiles or expectiles on the final performace.

Furthermore, we verify in Table 7.1 that IEQN avoids distributional collapse in practice. In fact, while IQN-1's estimated distribution is much narrower than IQN-0's – a confirmation that the Huber loss causes distributional collapse, IEQN's quantile spread much larger than IQN-1's. Moreover, the expectile spread of IEQN is much larger and stable than that of IEN-Naive, suggesting that expectile distributional RL yields degenerate distributions, as noted in (Rowland et al., 2019), but that dual expectile-quantile distributional RL avoids this collapse.

7.6 CONCLUSION

We proposed a statistics-based approach to distributional reinforcement learning that uses the simultaneous estimation of quantiles and expectiles of the action-value distribution. This approach presents the advantage of leveraging the efficiency of the expectile-based loss for both expectile and quantile

estimation while solving the theoretical shortcomings of expectile-based distributional reinforcement learning, which often lead to a collapse of the expectile function in practice.

We showed on a toy environment how the dual optimization affects the statistics recovered in distributional RL: in short, the quantile function is estimated more accurately than with vanilla quantile regression and the expectile function remains consistent after several steps of temporal difference training. We then benchmarked our approach at scale, on the Atari5 benchmark. Our model, IEQN, matched the performance of the Huber-based IQN-1 and surpassed that of both expectile and quantile-based agents, demonstrating its effectiveness in practical scenarios.

We open possibilities for future research to use a distributional approach that performs well and does not collapse. For future work, we plan to investigate how the dual approach can be used in risk-aware decision-making problems, and how it performs when the goal is to optimize risk metrics such as (conditional) value-at-risk. Moreover, we plan to gather insights into what type of behavior is favored by the quantile and expectile loss, respectively.

7.7 REFLECTIONS ON THE CHAPTER

7.7.1 Research outcomes

This chapter serves towards answering my final research question:

Research Question 5. *How can we train reinforcement learning algorithms to handle high degrees of uncertainty, which is common in interactive recommender systems?*

Distributional reinforcement learning is a promising approach for training RL agents under uncertainty, but this chapter showed that properly capturing the environmental stochasticity, let alone act with respect to it, is far from trivial. We proposed a theoretically valid but efficient approach to the distributional value estimation problem. More work is now needed to find principled ways to act with respect to the estimated uncertainty, and to study the effects of doing so on the behavior of recommender systems.

7.7.2 Additional thoughts

Expectiles possess a range of interesting properties that we have not used in this chapter. They provide a natural measure of heteroskedasticity, which enables calibrated regression on heteroskedastic data (Barry et al., 2022). They allow us to better study the effect of binary variables than quantiles, as the expectile function of a binary variable spans the whole $[0, 1]$ interval, whereas quantiles have to be either 0 or 1. An interesting corrolary is the ability of expectiles to capture intersectional effects: Philipps (2021a) uses expectile regression to study the effect of being Black on mortgage approval rates given the individual is likely (high expectile) or unlikely (low expectile) to be approved given their other personal data. Expectile regression has also been used as a way to regress a soft maximum other data points, by approximating the regression line of an expectile fraction close to 1 (Kostrikov et al., 2022). Finally, expectiles have been used in finance applications as they yield risk measures with favorable properties, compared to metrics based on quantiles like CVaR (Chen, 2018).

All of these properties make expectile regression a valuable tool that, I believe, is underexplored and could have practical applications in many fields, including information retrieval.

CHAPTER APPENDIX

7.A HYPERPARAMETERS, CODE AND IMPLEMENTATION DETAILS

7.A.1 Hyperparameters

We use JAX (Bradbury et al., 2018) to train our models. A full training procedure of 200M training frames and corresponding validation epochs takes approximately 50 hours in our setup.

Table 7.A.1: Z-function hyperparameters.

Key	Value
Discount factor	0.99
Batch size	32
Fraction distribution	$\mathcal{U}([0, 1])$
Learning rate	$1e^{-4}$
Random frames before training	200000
Size of convolutional layers	$[32, 64, 64]$
Size of fully-connected layer	512
Critic updates per sample	2
Buffer size	1e6
Frames between target network updates	35000
Target network update rate	1.0

Table 7.A.2: Mapper hyperparameters.

Key	Value
Layer size	64
Learning rate	$7e^{-5}$
Target network update rate	0.5

7.A.2 Code

Our training and evaluation loop is based on CleanRL (Huang et al., 2022b). The anonymized code base is available on `https://anonymous.4open.science/r/ieqn`.

7.A.3 Sharing the mapper's parameters

Sharing the mapper's parameters across states and actions allows to lighten the computational burden, as it is part of the goal of this chapter. We found this technique to work well in practice on the Atari5 benchmark, although it requires additional assumptions in theory. We review these assumptions in this section.

Yao and Tong (1996) showed that there exists such a shared mapping between quantile and expectile a when the regression follows a location-scale model, i.e., for random variables X and Y:

$$Y = \mu(X) + \sigma(X)\varepsilon \,,$$

where μ and σ are continuous functions, ε is centered and finite-variance, and ε, X are independent. When the return distribution follows this model, X being the state-action variable in this context, sharing the mapper's parameters is theoretically valid. While this may seem limiting, it does not require all state-action pairs to be allocated the same distributions, only that they share a common shape. Moreover, the location-scale family is quite broad, as it includes, e.g., Normal, Student, Cauchy, GEV distributions, and more.

In many distributional reinforcement learning scenarios, the assumption may be satisfied. For instance, when the environmental stochasticity emerges from small, independent perturbations, i.e., normally-distributed errors, the return distribution at every state will still be normally distributed as convo-

lutions of Gaussian distributions are also Gaussian. On the other hand, this assumption can fail under high-frequency transition distributions, i.e., branching behaviors, where the same state-action pair can yield drastically different outcomes and the reward-next-state distribution has non-continuous support. We leave for future work the investigation of when sharing the mapper's parameters across state-action pairs fails in practice.

7.B TOY MARKOV DECISION PROCESS

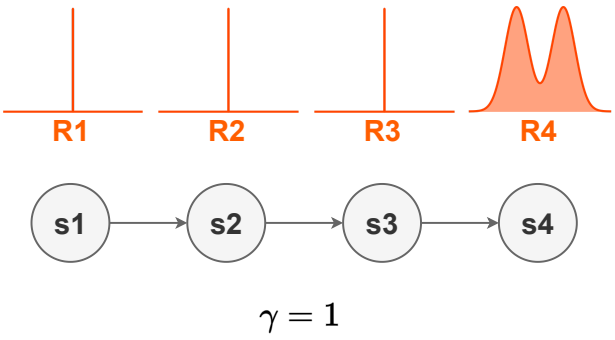


Figure 7.B.1: Toy Markov decision process

7.B.1 Expectile regression

Expectiles were originally introduced as a family of estimators of *location parameters* for a given distribution, to palliate possible heteroskedasticity of the error terms in regression (Newey and Powell, 1987; Philipps, 2021a).

Expectiles can be seen as mean estimators under missing data (Philipps, 2021b). Unlike quantiles, they span the entire convex hull of the distribution's support, and on this ensemble, the expectile function is strictly increasing: an expectile fraction is always associated to a unique value. Expectiles have been used in reinforcement learning successfully before (Rowland et al., 2019), but in a way that requires a slow optimization step to achieve satisfactory performance. Moreover, expectile regression is subject to the same crossing issue as quantiles, albeit empirically less so (Waltrup et al., 2015). Expectiles have also

been used in offline reinforcement learning to compute a soft maximum over potential outcomes seen in the offline data (Kostrikov et al., 2022).

7.C PROOFS

Our proof of Theorem 2 requires the absolute continuity of the expectile function. Therefore, we first prove the following lemma:

Lemma 1. *Let Z be a random variable taking values in $[a, b]$ with finite second moment and whose CDF admits finitely many discontinuities. Then, the expectile function $E_Z : \tau \mapsto \arg\min_e \mathbb{E}_{z\sim Z}\left[(\tau \mathbb{1}_{z>e} + (1-\tau)\mathbb{1}_{z\leq e})\,(z-e)^2\right]$ is absolutely continuous on $[0, 1]$.*

Proof. Our proof relies on the Banach-Zarecki theorem (Zaretsky, 1925), which states that any real-valued function f defined on a real bounded closed interval is absolutely continuous if and only if on this interval:

(i) f is continuous;
(ii) f has bounded variation;
(iii) f follows the Luzin N property (Luzin, 1915), i.e., the image by f of a set with null Lebesgue measure also has null Lebesgue measure.

It is well-known that the expectile function is continuous on $[0, 1]$ (Holzmann and Klar, 2016; Philipps, 2021b). Therefore, (i) is satisfied.

E_Z is monotonically increasing and takes values in the finite support of Z. Therefore it has bounded variation and (ii) is satisfied.

In order to prove (iii), we first note that any function that is differentiable on a co-countable set has the Luzin N property (Luzin, 1915). We therefore use our assumption that Z admits a finite number of discontinuities in the following.

Let F_Z be the CDF of Z and $D = \{z \in [a, b]\ ,\ \lim_{x\to z} F_Z(x) \neq F_Z(z)\}$ be the finite set of points at which F_Z is not continuous. D is a finite set within a metric space and therefore closed. As a consequence, its complement $C_{[a,b]} = [a, b]\backslash D$ is open in $[a, b]$, i.e., $\forall z \in C_{[a,b]}, \exists \varepsilon > 0$ such that $\forall x \in [a, b]\ ,\ d(x, z) < \varepsilon \Rightarrow x \in C_{[a,b]}$. In other words, if F_Z is continuous at a point within $[a, b]$, it is also continuous in a neighborhood of that point within $[a, b]$. By assumption, the set $C^{\mathcal{N}}_{[a,b]} = \left\{z \in [a, b]\ ,\ \exists \varepsilon > 0\ ,\ \forall x \in [a, b]\ ,\ d(x, z) < \varepsilon \Rightarrow x \in C_{[a,b]}\right\}$ of points where F_Z is continuous in a neighborhood of said point is therefore co-finite.

It has been shown that the expectile function E_Z is continuously differentiable at any point $\tau \in [0,1]$ such that F_Z is continuous in a neighborhood of $E_Z(\tau)$ (Holzmann and Klar, 2016; Newey and Powell, 1987). The expectile function is bijective (Philipps, 2021b) so the set of points where E_Z is differentiable $\mathcal{D}^{E_Z}_{[a,b]} = E_Z^{-1}\left(C^{\mathcal{N}}_{[a,b]}\right)$ is also a co-finite set.

The expectile function is differentiable on a co-finite (and thus co-countable) set, i.e., it has the Luzin N property (Luzin, 1915), which yields (iii).

We can finally apply the Banach-Zarecki theorem and conclude that the expectile function E_Z is absolutely continuous on $[0,1]$. □

We can now use the absolute continuity of the expectile function under our assumptions to prove the following theorem:

Theorem 2. *Let $\tau_k = \frac{2k-1}{2K}$, for $k = 1, \dots, K$, and let $\Pi^K_{\mathcal{M}} : \mathscr{P}(\mathbb{R}) \to \mathscr{P}(\mathbb{R})$ be the dual regression projection operator defined as:*

$$\forall \eta \in \mathscr{P}(\mathbb{R}),\ \ \Pi^K_{\mathcal{M}}(\eta) = \frac{1}{K}\sum_{k=1}^{K} \delta_{E_\eta\left(\text{floor}^K\left(E_\eta^{-1}(F_\eta^{-1}(\tau_k))\right)\right)} = \frac{1}{K}\sum_{k=1}^{K} \delta_{E_\eta\left(\frac{2\lfloor K\mathcal{M}(\tau_k)+1/2\rfloor-1}{2K}\right)}\,,$$

where $E_\eta : [0,1] \to \mathbb{R}$ is the expectile function of η, $F_\eta^{-1} : [0,1] \to \mathbb{R}$ is the inverse CDF – i.e., the quantile function – of η, and $\text{floor}^K(x) = \tau_{\lfloor Kx+\frac{1}{2}\rfloor}$. Let $\eta \in \mathscr{P}(\mathbb{R})$ be a bounded-support probability distribution with finite second moment and whose CDF admits finitely many discontinuities, and let W_1 be the 1-Wasserstein distance. Then:

$$\lim_{K\to\infty} W_1(\Pi^K_{\mathcal{M}}\eta, \eta) = 0\,.$$

Proof. Thanks to the triangle inequality, we have :

$$W_1(\Pi^K_{\mathcal{M}}\eta, \eta) \leqslant W_1(\Pi^K_{\mathcal{M}}\eta, \Pi^K_{Q}\eta) + W_1(\Pi^K_{Q}\eta, \eta)\,, \tag{7.9}$$

where Π^K_Q is the projected quantile regression estimator defined as:

$$\forall \eta \in \mathscr{P}(\mathbb{R}),\ \ \Pi^K_{Q}(\eta) = \frac{1}{K}\sum_{k=1}^{K} \delta_{F_\eta^{-1}(\tau_k)}\,.$$

Rowland et al. (2019, Lemma 3.2) showed that we have $W_1(\Pi_Q^K \eta, \eta) = \mathcal{O}\left(\frac{1}{K}\right)$. We now turn to the first term:

$$\begin{aligned} W_1(\Pi_{\mathcal{M}}^K \eta, \Pi_Q \eta) &= \sum_{i=0}^{K-1} \frac{1}{K} \left| E_\eta \left(\text{floor}^K \left(E_\eta^{-1} \left(F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right) \right) \right) - F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right| \\ &= \sum_{i=0}^{K-1} \frac{1}{K} \left| E_\eta \left(\text{floor}^K \left(E_\eta^{-1} \left(F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right) \right) \right) - \right. \\ &\qquad \left. E_\eta \left(E_\eta^{-1} \left(F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right) \right) \right| \\ &\leqslant \sum_{i=0}^{K-1} \frac{1}{K} \left| E_\eta \left(\text{floor}^K \left(E_\eta^{-1} \left(F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right) \right) \right) - \right. \\ &\qquad \left. E_\eta \left(\text{floor}^K \left(E_\eta^{-1} \left(F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right) \right) + \frac{1}{K} \right) \right| , \end{aligned} \tag{7.10}$$

where the last inequality is obtained thanks to the monotonicity of the expectile function. By absolute continuity of the expectile function under our assumptions (proven in Lemma 1), we have:

$$\begin{aligned} \lim_{K \to \infty} \left| E_\eta \left(\text{floor}^K \left(E_\eta^{-1} \left(F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right) \right) \right) - \right. \\ \left. E_\eta \left(\text{floor}^K \left(E_\eta^{-1} \left(F_\eta^{-1} \left(\frac{2i+1}{2K} \right) \right) \right) + \frac{1}{K} \right) \right| = 0, \end{aligned} \tag{7.11}$$

from which we can deduce $\lim_{K \to \infty} W_1(\Pi_{\mathcal{M}}^K \eta, \Pi_Q \eta) = 0$ and finally

$$\lim_{K \to \infty} W_1(\Pi_{\mathcal{M}}^K \eta, \eta) = 0$$

. □

Finally, we can derive our main result for the use of distributional dynamic programming with both quantiles and expectiles: On Markov decision processes with bounded rewards and $\gamma < 1$, the dual expectile-quantile operator converges pointwise to the distributional Bellman operator:

$$\lim_{K \to \infty} \mathcal{T}_{\mathcal{M}^K}^\pi = \mathcal{T}^\pi \text{ pointwise.}$$

Proof. We have $\mathcal{T}_{\mathcal{M}^K}^\pi = \Pi_{\mathcal{M}}^K \mathcal{T}^\pi$. Bellemare et al. (2023) have shown that the set of empirical distributions $\mathcal{F}_E$ is closed under the operator $\mathcal{T}^\pi$ (Proposition 5.7). Thus, for any empirical return distribution $\eta \in \mathcal{F}_E$, $\mathcal{T}^\pi \eta$ is also

empirical and its CDF admits finitely many discontinuities. Moreover, it has bounded support. Indeed, if without loss of generality we consider that the reward distribution take values in $[0, R_{\max}]$, we have that every possible return distribution η takes values in $[0, \frac{R_{\max}}{1-\gamma}]$, and therefore $\mathcal{T}^\pi \eta$ takes values in $[0, R_{max} + \gamma \frac{R_{\max}}{1-\gamma}] = [0, \frac{R_{\max}}{1-\gamma}]$.

We can now apply Theorem 2:

$$\forall \eta \in \mathcal{F}_E \, , \lim_{K \to \infty} W_1(\Pi^K_{\mathcal{M}} \mathcal{T}^\pi \eta, \mathcal{T}^\pi \eta) = 0 \, ,$$

and the result immediately follows. □

8

CONCLUSION

Recommender systems are and will continue to be imperfect. Having explored many challenges of recommender systems, I now realize how easily the small amount of available signal gets lost in noise and bias. Moreover, we cannot directly observe how the systems we build are doing; we must resort to metrics, which often comes with caveats. This makes it especially hard to hold the course of scientific progress. A natural reaction to that complexity would be to turn to simplistic approaches, that you could expect to be less prone to unexpected failures. Yet, simple, static, non-debiasing approaches are far from exempt of failure modes, as we have seen throughout the thesis (see, e.g., Chapter 3 or Chapter 4). Furthermore, we have observed multiple occurrences of meaningful signals being detected by approaches that cannot fully capture the complexity of human interaction with recommender systems (see, e.g., Chapter 5 or Chapter 6). Therefore, I argue that acknowledging the complexity of human behavior on online platforms, and learning to manage it, is a viable way of making recommender systems less imperfect, as long as the proper tools are used to ensure robust and reliable progress.

In this thesis I tried to provide a toolbox for recommender systems that are aware of the distribution shifts involved with offering recommendations to a person, and with deploying a new version of the model. A major requirement for ensuring progress is a reliable evaluation setup. In Chapters 2 to 5, we investigated how robust the evaluation practices traditionally used for static recommender systems are in this new, augmented setting. We propose metrics and protocols that address the limitations we found. In Chapters 6 and 7, we propose tools for adapting off-the-shelf dynamic RL-based approaches to the

very specific and challenging recommendation setting. I will now answer the research questions defined in the introduction, in light of our findings.

8.1 SUMMARY OF FINDINGS

Research Question 1. *How can we evaluate recommender systems in a way that accounts for their dynamic and interactive nature?*

In Chapter 2, we saw that next-item prediction, the most prevalent evaluation protocol for sequential recommendation, is not adapted to evaluating the interactions between recommender systems and dynamic users. Essentially, next-item prediction consists in building an idealized, static representation of a user and making sure that our models match the idealized user preferences. However, recommendation is not only a matching task, but also a sequential control task, which must be evaluated on the key variables of choice (e.g., click-through rate, user satisfaction) resulting from the whole sequence of interactions. This is most easily done through A/B tests on a live recommender system, but doing so is often costly and simply inaccessible for many academic researchers. We therefore identified multiple possibilities for offline evaluation that each have strengths and weaknesses and can be used together to enhance reliability: counterfactual off-policy evaluation, intermediate model evaluation, uncertainty-aware evaluation and simulators. In Chapter 3, we proposed such a simulator suitable for research in recommender systems with dynamic users.

In Chapter 4, we identified another major hurdle in the way of evaluating the performance of recommender systems under the changes they induce themselves: models trained on biased data can largely replicate these biases and therefore fail on certain downstream tasks after deployment. This issue goes undetected by current evaluation metrics, which are once again based on a static, idealized representation of a user, this time often based on expert annotations.

Research Question 2. *Can we predict in a fully offline manner the performance of models learned on biased data?*

In Chapter 4, our empirical analysis across a wide range of semi-synthetic benchmarks revealed that models trained on implicit feedback data can retain

residual bias and therefore come with disappointing performance after deployment. We then found in Chapter 5 that distributional metrics, i.e., metrics that consider the entire distributions of predictions and labels, can help detect biases in the learned models. We proposed one such metric, CMIP, and showed that it is a good predictor of downstream performance for models trained on biased data. We can therefore answer the question positively: while our proposed approach is not foolproof and does not detect all types of residual bias, it shows that it is possible to go beyond static evaluation and better predict the downstream performance of models before deploying them.

Research Question 3. *When do we need assumptions on user behavior, and how can we test for the validity of these assumptions?*

Chapter 4 showed that assumptions on user behavior, e.g., the examination hypothesis, are required to learn effective click and relevance predictors on biased implicit feedback data. Moreover, we found that simple models with fewer parameters and flexibility in their structure (e.g., PBM, UBM) tend to be more robust under the distribution shift that comes with deploying them, even when the true user behavior is more complex. This observation reflects a dilemma: we must find a trade-off between realism of the chosen user assumptions and robustness of the model. Our proposed CMIP metric constitutes a tool that helps practitioners choose the most appropriate set of assumptions, without bearing the cost of performing online evaluation on the downstream task.

With a different perspective, in Chapter 6, we investigated when assumptions about the user are not required. We proposed an online reinforcement learning agent that effectively learns to recommend slates (i.e., lists) of items without the need for assumptions on the user behaves and how the slate can be composed, despite the combinatorial nature of slate recommendation.

Research Question 4. *How to train reinforcement learning agents that recommend slates of items to users effectively and efficiently?*

In Chapters 3 and 6, we showed that appropriate action representation learning is key to the success of RL-based slate recommender systems. In particular, continuous relaxations of the discrete but combinatiorally large slate action space seems to perform well. Even a simple top-K approach that uses nearest neighbor search on item embeddings is often hard to beat. However, its performance is largely dependent on the quality of the learned item embeddings

and the complexity of the task. The flexible GeMS approach is overall more robust to these factors.

Research Question 5. *How can we train reinforcement learning algorithms to handle high degrees of uncertainty, which is common in interactive recommender systems?*

In Chapter 3, we found that in a typical recommendation scenario, environmental uncertainty is a major hurdle for reinforcement learning agents. This is due to both the very poor observability of the user preference and mindset – interactions logs convey very little information – and to noise in the user feedback. While we identified agents that are more robust to such uncertainty, e.g., agents employing a transformer-based state encoder, developing models and algorithms that handle uncertainty well is crucial for recommender systems.

In Chapter 7, we investigated the use of distributional reinforcement learning algorithms, as they are a natural tool for handling uncertainty. We found that these algorithms can indeed better handle environmental uncertainty. But to enable full distribution estimation and therefore proper uncertainty estimation, they must retain certain theoretical guarantees. In particular, we found that approaches using L_2 loss functions, despite their strong empirical performance when estimating the expected value of the return distribution, lose their ability to accurately estimate other points of the distribution, contrary to L_1-based loss functions. We therefore proposed a method that fuses the effectiveness of L_2-based loss functions with the distributional estimation guarantees of L_1-based loss functions.

8.2 FUTURE WORK

Finally, I would like to take a forward look at the future of recommender systems and their dynamic aspect. With the sustained use of recommendation-powered platforms and the emerging popularity of short content services, there is little doubt that recommender systems will need to account for the long-term satisfaction of their users across hundreds of thousands of interactions, in order to provide a satisfying user experience. This is first and foremost a concern for people using recommendation-powered platforms, who look for an enriching, entertaining, and mind-opening experience thanks to online content. It is also required for all major platforms as user retention is a well-known chal-

lenge (Anderson et al., 2020), and as they come under increased scrutiny due to the potential detrimental effects of filtering information with recommender systems (European Commission, 2024). I now list three directions that I consider to be promising and relevant for future recommender systems.

Improved capabilities for reinforcement learning agents Firstly, and unsurprisingly, more work is needed to make dynamic recommender systems more effective in real-world scenarios. This will likely involve training agents in an offline manner, and deploying them in a zero-shot or few-shot scenario. Offline reinforcement learning is known to be challenging, and its application to recommender systems may require additional efforts. It is still an open question how to integrate the inductive biases prevalent in information retrieval (either known from user studies or learned via bias estimation or click modeling techniques) into offline RL agents. Moreover, to properly account for the high uncertainty of the recommendation task, we currently lack theoretically sound and empirically effective methods for risk-aware control. Such methods could be used internally, for instance to provide exploration heuristics or avoid out-of-distribution situations, or to provide a better user experience, for instance through confidence assessments or to adapt the recommendations to the user's mood (exploratory or conservative). More work is also needed to balance the effectiveness and efficiency of RL-based recommender systems, especially as recommender system scenarios can scale to millions or billions of items in the catalog.

Richer representations Secondly, and while this thesis was mostly concerned with reasoning capabilities, rather than representation capabilities, a lot of potential improvements lie in richer representations. One way of doing so is to include more information as input to the model: content of the item, metadata, richer feedback, etc. Many datasets already include such information, but more work on the evaluation is required to enable progress on this aspect. Models can also enrich their capabilities, for instance by modeling more different and complex biases, by being able to handle more modalities or by leveraging pretrained representations. Finally, I believe that semantic representations which incorporate this information in a way that is tailored to the recommendation task will be required in order to enable agents with both strong reasoning capabilities and rich representations.

Multi-objective optimization Thirdly, more objectives can and should be considered to provide a better user experience. In this thesis, only the cumulative number of clicks was considered as an objective. We showed that by optimizing the whole sequence of recommendations, rather than the immediate value, even such a utility-based objective can yield emergent properties such as diversity, as it is a means to provide sustained satisfaction to the users. Yet, it is possible and often desirable to explicitly set different objectives: fairness metrics, churn rate, etc. Considering the dynamic aspect of recommender systems is both a challenge and an opportunity for some of these objectives: for instance, while sparse rewards like churn rate are notoriously difficult to handle for RL algorithms, multi-step optimization opens the possibility to achieve fairness on the whole sequence of interactions rather than for each recommendation. Finally, how to optimize for several of these objectives at the same time remains largely under-explored, both from a theoretical and empirical perspective.

BIBLIOGRAPHY

Afsar, M. Mehdi, Trafford Crump, and Behrouz Far (2022). "Reinforcement Learning based Recommender Systems: A Survey." In: *ACM Comput. Surv.* 55.7. ISSN: 0360-0300. DOI: 10.1145/3543846. URL: https://doi.org/10.1145/3543846.

Agarwal, Aman, Xuanhui Wang, Cheng Li, Michael Bendersky, and Marc Najork (2019). "Addressing Trust Bias for Unbiased Learning-to-Rank." In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 4–14. ISBN: 9781450366748. DOI: 10.1145/3308558.3313697. URL: https://doi.org/10.1145/3308558.3313697.

Agarwal, Rishabh, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare (2021). "Deep reinforcement learning at the edge of the statistical precipice." In: *Advances in Neural Information Processing Systems* 34.

Aggarwal, Charu C. (2016). *Recommender Systems: The Textbook*. 1st. Springer Publishing Company, Incorporated. ISBN: 3319296574.

Agrawal, Shipra and Navin Goyal (2013). "Thompson Sampling for Contextual Bandits with Linear Payoffs." In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, pp. 127–135. URL: https://proceedings.mlr.press/v28/agrawal13.html.

Ai, Qingyao, Tao Yang, Huazheng Wang, and Jiaxin Mao (Feb. 2021). "Unbiased Learning to Rank: Online or Offline?" In: *ACM Trans. Inf. Syst.* 39.2. ISSN: 1046-8188.

Aitchison, Matthew, Penny Sweetser, and Marcus Hutter (2023). "Atari-5: Distilling the Arcade Learning Environment down to Five Games." In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 421–438.

Anderson, Ashton, 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*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2155–2165. ISBN: 9781450370233. DOI: 10.1145/3366423.3380281. URL: https://doi.org/10.1145/3366423.3380281.

Andrade, Chittaranjan (2018). "Internal, External, and Ecological Validity in Research Design, Conduct, and Evaluation." In: *Indian Journal of Psychological Medicine* 40 (5), pp. 498–499.

Argenson, Arthur and Gabriel Dulac-Arnold (2021). *Model-Based Offline Planning*. arXiv: 2008.05556 [cs.LG].

Athey, Susan, Raj Chetty, Guido W Imbens, and Hyunseung Kang (2019). *The Surrogate Index: Combining Short-Term Proxies to Estimate Long-Term Treatment Effects More Rapidly and Precisely*. Tech. rep. National Bureau of Economic Research.

Bai, Xueying, Jian Guan, and Hongning Wang (2019). "A Model-Based Reinforcement Learning with Adversarial Training for Online Recommendation." In: *NeurIPS '19*, pp. 10734–10745.

Bakshy, Eytan, Solomon Messing, and Lada Adamic (2015). "Exposure to Ideologically Diverse News and Opinion on Facebook." In: *Science* 348.6239, pp. 1130–1132.

Barry, Amadou, Karim Oualkacha, and Arthur Charpentier (2022). "A new GEE method to account for heteroscedasticity using asymmetric least-square regressions." In: *Journal of Applied Statistics* 49.14, pp. 3564–3590. DOI: 10.1080/02664763.2021.1957789. URL: https://doi.org/10.1080/02664763.2021.1957789.

Bellemare, Marc G., Will Dabney, and Rémi Munos (2017). "A distributional perspective on reinforcement learning." In: *ICML*. PMLR, pp. 449–458.

Bellemare, Marc G., Will Dabney, and Mark Rowland (2023). *Distributional Reinforcement Learning*. MIT Press.

Bellemare, Marc G., Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The Arcade Learning Environment: An Evaluation Platform for General Agents." In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Ben-Shimon, David, Michael Friedmann, Alexander Tsikinovsky, Johannes Hörle, Lior Rokach, and Bracha Shapira (2015). *RecSys Challenge 2015*. URL: https://recsys.acm.org/recsys15/challenge/.

Borisov, Alexey, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov (2016). "A Neural Click Model for Web Search." In: *Proceedings of the 25th International Conference on World Wide Web*. WWW '16. Montréal, Québec, Canada: IW3C2, 531–541. ISBN: 9781450341431.

Borisov, Alexey, Martijn Wardenaar, Ilya Markov, and Maarten de Rijke (2018). "A Click Sequence Model for Web Search." In: *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '18. Ann Arbor, MI, USA: ACM, 45–54. ISBN: 9781450356572.

Botteghi, Nicolò, Mannes Poel, Beril Sirmaçek, and Christoph Brune (2021). "Low-Dimensional State and Action Representation Learning with MDP Homomorphism Metrics." In: *arXiv:2107.01677*.

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. URL: http://github.com/google/jax.

Brandfonbrener, David, Will Whitney, Rajesh Ranganath, and Joan Bruna (2021). "Offline RL Without Off-Policy Evaluation." In: *NeurIPS*, pp. 4933–4946.

Bruch, Sebastian, Shuguang Han, Michael Bendersky, and Marc Najork (2020). "A Stochastic Treatment of Learning to Rank Scoring Functions." In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. WSDM '20. Houston, TX, USA: Association for Computing Machinery, 61–69. ISBN: 9781450368223. DOI: 10.1145/3336191.3371844. URL: https://doi.org/10.1145/3336191.3371844.

Burges, Christopher J. C. (2010). *From RankNet to LambdaRank to LambdaMART: An Overview*. Tech. rep. Microsoft Research. URL: http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf.

Chaimalas, Iason, 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*. RecSys '23. Singapore, Singapore: Association for Computing Machinery, 715–722. ISBN: 9798400702419. DOI: 10.1145/3604915.3608820. URL: https://doi.org/10.1145/3604915.3608820.

Chamsi Abu Quba, Rana, Salima Hassas, Hammam Chamsi, and Usama Fayyad (June 2014). "From a "Cold" to a "Warm" Start in Recommender systems." In: *IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2014)*. Parma, Italy: IEEE Digital Library, pp. 1–7. DOI: 10.1109/WETICE.2014.66. URL: https://hal.science/hal-01301058.

Chandak, Yash, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas (2019). "Learning Action Representations for Reinforcement Learning." In: *ICML '19*, pp. 941–950.

Chandar, Praveen, Brian St. Thomas, Lucas Maystre, Vijay Pappu, Roberto Sanchis-Ojeda, Tiffany Wu, Ben Carterette, Mounia Lalmas, and Tony Jebara (2022). "Using Survival Models to Estimate User Engagement in Online Experiments." In: *WWW '22*, 3186–3195.

Chapelle, Olivier and Yi Chang (2011). "Yahoo! Learning to Rank Challenge Overview." In: *Proceedings of the Learning to Rank Challenge*. Ed. by Olivier Chapelle, Yi Chang, and Tie-Yan Liu. Vol. 14. Proceedings of Machine Learning Research. Haifa, Israel: PMLR, pp. 1–24. URL: https://proceedings.mlr.press/v14/chapelle11a.html.

Chapelle, Olivier, Donald Metlzer, Ya Zhang, and Pierre Grinspan (2009). "Expected Reciprocal Rank for Graded Relevance." In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. CIKM '09. Hong Kong, China: Association for Computing Machinery, 621–630. ISBN: 9781605585123. DOI: 10.1145/1645953.1646033. URL: https://doi.org/10.1145/1645953.1646033.

Chapelle, Olivier and Ya Zhang (2009). "A Dynamic Bayesian Network Click Model for Web Search Ranking." In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. Madrid, Spain: ACM, 1–10. ISBN: 9781605584874. DOI: 10.1145/1526709.1526711.

Chen, Hung-Hsuan, Chu-An Chung, Hsin-Chien Huang, and Wen Tsui (2017). "Common Pitfalls in Training and Evaluating Recommender Systems." In: *ACM SIGKDD Explorations Newsletter* 19.1, 37–45.

Chen, James Ming (2018). "On Exactitude in Financial Regulation: Value-at-Risk, Expected Shortfall, and Expectiles." In: *Risks* 6.2. ISSN: 2227-9091. DOI: 10.3390/risks6020061. URL: https://www.mdpi.com/2227-9091/6/2/61.

Chen, Jia, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma (2019a). "TianGong-ST: A New Dataset with Large-Scale Refined Real-World Web Search Sessions." In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Beijing, China: ACM, 2485–2488. ISBN: 9781450369763. DOI: 10.1145/3357384.3358158.

Chen, Jia, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma (2020). "A Context-Aware Click Model for Web Search." In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. WSDM '20. Houston, TX, USA: ACM, 88–96. ISBN: 9781450368223. DOI: 10.1145/3336191.3371819.

Chen, Minmin, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi (2019b). "Top-K Off-Policy Correction for a REINFORCE Recommender System." In: *WSDM*, pp. 456–464. DOI: https://doi.org/10.1145/3289600.3290999.

Chen, Xinshi, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song (2019c). "Generative Adversarial User Model for Reinforcement Learning Based Recommendation System." In: *ICML '19*, pp. 1052–1061.

Chen, Ye and Tak W Yan (2012). "Position-normalized click prediction in search advertising." In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, pp. 795–803.

Cheng, Heng-Tze, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah (2016). "Wide & Deep Learning for Recommender Systems." In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. DLRS 2016. Boston, MA, USA: Association for Computing Machinery, 7–10. ISBN: 9781450347952. DOI: 10.1145/2988450.2988454. URL: https://doi.org/10.1145/2988450.2988454.

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." In: *EMNLP '14*, pp. 1724–1734.

Chuklin, Aleksandr, Ilya Markov, and Maarten de Rijke (2015). *Click Models for Web Search*. Morgan & Claypool. ISBN: 9781627056489. DOI: 10.2200/S00654ED1V01Y201507ICR043.

Chuklin, Aleksandr, Pavel Serdyukov, and Maarten de Rijke (2013). "Click Model-Based Information Retrieval Metrics." In: *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '13. Dublin, Ireland: Association for Computing Machinery, 493–502. ISBN: 9781450320344. DOI: 10.1145/2484028.2484071. URL: https://doi.org/10.1145/2484028.2484071.

Cinus, Federico, Marco Minici, Corrado Monti, and Francesco Bonchi (2022). "The Effect of People Recommenders on Echo Chambers and Polarization." In: *Proceedings of the International AAAI Conference on Web and Social Media* 16.1, pp. 90–101. DOI: 10.1609/icwsm.v16i1.19275. URL: https://ojs.aaai.org/index.php/ICWSM/article/view/19275.

Craswell, Nick, 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*. WSDM '08. Palo Alto, California, USA: Association for Computing Machinery, 87–94. ISBN: 9781595939272. DOI: 10.1145/1341531.1341545. URL: https://doi.org/10.1145/1341531.1341545.

Cremonesi, Paolo and Dietmar Jannach (2021). "Progress in Recommender Systems Research: Crisis? What Crisis?" In: *AI Magazine* 42.3, pp. 43–54.

Dabney, Will, Georg Ostrovski, David Silver, and Rémi Munos (2018a). "Implicit quantile networks for distributional reinforcement learning." In: *ICML*. PMLR, pp. 1096–1105.

Dabney, Will, Mark Rowland, Marc Bellemare, and Rémi Munos (2018b). "Distributional reinforcement learning with quantile regression." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32.

Dai, Xinyi, Jianghao Lin, Weinan Zhang, Shuai Li, Weiwen Liu, Ruiming Tang, Xiuqiang He, Jianye Hao, Jun Wang, and Yong Yu (2021). "An Adversarial Imitation Click Model for Infor-

mation Retrieval." In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: ACM, 1809–1820. ISBN: 9781450383127. DOI: 10.1145/3442381.3449913.

Dang, Van, Michael Bendersky, and W. Bruce Croft (2013). "Two-Stage Learning to Rank for Information Retrieval." In: *ECIR '13*, pp. 423–434.

Dawid, Alexander Philip (1979). "Conditional Independence in Statistical Theory." In: *Journal of the Royal Statistical Society: Series B (Methodological)* 41.1, pp. 1–15. DOI: https://doi.org/10.1111/j.2517-6161.1979.tb01052.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1979.tb01052.x.

Deffayet, Romain, Philipp Hager, Jean-Michel Renders, and Maarten de Rijke (2023a). "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*. SIGIR '23. Taipei, Taiwan: Association for Computing Machinery, 558–568. ISBN: 9781450394086. DOI: 10.1145/3539618.3591639. URL: https://doi.org/10.1145/3539618.3591639.

Deffayet, Romain, Jean-Michel Renders, and Maarten de Rijke (2023b). "Evaluating the Robustness of Click Models to Policy Distributional Shift." In: *ACM Trans. Inf. Syst.* 41.4. ISSN: 1046-8188. DOI: 10.1145/3569086. URL: https://doi.org/10.1145/3569086.

Deffayet, Romain, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke (2022). "Offline Evaluation for Reinforcement Learning-Based Recommendation: A Critical Issue and Some Alternatives." In: *ACM SIGIR Forum* 56.2, 3:1–3:14. DOI: https://doi.org/10.1145/3582900.3582905.

Deffayet, Romain, Thibaut Thonet, Jean-Michel Renders, and Maarten de Rijke (2023c). "Generative Slate Recommendation with Reinforcement Learning." In: *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*. WSDM '23. Singapore, Singapore: Association for Computing Machinery, 580–588. ISBN: 9781450394079. DOI: 10.1145/3539597.3570412. URL: https://doi.org/10.1145/3539597.3570412.

Doran, Gary, Krikamol Muandet, Kun Zhang, and Bernhard Schölkopf (2014). "A Permutation-Based Kernel Conditional Independence Test." In: *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*. UAI'14. Quebec City, Quebec, Canada: AUAI Press, 132–141. ISBN: 9780974903910.

Dulac-Arnold, Gabriel, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin (2015). "Deep Reinforcement Learning in Large Discrete Action Spaces." In: *arXiv:1512.07679*.

Dupret, Georges E. and Benjamin Piwowarski (2008). "A User Browsing Model to Predict Search Engine Click Data from Past Observations." In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '08. Singapore, Singapore: ACM, 331–338. ISBN: 9781605581644. DOI: 10.1145/1390334.1390392.

European Commission (2024). *Supervision of the designated very large online platforms and search engines under DSA*. [Online; accessed 23-March-2024]. URL: https://web.archive.org/web/20240329135127/https://digital-strategy.ec.europa.eu/en/policies/list-designated-vlops-and-vloses.

Ferrari Dacrema, Maurizio, Paolo Cremonesi, and Dietmar Jannach (2019). "Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches." In: *RecSys*, 101–109.

Flaxman, Seth R., Sharad Goel, and Justin M. Rao (2016). "Filter Bubbles, Echo Chambers, and Online News Consumption." In: *Public Opinion Quarterly* 80.S1, pp. 298–320.

Fu, Justin, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine (2020). "D4RL: Datasets for Deep Data-Driven Reinforcement Learning." In: *arXiv:2004.07219*.

Fu, Justin, Aviral Kumar, Matthew Soh, and Sergey Levine (2019). "Diagnosing Bottlenecks in Deep Q-learning Algorithms." In: *ICML*, pp. 2021–2030.

Fu, Justin, Mohammad Norouzi, Ofir Nachum, George Tucker, Ziyu Wang, Alexander Novikov, Mengjiao Yang, Michael R. Zhang, Yutian Chen, Aviral Kumar, Cosmin Paduraru, Sergey Levine, and Tom Le Paine (2021). "Benchmarks for Deep Off-Policy Evaluation." In: *ICLR*.

Fujimoto, Scott, David Meger, and Doina Precup (2019). "Off-Policy Deep Reinforcement Learning without Exploration." In: *ICML*.

Fukumizu, Kenji, Francis R. Bach, and Michael I. Jordan (2004). "Dimensionality Reduction for Supervised Learning with Reproducing Kernel Hilbert Spaces." In: *J. Mach. Learn. Res.* 5, 73–99. ISSN: 1532-4435. URL: https://dl.acm.org/doi/10.5555/1005332.1005335.

Gao, Chongming, 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*. SIGIR '23. Taipei, Taiwan: Association for Computing Machinery, 238–248. ISBN: 9781450394086. DOI: 10.1145/3539618.3591636. URL: https://doi.org/10.1145/3539618.3591636.

Garcin, Florent, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber (2014). "Offline and Online Evaluation of News Recommender Systems at Swissinfo.Ch." In: *RecSys*, 169–176.

Ghosh, Dibya, Anurag Ajay, Pulkit Agrawal, and Sergey Levine (2022). "Offline RL Policies Should Be Trained to be Adaptive." In: *ICML*, pp. 7513–7530.

Gomez-Uribe, Carlos A. and Neil Hunt (2016). "The Netflix Recommender System: Algorithms, Business Value, and Innovation." In: *ACM Trans. Manage. Inf. Syst.* 6.4.

Grotov, Artem, Aleksandr Chuklin, Ilya Markov, Luka Stout, Finde Xumara, and Maarten de Rijke (2015). "A Comparative Study of Click Models for Web Search." In: *CLEF 2015*. Cham, Switzerland: Springer, pp. 78–90.

GroupLens (n.d.). *MovieLens datasets*. URL: https://grouplens.org/datasets/movielens/.

Gu, Pengjie, Mengchen Zhao, Chen Chen, Dong Li, Jianye Hao, and Bo An (2022). "Learning Pseudometric-based Action Representations for Offline Reinforcement Learning." In: *ICML*, pp. 7902–7918.

Gulcehre, Caglar, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zołna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, Jerry Li, Mohammad Norouzi, Matt Hoffman, Nicolas Heess, and Nando de Freitas (2020). "RL Unplugged: A Suite of Benchmarks for Offline Reinforcement Learning." In: *NeurIPS*.

Guo, Huifeng, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He (2017). "DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction." In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. Melbourne, Australia: AAAI Press, 1725–1731. ISBN: 9780999241103.

Gupta, Shashank, 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*. SIGIR '23. Taipei, Taiwan: Association for Computing Machinery, 3440–3443. ISBN: 9781450394086. DOI: 10.1145/3539618.3594247. URL: https://doi.org/10.1145/3539618.3594247.

Ha, David and Jürgen Schmidhuber (2018). "Recurrent World Models Facilitate Policy Evolution." In: *NeurIPS '18*, pp. 2455–2467.

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

Hanna, Josiah, Scott Niekum, and Peter Stone (2019). "Importance Sampling Policy Evaluation with an Estimated Behavior Policy." In: *ICML*, pp. 2605–2613.

Hansen, Christian, Rishabh Mehrotra, Casper Hansen, Brian Brost, Lucas Maystre, and Mounia Lalmas (2021). "Shifting Consumption towards Diverse Content on Music Streaming Platforms." In: *WSDM '21*, 238–246.

Hasselt, Hado van, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil (2018). "Deep Reinforcement Learning and the Deadly Triad." In: *arXiv:1812.02648*.

He, Chen, Denis Parra, and Katrien Verbert (2016). "Interactive Recommender Systems: A Survey of the State of the Art and Future Research Challenges and Opportunities." In: *Expert Systems with Applications* 56, pp. 9–27. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.02.013. URL: https://www.sciencedirect.com/science/article/pii/S0957417416300367.

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

Hohnhold, Henning, Deirdre O'Brien, and Diane Tang (2015). "Focusing on the Long-Term: It's Good for Users and Business." In: *KDD '15*, 1849–1858.

Holzmann, Hajo and Bernhard Klar (2016). "Expectile asymptotics." In: *Electronic Journal of Statistics* 10.2, pp. 2355 –2371.

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

Huang, Jin, 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: *RecSys*, 190–199.

Huang, Shengyi, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo (2022b). "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms." In: *JMLR* 23.274, pp. 1–18.

Huleihel, Wasim, 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*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 1432–1440. URL: https://proceedings.mlr.press/v130/huleihel21a.html.

Hussein, Ahmed, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne (2017). "Imitation Learning: A Survey of Learning Methods." In: *ACM Comput. Surv.* 50.2.

Ie, Eugene, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier (2019a). *RecSim: A Configurable Simulation Platform for Recommender Systems*. arXiv: 1909.04847 [cs.LG].

Ie, Eugene, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier (July 2019b). "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, pp. 2592–2599. DOI: 10.24963/ijcai.2019/360. URL: https://doi.org/10.24963/ijcai.2019/360.

Instacart (2017). *Instacart market basket analysis*. URL: https://www.kaggle.com/c/instacart-market-basket-analysis/data.

Jagerman, Rolf, Ilya Markov, and Maarten de Rijke (2019). "When People Change their Mind: Off-policy Evaluation in Non-stationary Recommendation Environments." In: *WSDM 2019: 12th International Conference on Web Search and Data Mining*. ACM, pp. 447–455.

Jannach, Dietmar, Pearl Pu, Francesco Ricci, and Markus Zanker (2021). "Recommender Systems: Past, Present, Future." In: *AI Mag.* 42.3, pp. 3–6.

Jannach, Dietmar, Paul Resnick, Alexander Tuzhilin, and Markus Zanker (2016). "Recommender Systems — beyond Matrix Completion." In: *Commun. ACM* 59.11, 94–102.

Janner, Michael, Qiyang Li, and Sergey Levine (2021). *Reinforcement Learning as One Big Sequence Modeling Problem*. arXiv: 2106.02039 [cs.LG].

Järvelin, Kalervo and Jaana Kekäläinen (Oct. 2002). "Cumulated Gain-Based Evaluation of IR Techniques." In: *ACM Trans. Inf. Syst.* 20.4, 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418.

Jeunen, Olivier (2019). "Revisiting Offline Evaluation for Implicit-Feedback Recommender Systems." In: *RecSys*, 596–600.

Jeunen, Olivier (2023). "A Common Misassumption in Online Experiments with Machine Learning Models." In: *PERSPECTIVES workshop at RecSys'23*. URL: https://arxiv.org/abs/2304.10900.

Jeunen, Olivier and Bart Goethals (2021). "Pessimistic Reward Models for Off-Policy Learning in Recommendation." In: *Fifteenth ACM Conference on Recommender Systems*. RecSys '21. Amsterdam, Netherlands: Association for Computing Machinery, 63–74. ISBN: 9781450384582. DOI: 10.1145/3460231.3474247. URL: https://doi.org/10.1145/3460231.3474247.

Ji, Luo, Qi Qin, Bingqing Han, and Hongxia Yang (2021). "Reinforcement Learning to Optimize Lifetime Value in Cold-Start Recommendation." In: *CIKM*, 782–791.

Ji, Yitong, Aixin Sun, Jie Zhang, and Chenliang Li (2020). "A Critical Study on Data Leakage in Recommender System Offline Evaluation." In: *arXiv:2010.11060*.

Jiang, Ray, Sven Gowal, Yuqiu Qian, Timothy A. Mann, and Danilo J. Rezende (2019). "Beyond Greedy Ranking: Slate Optimization via List-CVAE." In: *ICLR '19*.

Joachims, Thorsten, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay (2005). "Accurately Interpreting Clickthrough Data as Implicit Feedback." In: *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Re-*

trieval. SIGIR '05. Salvador, Brazil: ACM, 154–161. ISBN: 1595930345. DOI: 10.1145/1076034.1076063.

Joachims, Thorsten, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay (Apr. 2007). "Evaluating the Accuracy of Implicit Feedback from Clicks and Query Reformulations in Web Search." In: *ACM Trans. Inf. Syst.* 25.2, 7–es. ISSN: 1046-8188. DOI: 10.1145/1229179.1229181.

Joachims, Thorsten, Adith Swaminathan, and Tobias Schnabel (2017). "Unbiased Learning-to-Rank with Biased Feedback." In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM '17. Cambridge, United Kingdom: Association for Computing Machinery, 781–789. ISBN: 9781450346757. DOI: 10.1145/3018661.3018699. URL: https://doi.org/10.1145/3018661.3018699.

Jullien, Sami, Mozhdeh Ariannezhad, Paul Groth, and Maarten de Rijke (2023). "A Simulation Environment and Reinforcement Learning Method for Waste Reduction." In: *TMLR*.

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). "Planning and Acting in Partially Observable Stochastic Domains." In: *Artificial Intelligence* 101.1, pp. 99–134.

Kingma, Diederik and Max Welling (Dec. 2014). "Auto-Encoding Variational Bayes." In: *ICLR '14*.

Kiyohara, Haruka and Kosuke Kawakami (2022). "OFRL: Designing an Offline Reinforcement Learning and Policy Evaluation Platform from Practical Perspectives." In: *CONSEQUENCES+REVEAL@RecSys*.

Kiyohara, Haruka, 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." In: *arXiv preprint arXiv:23xx.xxxxx*. URL: https://github.com/hakuhodo-technologies/scope-rl.

Knyazev, Norman and Harrie Oosterhuis (2023). "A Lightweight Method for Modeling Confidence in Recommendations with Learned Beta Distributions." In: *RecSys'23*. New York, NY, USA: Association for Computing Machinery. URL: https://arxiv.org/abs/2308.03186.

Koller, Daphne and Nir Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press. URL: https://mitpress.mit.edu/9780262013192/probabilistic-graphical-models/.

Koller, Daphne and Mehran Sahami (1996). "Toward Optimal Feature Selection." In: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. ICML'96. Bari, Italy: Morgan Kaufmann Publishers Inc., 284–292. ISBN: 1558604197. URL: https://dl.acm.org/doi/10.5555/3091696.3091731.

Koren, Yehuda, Robert M. Bell, and Chris Volinsky (2009). "Matrix Factorization Techniques for Recommender Systems." In: *Computer* 42.8, pp. 30–37.

Kostrikov, Ilya, Ashvin Nair, and Sergey Levine (2022). "Offline Reinforcement Learning with Implicit Q-Learning." In: *ICLR*.

Krichene, Walid and Steffen Rendle (2020). "On Sampled Metrics for Item Recommendation." In: *KDD*, 1748–1757.

Kullback, Solomon and Richard A. Leibler (1951). "On Information and Sufficiency." In: *The Annals of Mathematical Statistics* 22.1, pp. 79–86.

Kumar, Aviral, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine (2021). "A Workflow for Offline Model-Free Robotic Reinforcement Learning." In: *CoRL*.

Kveton, Branislav, Csaba Szepesvári, Zheng Wen, and Azin Ashkan (2015). "Cascading Bandits: Learning to Rank in the Cascade Model." In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 767–776.

Last.fm (n.d.). URL: https://last.fm/api.

Lee, Hojoon, Dongyoon Hwang, Kyushik Min, and Jaegul Choo (2022a). "Towards Validating Long-Term User Feedbacks in Interactive Recommendation Systems." In: *SIGIR*, pp. 2607–2611. DOI: 10.48550/arXiv.2308.11137.

Lee, Taeyoon, Donghyun Sung, Kyoungyeon Choi, Choongin Lee, Changwoo Park, and Keunjun Choi (2022b). *Learning Dynamic Manipulation Skills from Haptic-Play*. arXiv: 2207.14007 [cs.RO].

Levine, Sergey, Aviral Kumar, George Tucker, and Justin Fu (2020). *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. arXiv: 2005.01643 [cs.LG].

Li, Lihong, 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*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 661–670. ISBN: 9781605587998. DOI: 10.1145/1772690.1772758. URL: https://doi.org/10.1145/1772690.1772758.

Li, Shuai, Yasin Abbasi-Yadkori, Branislav Kveton, S. Muthukrishnan, Vishwa Vinay, and Zheng Wen (2018). "Offline Evaluation of Ranking Policies with Click Models." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '18. London, United Kingdom: ACM, 1685–1694. ISBN: 9781450355520. DOI: 10.1145/3219819.3220028.

Lillicrap, Timothy P., 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*.

Lin, Jianghao, Weiwen Liu, Xinyi Dai, Weinan Zhang, Shuai Li, Ruiming Tang, Xiuqiang He, Jianye Hao, and Yong Yu (2021). "A Graph-Enhanced Click Model for Web Search." In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 1259–1268. ISBN: 9781450380379. DOI: 10.1145/3404835.3462895. URL: https://doi.org/10.1145/3404835.3462895.

Liu, Shuchang, 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*, pp. 833–844. DOI: https://doi.org/10.1145/3543507.3583244.

Liu, Shuchang, Fei Sun, Yingqiang Ge, Changhua Pei, and Yongfeng Zhang (2021). "Variation Control and Evaluation for Generative Slate Recommendations." In: *WWW '21*, 436–448.

Liu, Tie-Yan (2009). "Learning to Rank for Information Retrieval." In: *Found. Trends Inf. Retr.* 3.3, 225–331. ISSN: 1554-0669. DOI: 10.1561/1500000016. URL: https://doi.org/10.1561/1500000016.

Liu, Yiqun, Xiaohui Xie, Chao Wang, Jian-Yun Nie, Min Zhang, and Shaoping Ma (2016). "Time-Aware Click Model." In: *ACM Trans. Inf. Syst.* 35.3. ISSN: 1046-8188. DOI: 10.1145/2988230. URL: https://doi.org/10.1145/2988230.

Lopez-Paz, David and Maxime Oquab (2017). "Revisiting Classifier Two-Sample Tests." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SJkXfE5xx.

Lowerre, Bruce (1976). "The HARPY speech recognition system." In: *The Journal of the Acoustical Society of America* 60.S1.

Luce, Duncan (1959). *Individual Choice Behavior: A Theoretical Analysis*. Courier Corporation.

Ludewig, Malte, Noemi Mauro, Sara Latifi, and Dietmar Jannach (2019). "Performance Comparison of Neural and Non-Neural Approaches to Session-Based Recommendation." In: *RecSys*, 462–466.

Luzin, Nikolai (1915). "The Integral and Trigonometric Series (Russian)." PhD thesis. Moscow State University.

Machado, Marlos C., Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling (2018). "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents." In: *Journal of Artificial Intelligence Research* 61, pp. 523–562.

Majumdar, Angshul 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)*, pp. 3656–3663. DOI: 10.1109/IJCNN.2017.7966316.

Margaritis, Dimitris (2005). "Distribution-Free Learning of Bayesian Network Structure in Continuous Domains." In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*. AAAI'05. Pittsburgh, Pennsylvania: AAAI Press, 825–830. ISBN: 157735236X. URL: https://dl.acm.org/doi/10.5555/1619410.1619465.

Martin, John, Michal Lyskawinski, Xiaohu Li, and Brendan Englot (2020). "Stochastically Dominant Distributional Reinforcement Learning." In: *ICML*. Vol. 119. PMLR, pp. 6745–6754.

Masrour, Farzan, Tyler Wilson, Heng Yan, Pang-Ning Tan, and Abdol-Hossein Esfahanian (2020). "Bursting the Filter Bubble: Fairness-Aware Network Link Prediction." In: *AAAI '20*, pp. 841–848.

Matsushima, Tatsuya, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu (2020). *Deployment-Efficient Reinforcement Learning via Model-Based Offline Optimization*. arXiv: 2006.03647 [cs.LG].

Mavrin, Borislav, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu (2019). "Distributional reinforcement learning for efficient exploration." In: *ICML*. PMLR, pp. 4424–4434.

McInerney, James, Brian Brost, Praveen Chandar, Rishabh Mehrotra, and Benjamin Carterette (2020). "Counterfactual Evaluation of Slate Recommendations with Sequential Reward Interactions." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 1779–1788. ISBN: 9781450379984.

McMahan, H Brendan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. (2013). "Ad Click Prediction: A View from the Trenches." In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, pp. 1222–1230.

McNee, Sean M., John Riedl, and Joseph A. Konstan (2006a). "Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems." In: *CHI*, 1097–1101.

McNee, Sean M., John Riedl, and Joseph A. Konstan (2006b). "Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems." In: *CHI '06 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '06. Montréal, Québec, Canada: Association for Computing Machinery, 1097–1101. ISBN: 1595932984. DOI: 10.1145/1125451.1125659. URL: https://doi.org/10.1145/1125451.1125659.

Mehrotra, Rishabh (2021). "Algorithmic Balancing of Familiarity, Similarity, & Discovery in Music Recommendations." In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 3996–4005. ISBN: 9781450384469. DOI: 10.1145/3459637.3481893. URL: https://doi.org/10.1145/3459637.3481893.

Melville, Prem and Vikas Sindhwani (2010). "Recommender Systems." In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, pp. 829–838. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_705. URL: https://doi.org/10.1007/978-0-387-30164-8_705.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning." In: *arXiv preprint arXiv:1312.5602*.

Mukherjee, Sudipto, Himanshu Asnani, and Sreeram Kannan (2019). "CCMI : Classifier based Conditional Mutual Information Estimation." In: *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, (UAI)*. Vol. 115. Proceedings of Machine Learning Research. AUAI Press, pp. 1083–1093. URL: http://proceedings.mlr.press/v115/mukherjee20a.html.

Newey, Whitney K. and James L. Powell (1987). "Asymmetric Least Squares Estimation and Testing." In: *Econometrica* 55.4, pp. 819–847.

Oosterhuis, Harrie (2021a). "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*. New York, NY, USA: Association for Computing Machinery, 1023–1032. ISBN: 9781450380379. URL: https://doi.org/10.1145/3404835.3462830.

Oosterhuis, Harrie (2021b). "Learning from user interactions with rankings: a unification of the field." In: *SIGIR Forum* 54.2. ISSN: 0163-5840. DOI: 10.1145/3483382.3483402. URL: https://doi.org/10.1145/3483382.3483402.

Oosterhuis, Harrie (2022). "Reaching the End of Unbiasedness: Uncovering Implicit Limitations of Click-Based Learning to Rank." In: *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval*. ICTIR '22. Madrid, Spain: Association for Computing Machinery, 264–274. ISBN: 9781450394123. DOI: 10.1145/3539813.3545137. URL: https://doi.org/10.1145/3539813.3545137.

Oosterhuis, Harrie and Maarten de Rijke (2020a). "Policy-Aware Unbiased Learning to Rank for Top-k Rankings." In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 489–498. ISBN: 9781450380164. URL: https://doi.org/10.1145/3397271.3401102.

Oosterhuis, Harrie and Maarten de Rijke (2020b). "Taking the Counterfactual Online: Efficient and Unbiased Online Evaluation for Ranking." In: *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval*. ICTIR '20. Virtual Event, Norway: Association for Computing Machinery, 137–144. ISBN: 9781450380676. DOI: 10.1145/3409256.3409820. URL: https://doi.org/10.1145/3409256.3409820.

Oosterhuis, Harrie and Maarten de Rijke (2021a). "Robust Generalization and Safe Query-Specialization in Counterfactual Learning to Rank." In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 158–170. ISBN: 9781450383127. DOI: 10.1145/3442381.3450018. URL: https://doi.org/10.1145/3442381.3450018.

Oosterhuis, Harrie and Maarten de Rijke (2021b). "Unifying Online and Counterfactual Learning to Rank: A Novel Counterfactual Estimator That Effectively Utilizes Online Interventions." In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: ACM, 463–471. ISBN: 9781450382977. DOI: 10.1145/3437963.3441794.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba (2020). "Learning Dexterous In-Hand Manipulation." In: *The International Journal of Robotics Research* 39.1, pp. 3–20.

Ovaisi, Zohreh, Ragib Ahsan, Yifan Zhang, Kathryn Vasilaky, and Elena Zheleva (2020). "Correcting for Selection Bias in Learning-to-Rank Systems." In: *Proceedings of The Web Conference 2020*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 1863–1873. ISBN: 9781450370233. DOI: 10.1145/3366423.3380255. URL: https://doi.org/10.1145/3366423.3380255.

Padakandla, Sindhu, Prabuchandran K. J., and Shalabh Bhatnagar (2020). "Reinforcement Learning Algorithm for Non-stationary Environments." In: *Applied Intelligence* 50.11, pp. 3590–3606. DOI: 10.1007/s10489-020-01758-5. URL: https://doi.org/10.1007%2Fs10489-020-01758-5.

Pariser, Eli (2011). *The Filter Bubble: What the Internet Is Hiding from You*. The Penguin Press.

Pearl, Judea (2009). *Causality*. Cambridge university press. DOI: 10.1017/CBO9780511803161.

Perdomo, Juan, Tijana Zrnic, Celestine Mendler-Dünner, and Moritz Hardt (2020). "Performative Prediction." In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 7599–7609. URL: https://proceedings.mlr.press/v119/perdomo20a.html.

Philipps, Colin (2021a). "When is an Expectile the Best Linear Unbiased Estimator?" In: *SSRN*.

Philipps, Collin (2021b). "Interpreting Expectiles." In: *SSRN*.

Plackett, Robin (1975). "The Analysis of Permutations." In: *Journal of the Royal Statistical Society* 24.2. DOI: 10.2307/2346567.

Polyak, Boris T. and Anatoli B. Juditsky (1992). "Acceleration of Stochastic Approximation by Averaging." In: *SIAM Journal on Control and Optimization* 30.4, pp. 838–855.

Precup, Doina, Richard S. Sutton, and Satinder P. Singh (2000). "Eligibility Traces for Off-Policy Policy Evaluation." In: *Proceedings of the Seventeenth International Conference on Machine*

Learning. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 759–766. ISBN: 1558607072.

Qin, Rongjun, Songyi Gao, Xingyuan Zhang, Zhen Xu, Shengkai Huang, Zewen Li, Weinan Zhang, and Yang Yu (2021). "NeoRL: A Near Real-World Benchmark for Offline Reinforcement Learning." In: *arXiv:2102.00714*.

Qin, Tao and Tie-Yan Liu (2013). *Introducing LETOR 4.0 Datasets*. abs/1306.2597.

Quadrana, Massimo, Paolo Cremonesi, and Dietmar Jannach (2018). "Sequence-Aware Recommender Systems." In: *ACM Comput. Surv.* 51.4.

Rahdari, Behnam, Branislav Kveton, and Peter Brusilovsky (2022). *From Ranked Lists to Carousels: A Carousel Click Model*. URL: https://arxiv.org/abs/2209.13426.

Reichlin, Alfredo, Giovanni Luca Marchetti, Hang Yin, Ali Ghadirzadeh, and Danica Kragic (2022). "Back to the Manifold: Recovering from Out-of-Distribution States." In: *arXiv:2207.08673*.

Rendle, Steffen, Li Zhang, and Yehuda Koren (2019). "On the Difficulty of Evaluating Baselines: A Study on Recommender Systems." In: *arXiv:1905.01395*.

RetailRocket (2016). *RetailRocket recommender system dataset*. URL: https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset.

Robertson, Stephen E. (1977). "The Probability Ranking Principle in IR." In: *Journal of Documentation* 33.4, pp. 294–304.

Robertson, Stephen E., Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford (1994). "Okapi at TREC-3." In: *Proceedings of The Third Text REtrieval Conference, TREC*. Vol. 500-225. NIST Special Publication. Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST), pp. 109–126.

Rohde, David, 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].

Rossi, Wilbert Samuel, Jan Willem Polderman, and Paolo Frasca (2021). "The Closed Loop between Opinion Formation and Personalised Recommendations." In: *IEEE Transactions on Control of Network Systems*.

Rowland, Mark, Robert Dadashi, Saurabh Kumar, Rémi Munos, Marc G. Bellemare, and Will Dabney (2019). "Statistics and samples in distributional reinforcement learning." In: *ICML*. PMLR, pp. 5528–5536.

Rowland, Mark, Rémi Munos, Mohammad Gheshlaghi Azar, Yunhao Tang, Georg Ostrovski, Anna Harutyunyan, Karl Tuyls, Marc G. Bellemare, and Will Dabney (2023). "An Analysis of Quantile Temporal-Difference Learning." In: *arXiv preprint arXiv:2301.04462*.

Rubens, Neil, Mehdi Elahi, Masashi Sugiyama, and Dain Kaplan (2015). "Active Learning in Recommender Systems." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. Boston, MA: Springer US, pp. 809–846. ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6_24. URL: https://doi.org/10.1007/978-1-4899-7637-6_24.

Saito, Yuta, 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*. Ed. by J. Vanschoren and S. Yeung. Vol. 1. Curran. URL: https://datasets-benchmarks-proceedings.

neurips.cc/paper_files/paper/2021/file/33e75ff09dd601bbe69f351039152189-Paper-round2.pdf.

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

Santana, Marlesson R. O., 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)*, pp. 189–197. DOI: 10.1109/ICDMW51313.2020.00035.

Sarvi, Fatemeh, Ali Vardasbi, Mohammad Aliannejadi, Sebastian Schelter, and Maarten de Rijke (2023). "On the Impact of Outlier Bias on User Clicks." In: SIGIR '23. New York, NY, USA: Association for Computing Machinery, 18–27. ISBN: 9781450394086. DOI: 10.1145/3539618.3591745. URL: https://doi.org/10.1145/3539618.3591745.

Sato, Masahiro (2021). "Online Evaluation Methods for the Causal Effect of Recommendations." In: *Proceedings of the 15th ACM Conference on Recommender Systems*. RecSys '21. Amsterdam, Netherlands: Association for Computing Machinery, 96–101. ISBN: 9781450384582. DOI: 10.1145/3460231.3474235. URL: https://doi.org/10.1145/3460231.3474235.

Sen, Rajat, Ananda Theertha Suresh, Karthikeyan Shanmugam, Alexandres G. Dimakis, and Sanjay Shakkettai (2017). "Model-Powered Conditional Independence Test." In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2955–2965. ISBN: 9781510860964. URL: https://dl.acm.org/doi/10.5555/3294996.3295055.

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

Serdyukov, Pavel, Nick Craswell, and Georges Dupret (2012). "WSCD 2012: Workshop on Web Search Click Data 2012." In: *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*. WSDM '12. Seattle, Washington, USA: ACM, 771–772. ISBN: 9781450307475. DOI: 10.1145/2124295.2124396.

Shen, Zheyan, Jiashuo Liu, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui (2021). *Towards Out-Of-Distribution Generalization: A Survey*. DOI: 10.48550/ARXIV.2108.13624. URL: https://arxiv.org/abs/2108.13624.

Shi, Jing-Cheng, Yang Yu, Qing Da, Shi-Yong Chen, and An-Xiang Zeng (2019). "Virtual-Taobao: Virtualizing Real-World Online Retail Environment for Reinforcement Learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01, pp. 4902–4909. DOI: 10.1609/aaai.v33i01.33014902. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4419.

Silva, Nícollas, 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." In: *Expert Systems with Applications* 197, p. 116669. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.116669. URL: https://www.sciencedirect.com/science/article/pii/S0957417422001543.

Silva, Álvaro Labarca, Denis Parra, and Rodrigo Toro Icarte (2024). "On the Unexpected Effectiveness of Reinforcement Learning for Sequential Recommendation." In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by Ruslan Salakhutdinov, Zico Kolter,

Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp. Vol. 235. Proceedings of Machine Learning Research. PMLR, pp. 45432–45450. URL: https://proceedings.mlr.press/v235/silva24b.html.

Singh, Ashudeep and Thorsten Joachims (2018). "Fairness of Exposure in Rankings." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '18. London, United Kingdom: ACM, 2219–2228. ISBN: 9781450355520. DOI: 10.1145/3219819.3220088.

Slivkins, Aleksandrs (2019). "Introduction to multi-armed bandits." In: *Foundations and Trends in Machine Learning* 12.1-2, pp. 1–286. DOI: 10.1561/2200000068.

Smith, James and Robert Winkler (2006). "The Optimizer's Curse: Skepticism and Postdecision Surprise in Decision Analysis." In: *Management Science* 52.

Steck, Harald (2010). "Training and Testing of Recommender Systems on Data Missing Not at Random." In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, USA: ACM.

Stooke, Adam, Kimin Lee, Pieter Abbeel, and Michael Laskin (2021). "Decoupling Representation Learning from Reinforcement Learning." In: *ICML '21*, pp. 9870–9879.

Strens, Malcolm (2000). "A Bayesian framework for reinforcement learning." In: *ICML*. Vol. 2000, pp. 943–950.

Su, Liangjun and Halbert White (2007). "A consistent characteristic function-based test for conditional independence." In: *Journal of Econometrics* 141.2, pp. 807–834. DOI: 10.1016/j.jeconom.2006.11.006.

Sun, Aixin (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*. SIGIR '23. Taipei, Taiwan: Association for Computing Machinery, 2629–2638. ISBN: 9781450394086. DOI: 10.1145/3539618.3591931. URL: https://doi.org/10.1145/3539618.3591931.

Sun, Zhu, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng (2020). "Are We Evaluating Rigorously? Benchmarking Recommendation for Reproducible Evaluation and Fair Comparison." In: *RecSys*, 23–32.

Sunehag, Peter, Richard Evans, Gabriel Dulac-Arnold, Yori Zwols, Daniel Visentin, and Ben Coppin (2015). "Deep Reinforcement Learning with Attention for Slate Markov Decision Processes with High-Dimensional States and Actions." In: *arXiv:1512.01124*.

Sutton, Richard and Andrew Barto (2018a). *Reinforcement Learning : an Introduction*. MIT Press. ISBN: 9780262039246.

Sutton, Richard and Andrew Barto (2018b). "Reinforcement Learning: An Introduction." In: MIT Press, pp. 326–329.

Swaminathan, Adith and Thorsten Joachims (2015). "Batch Learning from Logged Bandit Feedback through Counterfactual Risk Minimization." In: *Journal of Machine Learning Research* 16.52, pp. 1731–1755.

Swaminathan, Adith, 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*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 3635–3645. ISBN: 9781510860964.

Tobin, Josh, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel (2017). "Domain randomization for transferring deep neural networks from simulation to the real world." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

Towers, Mark, 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 (Mar. 2023). *Gymnasium*. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

Vardasbi, Ali, Harrie Oosterhuis, and Maarten de Rijke (2020a). "When Inverse Propensity Scoring Does Not Work: Affine Corrections for Unbiased Learning to Rank." In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. New York, NY, USA: ACM, 1475–1484. ISBN: 9781450368599.

Vardasbi, Ali, Maarten de Rijke, and Ilya Markov (2020b). "Cascade Model-Based Propensity Estimation for Counterfactual Learning to Rank." In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 2089–2092. ISBN: 9781450380164.

Vlassis, Nikos, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart (2012). "Bayesian reinforcement learning." In: *Reinforcement Learning: State-of-the-Art*, pp. 359–386.

Waller, Isaac and Ashton Anderson (2019). "Generalists and Specialists: Using Community Embeddings to Quantify Activity Diversity in Online Platforms." In: *WWW '19*, 1954–1964.

Waltrup, Linda Schulze, Fabian Sobotka, Thomas Kneib, and Göran Kauermann (2015). "Expectile and quantile regression—David and Goliath?" In: *Statistical Modelling* 15.5, pp. 433–456.

Wang, Angelina, Sayash Kapoor, Solon Barocas, and Arvind Narayanan (2024). "Against Predictive Optimization: On the Legitimacy of Decision-making Algorithms That Optimize Predictive Accuracy." In: *ACM J. Responsib. Comput.* 1.1. DOI: 10.1145/3636509. URL: https://doi.org/10.1145/3636509.

Wang, Kai, Zhene Zou, Minghao Zhao, Qilin Deng, Yue Shang, Yile Liang, Runze Wu, Xudong Shen, Tangjie Lyu, and Changjie Fan (2023a). "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*. SIGIR '23. Taipei, Taiwan: Association for Computing Machinery, 2935–2944. ISBN: 9781450394086. DOI: 10.1145/3539618.3591899. URL: https://doi.org/10.1145/3539618.3591899.

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

Wang, Shoujin, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Orgun (July 2019). "Sequential Recommender Systems: Challenges, Progress and Prospects." In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, pp. 6332–6338. DOI: 10.24963/ijcai.2019/883. URL: https://doi.org/10.24963/ijcai.2019/883.

Wang, Yu, 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*. WWW '22. Virtual Event, Lyon, France: Association for Computing

Machinery, 2015–2025. ISBN: 9781450390965. DOI: 10.1145/3485447.3512202. URL: https://doi.org/10.1145/3485447.3512202.

Watkins, Christopher and Peter Dayan (1992). "Q-learning." In: *Machine Learning* 8, pp. 279–292.

Xin, Xin, Tiago Pimentel, Alexandros Karatzoglou, Pengjie Ren, Konstantina Christakopoulou, and Zhaochun Ren (2022). "Rethinking Reinforcement Learning for Recommendation: A Prompt Perspective." In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22. Madrid, Spain: Association for Computing Machinery, 1347–1357. ISBN: 9781450387323. DOI: 10.1145/3477495.3531714. URL: https://doi.org/10.1145/3477495.3531714.

Yang, Derek, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tie-Yan Liu (2019). "Fully parameterized quantile function for distributional reinforcement learning." In: *NeurIPS* 32.

Yao, Qiwli and Howell Tong (1996). "Asymmetric least squares regression estimation: A nonparametric approach." In: *Journal of Nonparametric Statistics* 6.2-3, pp. 273–292.

Yu, Tianhe, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma (2020). "MOPO: Model-based Offline Policy Optimization." In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates.

Zangerle, Eva and Christine Bauer (2022). "Evaluating Recommender Systems: Survey and Framework." In: *ACM Comput. Surv.* 55.8. ISSN: 0360-0300. DOI: 10.1145/3556536. URL: https://doi.org/10.1145/3556536.

Zaretsky, Moisej A. (1925). "On one theorem on absolutely continuous functions (Russian)," in: *Doklady Rossiiskoi Akademii Nauk*.

Zhang, Junqi, Yiqun Liu, Jiaxin Mao, Weizhi Ma, Jiazheng Xu, Shaoping Ma, and Qi Tian (2022). "User Behavior Simulation for Search Result Re-Ranking." In: *ACM Trans. Inf. Syst.* Just Accepted. ISSN: 1046-8188. DOI: 10.1145/3511469. URL: https://doi.org/10.1145/3511469.

Zhao, Wayne Xin, Zihan Lin, Zhichao Feng, Pengfei Wang, and Ji-Rong Wen (2022). "A Revisiting Study of Appropriate Offline Evaluation for Top-N Recommendation Algorithms." In: *ACM Trans. Inf. Syst.*

Zheng, Yukun, Jiaxin Mao, Yiqun Liu, Cheng Luo, Min Zhang, and Shaoping Ma (2019). "Constructing Click Model for Mobile Search with Viewport Time." In: *ACM Trans. Inf. Syst.* 37.4. ISSN: 1046-8188. DOI: 10.1145/3360486. URL: https://doi.org/10.1145/3360486.

Zhou, Wenxuan, Sujay Bajracharya, and David Held (2020). "PLAS: Latent Action Space for Offline Reinforcement Learning." In: *CoRL '20*, pp. 1719–1735.

Zhu, Zeyuan Allen, Weizhu Chen, Tom Minka, Chenguang Zhu, and Zheng Chen (2010). "A Novel Click Model and Its Applications to Online Advertising." In: *Proceedings of the Third ACM International Conference on Web Search and Data Mining*. WSDM '10. New York, New York, USA: Association for Computing Machinery, 321–330. ISBN: 9781605588896. DOI: 10.1145/1718487.1718528. URL: https://doi.org/10.1145/1718487.1718528.

Zhuang, Honglei, Zhen Qin, Xuanhui Wang, Michael Bendersky, Xinyu Qian, Po Hu, and Dan Chary Chen (2021). "Cross-Positional Attention for Debiasing Clicks." In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 788–797. ISBN: 9781450383127. DOI: 10.1145/3442381.3450098. URL: https://doi.org/10.1145/3442381.3450098.

Zou, Lixin, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin (2019). "Reinforcement Learning to Optimize Long-Term User Engagement in Recommender Systems." In: *KDD '19*, pp. 2810–2818.

SUMMARY

Recommender systems can provide content that matches the interests of their users, but too often, their recommendations are redundant or overly biased towards a few categories of content. This is known to degrade the user experience, and affect both user-side and business-side metrics in the long run, and it is suspected to have detrimental consequences on certain societal concerns such as fairness and radicalization pathways. Part of the reasons why this happens is the fact that recommender systems do not account for the effect they have on people using them.

In this thesis, I investigate recommendation models and agents that are able to acknowledge the consequences of deploying them to a live recommendation platform, and can therefore correctly optimize the target metrics in the long-run. In particular, I distinguish two types of distribution shifts, i.e., of changes in user-system interactions due to the deployment of recommender systems: (i) algorithmic filtering in the selection and presentation of content that results in interaction logs that are biased towards certain items, and (ii) the effect of recommendations on a user's behavior and preferences (e.g., boredom, new interests, biased worldviews, ...) when multiple recommendations are made sequentially. Because of these effects of recommender systems on their users, recommendation is an inherently dynamic and interactive task.

The first part of the thesis is dedicated to the evaluation of dynamic and interactive recommender systems. I present limitations of the traditional way of evaluating sequential recommender systems and present some alternatives that are able to capture the dynamic and interactive nature of recommendation. In particular, my collaborators and I propose an extensive simulation environment that encompasses many challenges for future research on dynamic recommender systems. The second part of the thesis concerns the deployment of models trained on interaction logs that are biased by the selection presented to the user, which often comes from previous versions of the recommender system in production. While there exists models that can, under certain as-

sumptions, mitigate the biases present in the data, I found that current practices cannot ensure that the newly deployed model will be free of residual biases. We therefore propose a new metric that helps predict the robustness of the trained models to deployment. In the third part, I explore ways to make reinforcement learning agents, which are natural candidates for long-term optimization of dynamic recommender systems, adapted to the recommendation scenario. In particular, we propose an assumption-free agent that effectively and efficiently learns to balance accuracy and diversity of recommendations, in order to maximize long-term user engagement. Then, I investigate distributional reinforcement learning agents that are better able to capture the high uncertainty that can typically be found in recommendation scenarios and we propose an effective but theoretically sound update rule for distributional reinforcement learning.

SAMENVATTING

Recommender-systemen kunnen inhoud aanbieden die aansluit bij de interesses van hun gebruikers, maar al te vaak zijn hun aanbevelingen overbodig of te zeer gericht op een paar categorieën inhoud. Het is bekend dat dit de gebruikerservaring verslechtert en op de lange termijn zowel de gebruikerskant als de bedrijfskant beïnvloedt. Bovendien wordt vermoed dat dit nadelige gevolgen heeft voor bepaalde maatschappelijke problemen zoals on eerlijkheid en radicaliseringstrajecten. Een deel van de redenen waarom dit gebeurt, is het feit dat aanbevelingssystemen geen rekening houden met het effect dat ze hebben op mensen die ze gebruiken.

In dit proefschrift onderzoek ik aanbevelingsmodellen en agenten die in staat zijn om de gevolgen van hun inzet op een live aanbevelingsplatform te onderkennen, en daardoor de doelmetriek op de lange termijn correct kunnen optimaliseren. In het bijzonder onderscheid ik twee soorten distributieverschuivingen, d.w.z. veranderingen in de interacties tussen gebruiker en systeem als gevolg van de inzet van aanbevelingssystemen: (i) algoritmische filtering bij de selectie en presentatie van inhoud die resulteert in interactielogs die bevooroordeeld zijn ten opzichte van bepaalde items, en (ii) het effect van aanbevelingen op het gedrag en de voorkeuren van een gebruiker (bijv. verveling, nieuwe interesses, bevooroordeelde wereldbeelden, ...) wanneer meerdere aanbevelingen na elkaar worden gedaan. Vanwege deze effecten van aanbevelingssystemen op hun gebruikers, is aanbeveling een inherent dynamische en interactieve taak.

Het eerste deel van het proefschrift is gewijd aan de evaluatie van dynamische en interactieve aanbevelingssystemen. Ik presenteer de beperkingen van de traditionele manier om sequentiële aanbevelingssystemen te evalueren en presenteer enkele alternatieven die de dynamische en interactieve aard van aanbevelingen kunnen weergeven. In het bijzonder stellen mijn co-auteurs en ik een uitgebreide simulatieomgeving voor die veel uitdagingen bevat voor toekomstig onderzoek naar dynamische aanbevelingssystemen. Het tweede

deel van het proefschrift gaat over de inzet van modellen die getraind zijn op interactielogs die vertekend zijn door de selectie die aan de gebruiker is gepresenteerd, vaak afkomstig van eerdere versies van het recommender-systeem in productie. Hoewel er modellen bestaan die, onder bepaalde aannames, de vertekeningen in de gegevens kunnen verminderen, heb ik ontdekt dat de huidige praktijk niet kan garanderen dat het nieuw ingezette model vrij zal zijn van restvertekeningen. Daarom stellen we een nieuwe metriek voor die helpt bij het voorspellen van de robuustheid van de getrainde modellen bij het inzetten. In het derde deel onderzoek ik manieren om reinforcement learning agenten, die natuurlijke kandidaten zijn voor langetermijnoptimalisatie van dynamische aanbevelingssystemen, aan te passen aan het aanbevelingsscenario. In het bijzonder stellen we een agent zonder aannames voor die effectief en efficiënt leert om een evenwicht te vinden tussen nauwkeurigheid en diversiteit van aanbevelingen, om de betrokkenheid van gebruikers op de lange termijn te maximaliseren. Vervolgens onderzoek ik distributieve reinforcement learning agenten die beter in staat zijn om de grote onzekerheid te vatten die meestal voorkomt in aanbevelingsscenario's en stellen we een effectieve maar theoretisch verantwoorde updateregel voor distributief versterkingsleren voor.